UCAS 7

Proceedings of the Seventh International

Workshop on Unique Chips and Systems
UCAS-7

Held in conjunction with

The 18th International Symposium on
High Performance Computer Architecture (HPCA)

February 26, 2012
New Orleans, Louisiana, USA

UCAS-7 Organizing Committee

General Chairs

Byeong Kil Lee, The University of Texas at San Antonio
Dhireesha Kudithipudi, Rochester Institute of Technology
Tor Aamodt, University of British Columbia

Technical Program Committee

Nak Woong Eum, ETRI
Xin Fu, University of Kansas
Paul Gratz, Texas A&M Univ, College Station
Jie Han, University of Alberta
Jaehyuk Huh, KAIST
Ali Irturk, UCSD
Canturk Isci, IBM TJ Watson
Eugene John, UTSA
Dimitris Kaseridis, ARM
Tejas Karkhanis, IBM TJ Watson
Omer Khan, Univ of Massachusetts Lowell
Yong Bin Kim, Northeastern University
Shigeru Kusakabe, Kyushu University
Erik Lindholm, Nvidia
Chen Liu, FIU
Ingyu Lee, Troy University
Mike O'Connor, AMD
Resit Sendag, University of Rhode Island
Michael Shebanow, Nvidia
Abbas Sheibanyrad, TIMA Laboratory, France
Lei Wang, University of Connecticut

Table of Contents

Session III: Computer Architecture

Time: 3:20 PM- 4:40 PM (25 minutes- Full paper, 15 minutes-WIP)

Session I: High Performance Computer Architecture

Integrated Security for System-on-Chip Architectures

Stanley Bak
Department of Computer Science
University of Illinois at Urbana-Champaign
sbak2@illinois.edu

Jonathan Heiner
Computing Architectures Branch
United States Air Force Research Laboratory (AFRL-RI)
jonathan.heiner@rl.af.mil

***Abstract*—Managing access to information inside computer systems is vital to permit authorized sharing of hardware, without inadvertently permitting unauthorized sharing of data. Current approaches to managed information sharing, such as Multiple Independent Levels of Security (MILS), leverage on processor virtualization to provide information security. However, system-on-chip (SoC) architectures, which have multiple processing cores and other, non-processor computing elements, can not properly enforce security policies with processor virtualization alone.**

In this paper, we propose extending this virtualization property to other processing elements in the SoC architecture, so each one appears as though it is dedicated, but is actually shared over time. We develop the design of the Coarse-Grained Security Tagged Architecture (CG-STA), which uses this approach to reliably enforce information disclosure policies. A 16-core CG-STA prototype system, which includes shared, non-processor hardware elements, is implemented on a Xilinx ML501 FPGA Evaluation Platform. The prototype demonstrates the feasibility of the design, the security features it allows, and the trade-offs with the approach.

I. Introduction

Due to multitasking requirements, modern computers must share hardware resources among various computational processes, making them vulnerable to unauthorized information disclosure. Multitasking allows tasks and users with different security considerations or principles to perform computations on a single system through the sharing of the system's computational and storage resources. This sharing is essential to keep costs down, as each user does not need to buy a fully isolated computer. However, it is also potentially dangerous as information may be transferred, on purpose or inadvertently, in violation of the security policy. One crucial challenge is, therefore, to properly manage information access inside computer systems, so that authorized sharing of hardware is permitted, but unauthorized data disclosure is impossible.

In this paper, we consider embedded systems which incorporate a system-on-chip (SoC) design, where several cooperating processing elements, not restricted to a single CPU, are placed on the same physical chip. The need for SoC designs is driven by hardware/software co-design, reconfigurable computing, and the rapid adaptation of multicore technology.

Since SoC systems can also be shared among various security principals, approaches to properly manage information access are required. For example, a government-issued cell phone can reasonably be expected to process information flows of varying sensitivity on multiple processing elements within a single device. However, traditional information-security solutions often focus solely on information flows within a single logical CPU, creating a need for novel solutions to controlled information disclosure in embedded SoC architectures.

To address this need, we propose the Coarse-Grained Security Tagged Architecture (CG-STA). In this architecture, shown in Figure 1, each hardware component, such as a processor core, DSP, or memory unit, is associated with a tag manager. Dataflows leaving components are tagged with the current security mode of the underlying component. Incoming flows are then filtered to only permit components to receive data when allowed by the system-wide security policy. The security policy can, in this way, dictate and restrict the rights of each component to observe and communicate with other components. With an isolation-based security policy, for example, not only the processor appears as a dedicated resource that is actually shared over time (as with traditional virtualization), but every active component in the system also has this same property. By working only on input and output dataflows, our approach does not require a fine-grained understanding and verification of the dataflows through the internals of every hardware component, but can still provide useful control of information sharing.

The main contributions of this paper are:

- An SoC template architecture, the Coarse-Grained Security Tagged Architecture, which permits authorized sharing of hardware, but makes unauthorized data disclosure impossible;
- Analysis of the CG-STA in terms of added design complexity, flexibility, and trusted computing base requirements;
- The implementation of a 16-core hardware prototype CG-STA system, along with discussion of the challenges and limitations of our design.

This paper is organized in two main parts. After reviewing related work in Section II, Section III outlines the design and justification of the proposed CG-STA, starting from a clean-

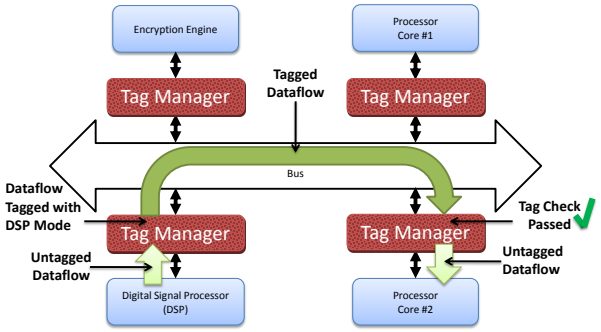


Fig. 1. A dataflow from the DSP to Processor Core #2 is transparently tagged and filtered by the Tag Managers to comply with the system-wide information disclosure policy.

slate design of a secure, multi-processing-element system. This is followed by the second part of the paper, Section IV, which describes the FPGA-based implementation of a 16-core CG-STA prototype. Challenges unique to CG-STA computing are presented and the future course for the research is then laid out. The paper finishes with conclusions in Section V.

II. Related Work

Classical approaches for information security focus primarily on securing data flows on processors, and do not extend easily to SoC architectures with multiple shared processing elements. For example, virtual machines [1] can be used to provide Multiple Independent Levels of Security (MILS) [2], which isolate computations of different security levels by using a minimalistic separation kernel.

In SoC systems, however, peripherals and other processing elements play a much larger role in the computation, and therefore must be correctly and efficiently shared. There are two common approaches to virtualize non-processor elements in a VMs: software device emulation, and hardware virtualization support. In software device emulation, device functions and access is controlled either within the hypervisor [3], or in an isolated VM (the approach taken by MILS). In addition to inflating the trusted computing base, this approach incurs significant overhead since it must emulate, in software, interfaces to processing elements. The second approach, hardware virtualization support, involves modifying the hardware of the components themselves to support virtualization [4]. Such modifications are likely to be extensive, device-specific, and must be correct for information security to be enforced. The proposed CG-STA, on the other hand, demands only a small hardware component called the Sterilize Mechanism to be information-secure, which incurs less overhead during operation than software device emulation, while being simpler and therefore easier to verify than hardware component virtualization. The complexity of the Sterilize Mechanism is further discussed in the context of our hardware implementation in Section IV-B.

A related approach for embedded systems is TrustZone [5] from ARM. In this approach, a single bit is used to partition the system into secure and nonsecure components. The CG-STA's tagging mechanism offers more flexibility than this approach and can therefore permit more complex security policies, as discussed in Section III-B.

The CG-STA is related to tagged architectures. In a tagged architecture, data is coupled with metadata called *tags* which describe the logical type or security classification of the associated data. In traditional tagged architectures [6], [7], [8], which we consider *fine-grained* tagged architectures, couple data flowing through a processor with meta-data about its type or security level, which is then tracked and used to either enforce information-flow policies or execute type-specific instructions within the processor. For example, adding two integers with a high-security classification produces a result with a high-security clearance, which, later, can not be loaded by the processor while a low-security clearance process is running. However, when extending this approach to systems with multiple processing elements, two drawbacks become apparent. First, correct design requires an in-depth understanding of the inner workings of the processing element (traditionally the processor), which would need to be replicated for each non-processor computation element. Second, there are many special-case tag-propagation rules which must be introduced due to unintuitive component usage. In a processor for instance, XORing a register with itself, a common way to clear a register, should also clear its associated tag, instead of propagating it according to the standard XOR tag-propagation rule [9]. Obtaining an exhaustive set of these special-case rules, while still maintaining information security, is nontrivial. Our proposed CG-STA, on the other hand, only maintains tagging information on an input/output flow level (which we consider a *course-grained* tagging approach). This allows information disclosure security to be easier to implement, specify, and verify. The downside of this, though, is that each component can only exist in a single processing mode at a time, which was not a restriction for fine-grained tagged architectures.

There is also a large body of work addressing information security strictly in software. Much recent research focuses on web browser policies that attempt to isolate untrusted computation from external sources, from trusted computation originating on the host computer [10]. For example, browser extension source code can automatically be examined to detect information flows which may violate this isolation policy[11].

While this line of work is certainly more easily applicable since it is software-based, the CG-STA addresses security isolation at a lower, hardware level. This is desirable because we strive to fulfill the NEAT requirements [2] of a high-security system. A NEAT security mechanism should be **N**on-bypassable, **E**valuatable, **A**lways Invoked and **T**amperproof. By using hardware for security, we trade off the flexibility of software approaches for higher security assurance. We must make sure, however, that the proposed hardware mechanism is powerful enough to enforce desired security policies, but not overbearing, since it can not be changed. The possible policies permitted by the CG-STA will be discussed further in Section

III-B.

Lastly, another class of security attacks which is not the focus of this work deals with covert channels [12] from the proposed architectural design. Guidelines for these channels typically require offline analysis and and online measurement in order to minimize their effects [13]. Mitigation techniques exist to perform these tasks [14], [15].

III. Architectural Justification

We now motivate the design of the Coarse-Grained Security Tagged Architecture, and arrive at the design which we have implemented in the second part of this paper. First, in Section III-A, we take a clean-slate design approach to building an information-secure system, arriving at the final design of the CG-STA. Then, Section III-B discusses the types of information-security policies permitable by the mechanisms present in the CG-STA.

A. CG-STA Design

We now take a clean-slate design approach to building an information-secure system. We describe several systems iteratively, at each step relaxing some restrictions to gain flexibility without compromising security. We will describe, in order, a fully-isolated system, a semi-isolated system (with shared interconnect), a restartable semi-isolated system, and finally the CG-STA.

We start by considering a **Fully-Isolated System**, shown in Figure 2. In this type of system, components processing data with different security classifications are completely disjoint and share no data. Clearly, such a system does not permit information leakage between different security classifications. Notice that the security of this type of system does not come from the physical separation of the system parts, but rather by the lack of communication between components with different security classifications.

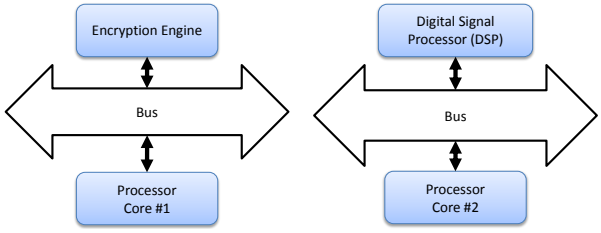


Fig. 2. In a Fully-Isolated System, computations in different security domains are performed on physically separated hardware.

An isolated communication policy is essential to the next step in the construction, which is to have a shared top-level interconnection system between various system components. This type of system, which we call a **Semi-Isolated System**, is shown in Figure 3. The essential property of the common interconnect is that it does not permit any communication between the different security classifications, and therefore the system is logically equivalent to the completely isolated system. Since we are using the shared interconnect to provide security correctness, it is part of the trusted computed base. Additionally, since it is a shared resource, care must be taken to prevent denial-of-service attacks and covert channels.

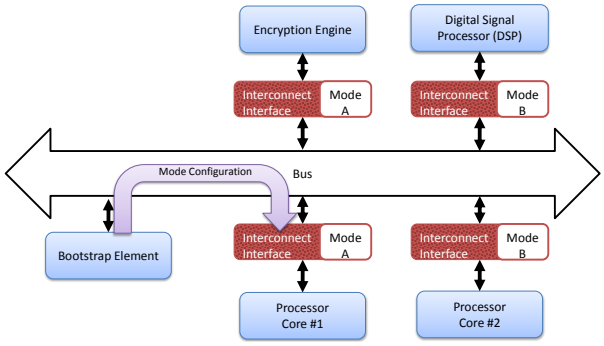


Fig. 3. In a Semi-Isolated System, computations in different security domains share a bus and their communication is tagged and filtered by Interconnect Interfaces. In the figure, components can only receive data sent by other components in the same security mode. The security modes are initially configured using the Bootstrap Element.

We have not yet disclosed the architecture of the shared interconnect, we have only described how it works at a high level. In a typical present-day architecture, one possibility for this interconnect would be something similar to the front-side bus. In some multicore and manycore systems, the interconnect could be a network-on-chip. Although possible, it is probably undesirable to use a lower-level interconnect, such as an intra-core bus. The primary reason for this is that removing the interconnect should result in physically isolated components which each run in a single security mode. If a low-level interconnect is used, a connection may still remain using the high-level interconnect (after removing the interconnect the components are not physically isolated). Additionally, some components are tightly coupled and it does not make sense to have them operate in different security modes. An example of this case could be a processor core and its local instruction store.

One remaining issue with the semi-isolated system is initial system configuration. The security modes must be assigned to the Interconnect Interfaces correctly prior to bus usage. We propose this bootstrapping is done by unique and unforgeable bus messages from a Bootstrap Element within the trusted computed base. Care must be taken by the Bootstrap Element to properly identify the components connecting to the Interconnect Interfaces. Additionally, this Bootstrap Element may send some minimal initialization data to the components, depending on the security context of the corresponding Interconnect Interface. However, this is a functional consideration rather than a correctness requirement.

The system described so far is safe from overt information leakage across security contexts, but it does not yet permit sharing of anything but the interconnect. In the next step of the construction, we permit each component to be used by multiple security levels over time (although at any one time instant each component only processes data in a single security level). Starting with our semi-isolated system, we can

run the entire system with one security configuration. In order to use a different security configuration, we can turn off the system, presumably clearing all stored internal state, and then restart the system using the Bootstrap Element to configure the system with a different mode configuration. As long as no state is saved between restarts, this **Restartable Semi-Isolated system**, shown in Figure 4, properly enforces the system-wide security policy. The restart must be initiated by a trusted component, since malicious control of this ability could lead to a denial of service. The Restarter component, which performs this task, is therefore part of the trusted computing base.

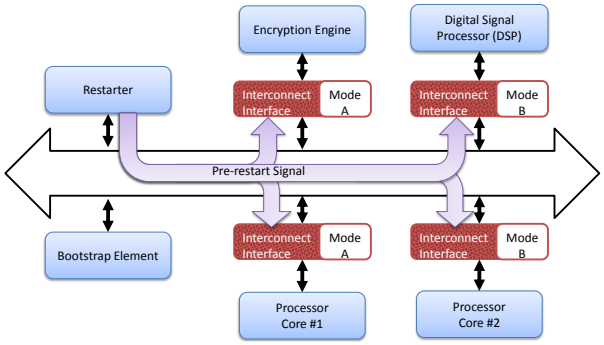


Fig. 4. A Restartable Semi-Isolated System contains a Restarter which, prior to system restart and reconfiguration, sends a pre-restart signal to all components. The components will then save their state to persistant storage (not shown), the system will be completely restarted (presumably clearly all internal component state), and the Bootstrap Element will reinitialize the Interconnect Interfaces with new security modes.

One large drawback of the approach described so far is that meaningful work may be lost when the system is restarted. To address this, we employ co-design of the interconnect hardware and the controlled components. Prior to initiating a restart, the Restarter sends a pre-restart signal to all components, indicating that the components should save their state to persistent storage. Each component can then send a feedback signal back to the Restarter when they have completed saving their state, or it will wait for a timeout to occur when the reset will be initiated anyway (in order to prevent components from causing a denial of service by delaying the restart indefinitely).

The reason this approach works and prevents information leakage is that restarting the system presumably clears the state of every component, such that information processed previously by each component is not available to the computation currently being performed by the component (unless it was accessed from permanent storage in accordance with the security policy). Notice, however, that an entire system restart is not necessary in order to have this property. Components can be restarted individually and reused, as long as their state is cleared between uses. This is the approach taken by the final Coarse-Grained Security Tagged Architecture (CG-STA).

In the **Coarse-Grained Security Tagged Architecture** (Figure 5), each component not only saves its state when instructed by the Restarter, but is also modified to include a hardware (non-bypassable) Sterilize Mechanism to reset the state of every value stored internally within the component. The Sterilize Mechanism is stronger than a traditional reset capability, which many components already support. Specifically, the Sterilize Mechanism must clear the entire accessible state, so that no prior information is available after it occurs, whereas a traditional reset need only bring the component into one valid initial state. In the Pacoblaze microcontroller, for example, a reset will set the program counter to 0, but will not clear the registers or internal scratchpad memory. The Sterilize Mechanism, however, has to do the extra work to clear all internally accessible information. The overhead and tradeoffs of the Sterilize Mechanism will be discussed further in the implementation section (Section IV-B).

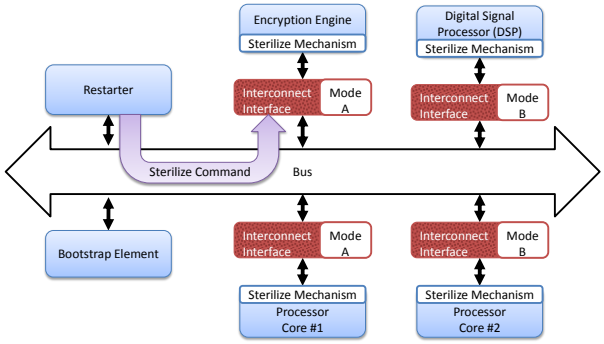


Fig. 5. The CG-STA requires that each component have a Sterilize Mechanism, which can be used by the Restarter to clear the internel state of each component during runtime security-mode reconfiguration.

Using the previously-described modules, individual components can be effectively restarted in a four-step process. First the Restarter instructs them to save their state. Then, after a save-complete message arrives or a timeout expires, the Restarter initiates the non-bypassable Sterilize Mechanism. Third, the Mode Setter assigns the reset component's Interconnect Interface a new security classification. Finally, the Mode Setter sends the component some initialization data, which the component uses to load its state from persistant storage.

The modules necessary for CG-STA computation are described in the Table I.

While there are several modules in the trusted computing base, the majority of the system complexity is contained in the untrusted components. Fine-grained component modification to support data flow tagging, a complex and potentially error-prone process, is not necessary with the CG-STA approach.

B. Security Policy

The specific security policy enforced by the CG-STA depends on the intent of the system designer. However, certain precautions must be taken when initiating mode changes of individual components, which we will describe in this section. First, however, we outline two common security policies supported by the CG-STA.

The easiest policy to enforce is the one described in the CG-STA design section, complete isolation. If, for instance, a finite number of virtual systems are to appear to run on

Module	Description	Trusted Computing Base?
Interconnect	Provides communication among components and other modules.	Yes
Interconnect Interface	Interfaces each component with the Interconnect; automatically tags bus transactions and filters messages in accordance with the security policy.	Yes
Bootstrap Element	Sets the security policy of each of the Interconnect Interfaces; sends initialization data to the components.	Yes
Restarter	Sends a message to the components informing them to save their state; activates the Sterilize Mechanism.	Yes
Sterilize Mechanism	Clears all stored values in the corresponding component.	Yes
Components	Operates on data flows. Examples include processor cores, DSPs, dedicated memories, and custom IP cores.	No

TABLE I
ALTHOUGH SEVERAL MODULES MUST BE PRESENT FOR CORRECT CG-STA COMPUTATION, THE MAJORITY OF THE SYSTEM COMPLEXITY IS CONTAINED IN THE UNTRUSTED COMPONENTS.

the same physical system, the CG-STA can be configured to provide verified isolation. Each virtual system is assigned a unique identifier. Each Interconnect Interface tags data flows from each component with the unique identifier of the corresponding virtual system. For filtering, only messages on the interconnect with a matching unique identifier are propagated by the Interconnect Interface into the underlying component for processing. In this way, an exclusive view of the physical system is presented to each virtual system.

Another security policy the CG-STA can enforce is the Bell and La Padula security policy [16]. In this model, there are four disclosure levels of information, unclassified, confidential, secret, and top secret, and then within each level there are several mutually exclusive compartments, all of which a component must have permission to access before receiving the data. Two security rules should be enforced by the Interconnect Interfaces. First, components should only be able to read data with a lower or equal security classification. Second, components should only be able to write data with an equal or higher security classification. To enforce such a model, the Interconnect Interface would tag all data leaving each component with the security level of the component, and, when receiving, only data with a lower or equal security classification would be propagated into the component. One issue with this approach is that discretionary upclassification of data is not possible. The reason for this is that the tags on the interconnect are transparent to the underlying components, so the security policy and its enforcement also becomes transparent. Discretionary upclassification, however, requires a security reclassification action to be initiated by the component. If discretionary upclassification of data is desired, the Interconnect Interfaces could be designed to perform some limited introspection of the outgoing data which would contain of an indication of the intentions of the underlying component (the intention to send the data at a higher classification level). If a higher security classification is indicated in the data, this would then affect the outgoing tag sent on the bus by the Interconnect Interface. Notice, however, that this requires adding complexity to the logic of the Interconnect Interface, which is part of the trusted computing base of the system, and would therefore have to be done with care to maintain the correctness of the system.

With all security policies, there are practical considerations due to the restart and sterilize procedures in the CG-STA. We describe two of these: potential for data loss during mode switches, and the consistent appearance of the system to each virtual system.

In most systems, data delivery on the front-side bus is assumed to be reliable. However, if the Sterilize Mechanism of the receiving component of a bus transaction is activated before the sending component, messages could effectively be dropped. For example, if a peripheral is sterilized before a component processing the peripheral's interrupt, the interrupt acknowledge bus message may not be observed arriving at the peripheral. For this reason, sending components should be sterilized before receiving components. Unfortunately, there is not always such a clear distinction among components, as system-wide communication flows rarely form a directed acyclic graph (DAG). However, recall that before activating the Sterilize Mechanism, components receive a save-state, pre-restart signal from the Restarter. The timeout after the pre-restart message before forced activation of the Sterilize Mechanism should be long enough such that the components can first enter a consistent state without any lingering communication, and then save their state to persistent storage.

An additional consideration is the view of the system presented to each virtual system. When components are restarted and their state is restored, it may be expected that the physical system has not changed and therefore the same components which were previously available are still active. Although components can be designed with plug-and-play support, which would allow components to appear as suddenly removed from the system, policies doing this would need to be co-designed with the underlying processing elements.

IV. HARDWARE CG-STA PROTOTYPE

Based on the CG-STA design developed in Section III, we have implemented a multicore CG-STA prototype in order to evaluate critical aspects of the design. First, we will present details about the overall architecture implementation. Next,

we describe the tradeoffs associated with the component's Sterilize Mechanism. Finally, we discuss implementation observations and their implications for CG-STA and tagged computing.

A. Implemented Components Overview

Our prototype CG-STA was developed on a Xilinx ML501 Evaluation Platform, which features a XC5VLX50 FPGA. On top of this FPGA, we programmed the logical architecture shown in Figure 6. The implemented architecture has a parameterized number of Pacoblaze cores connected to the bus, which has been verified as working with up to 16 independent processing cores.

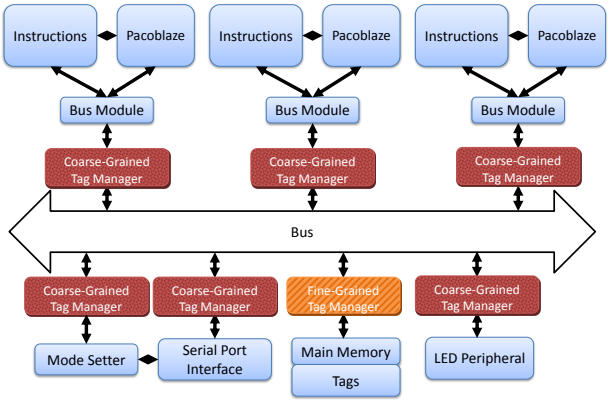


Fig. 6. The implemented CG-STA contains components that communicate through tag managers.

In our prototype (Figure 6), several types of components were implemented in order to have a functioning CG-STA system. Since we desired to have an external access to the system, we implemented a Serial Port Interface component to use as the main external communication mechanism. Through instructions from the serial port, the Mode Setter acts as the CG-STA Bootstrap Element and the Restarter. Messages sent from components must pass through an Interconnect Interface, either coarse-grained in the case of single-mode components, or fine-grained in the case of concurrently shared components such as the Main Memory Module. An LED Peripheral is present to visually display output which controls 8 LEDs on the ML501. Each Pacoblaze core is coupled with a bus module for interfacing with the system bus, and an associated instruction store. Upon receiving a sterilize command message from the Mode Setter, each Interconnect Interface initiates its component's Sterilize Mechanism.

B. Sterilize Mechanism

One main architectural requirement of the CG-STA is that each component be modified to include a Sterilize Mechanism capable of resetting the component's visible state. One of the goals of the implementation was to evaluate the effort required to add this mechanism to existing components, and the overhead it required both in terms of area and in terms of performance. Area overhead versus performance is a trade off, with the most direct design of a Sterilize Mechanism consuming more area.

Given the HDL source code, a direct Sterilize Mechanism can be added in a way similar to a synchronous reset, with the additional condition that every register in the design be reset to a constant value. This was the approach taken for modifying the Pacoblaze processor core. This process should be done recursively for all components internal to the design (in the Pacoblaze design it was also done for the ALU, register file, and all other internal modules). The main benefit of this approach is that it requires minimal understanding of the underlying component design and is therefore easy to do (we did not have to understand the internal operation of the Pacoblaze in order to add the Sterilize Mechanism). Additionally, tools could be made which scan the HDL for Verilog `reg` statements and VHDL `signal` statements to check that every register is in fact sterilized, and that all internally instantiated components also have a correct Sterilize Mechanism. Such automated checking would be significantly benefitial for architectural validation, since the Sterilize Mechanism is part of the trusted computing base in the architecture and existing components will likely need to be augmented in order to incorporate a Sterilize Mechanism. The drawback of this approach is that it may consume significant area, especially if the component has many registers which maintain a significant amount of state information. This is especially a problem for components whose main function is to store data, such as the instruction store or memory peripheral, so a different approach to designing their Sterilize Mechanism was taken.

The approach taken to design the Sterilize Mechanism for memory components is to trade off performance in order to reduce circuit area. For a memory module, it would be inefficient to reset the entire memory in a single cycle. Instead, a multiplexer is used for the write source into the memory block, and the sterilize signal starts a simple state machine which outputs zero to incrementally-increasing addresses in the memory block. In terms of Xilinx-specific FPGA considerations, this allows BRAM blocks to still be used to implement memory instead of flip flops, which saves significant area. However, the downside is that the Sterilize Mechanism is less efficient now, taking multiple cycles to fully sterilize the memory. For a fine-grained tag manager, such as the one used for the Main Memory Module, tags can be reset (which resets the memory) on a smaller granularity, which incurs less overhead than clearing the entire memory.

This area-time tradeoff is not a binary decision. Modern memory systems contain multiple banks which can be concurrently written. A sterilize implementation could, therefore, clear all the memory banks in parallel, which would save area over a one-cycle clear, but still be more efficient than a serial write over the entire memory space.

C. Architectural Observations

Over the course of implementing the CG-STA, several important observations were made. In this section we briefly

describe each of these, since they may have implications on future tagged architecture research.

First, we noticed that instructions do not map directly into a security compartment. When tracking information flow through a processor, it is clear that one of the inputs is the instruction, so therefore the processor's security classification should include the *instruction* compartment. However, the output of the processor is usually not instructions, so the output flow should *not* be tagged with the instrcution compartment. In a pure MLS system, this is a declassification of the instructions, since, strictly speaking there is a possibility they could be inferred from the output flow of the system, and is therefore a violation of the security policy. The solution in the CG-STA prototype is to employ different security modes for reading and writing. The check for the compartment instruction, then, does not occur when the bus module is receiving bus data, but is instead internal to the processor component. Specifically, the check occurs when data is written to the instruction store. This solution creates a hierarchy of tagging. At the top level, the Coarse-Grained Tag Managers tag security levels and enforce coarse-grained information-flow security. At the lower level, the processor itself does some additional tagging and checking on input / output flows, so only data tagged as instructions are allowed to be executed. This approach makes sense, since the instruction compartment requirement is specific to the processor core and not a generalized part of the architecture.

Another observation which may have implications for other tagged systems is that tagged main-memory writes incur additional overhead when compared with a non-tagged approach. This is because tagged memory writes must first read the tag of the memory address to which they are writing, to make sure they are not overwriting data with a higher security tag (which, if allowed, could be used to corrupt data and cause a denial of service). This effectively halves the possible write speed. This additional latency can not be avoided even if the tags are sent in parallel on the bus. Along this line, storing tags for every byte in addition to data has the effect of halving the usable memory size. One option here is to use a single tag for groups of data words instead of each tagging each word individually. In the case of a parallel tag bus, there is no extra overhead for performing tagged reads.

The next architectural observation is that the Sterilize Mechanism in the CG-STA may cause issues with external components. For example, if there is timing synchronization between an internal component and an external component, activating the Sterilize Mechansim of the internal component ruins this synchronization (since its internal state is cleared). This was apparent when the Serial Port Interface component was reset. When the Sterilize Mechanism was actived, the serial port hardware would lose clock synchronization with the host computer and the first byte sent after activating the Sterilize Mechanism was usually corrupted (after the first few bytes were sent, synchronization was restored). Also, if there was an internal component acting as a network interface, sterilizing its internal state would not reset external routers, which may cause unexpected behavior.

Although only the Sterilize Mechanism is necessary for information security, components must save and load their states for proper functionality across resets. This need not be done entirely in hardware, as we demonstrated with our Pacoblaze microcontroller. The software in our prototype would not only save its own state, but would also load and save the state of the LED peripheral. This requires, however, that all important component state is visible to the software so that it can be loaded and saved. Additionally, this introduces overhead when performing resets.

System-Wide Virtualization is related to IOMMU technology [17]. Using an architecture with an IOMMU, where peripheral I/O goes through a memory management unit, seems to be an intermediate solution to the information security problem. In an IOMMU architecture, external peripherals are isolated from the processor, but peripherals are not isolated from each other. In this sense, it is a hybrid solution between System-Wide Virtualization and no protection. With the CG-STA, security policies are possible which limit peripheral interactions to those allowed by the system's security policy.

Another observation is that compartment tags in our prototype are handled in a global, hardcoded way. Ideally there would be a dynamic tagging manager which could distribute unique tags to the Mode Setter and cooperating components during runtime, which would produce a cleaner implementation. If persistent tags across restarts are desired, the dynamic tag manager would have to take this into account when distributing tags with new compartments. This would allow components to request a tag with a new, unique compartment for an isolated execution of the system. The particular bit value of the compartment they get, however, is unimportant.

Finally, we present a list of features essential to general tagged architectures. In a tagged architecture, three types of components are necessary: (1) components which produce the tags, (2) components which move tags around with their associated data, and (3) components which perform operations and checks with the tags. The reliable functionality of the tagging mechanism depends on all three of these components. In the CG-STA, tags are created by the Interconnect Interfaces (configured by the Mode Setter). Tags are moved around with their associated data by both the bus, and, on a broader time span, using the fine-grained tagged Main Memory Module. The operations performed on the tags are bus-transaction filtering, which are also performed by the Interconnect Interfaces (configured by the Mode Setter). In order to attack the tagging mechanism of the CG-STA, one of these components would need to contain a vulnerability. In traditional tagged processors, tags are created based on manually assigning security classifications to input datasets, and performing processor operations on data. Tags are moved around within the processor, onto buses, and into memory and persistent storage. The tag operations, mostly security checks, are performed within the processor. If any one of the components dealing with tag creation, movement, or checking is incorrect, the functionality of the tagging can be circumvented.

Much research on tagged architectures, however, is con-

cerned with either tag creation or tag checking, and less emphasis is placed on tag movement through the system. Tag movement should guarantee that data stays coupled with its description tags. This process has many avenues of attack. For example, if separate buses transport the data and the tags, it is possible that the tag bus experiences a parity error and has to resend, while the data bus successfully completes the transaction. This may decouple the data from the bus for a short time. Alternatively, if the data and tag are sent in back-to-back bus transactions, rebooting the machine between bus transactions might violate tag movement requirements. If the tags can be arbitrarily overwritten in memory, they can be decoupled from their data. If tags are stored in persistent storage, which can be accessed externally, tags may be decoupled from data. Keeping tags coupled with their data throughout the architecture is just as important to the reliability of the tagging mechanism as having correct tag propagation rules, or correctly classifying input data.

D. Future Extensions

The current CG-STA was demonstrated as functional on a 16-core Pacoblaze prototype. The next immediate step for CG-STA research is to modify the arhcitecture to be closer to a deployment system. Specifically, immediate steps to take would be to use a 32-bit microprocessor such as the Microblaze or LEON3 processor with proper preemption instead of an 8-bit microcontroller. The overhead of the Sterilize Mechanism on these processors, in terms of area, timing, and possible effect on critical path, would be more interesting to evaluate. With such processors, a high-end bus would be required, and new tag managers would need to be developed to provide this interface. An evaluation of benchmark applications running on such an off-the-shelf processor, instead of programs designed spefifically for our architecture, is also needed. Additionally, rather than having embedded programs with direct access to physical memory, running the CG-STA with full operating systems on the cores may present additional challenges and require unique solutions. Another desired extension is to provide a disciplined way to interact with external components, such as network cards and other off-chip peripherals, that would not cause unintended behavior after activating the Sterilize Mechanism.

V. Conclusions

We have designed, evaluated, implemented a CG-STA system-on-chip architecture to provide integrated security for embedded systems. This was done by designing system components which maintain information about their current security principal, enforce flows between components, and require components to sterilize, or flush, their entire internal state before switching security principals. A 16-core CG-STA prototype was implemented, verified, and evaluated on top of a Xilinx ML501 FPGA.

The CG-STA approach has both drawbacks and benefits. The primary drawback is that each computational element can only be in one security mode at any point in time. A single component is not allowed to concurrently process data in different security domains. Additionally, hardware components do need minor modifications to be able to sterilize their internal state, as well as load and save important values across state flushes. This may not always be possible since processing elements may not be distributed as HDL code. If this is possible, however, there are significant benefits to the CG-STA approach. The design can take advantage of the dissemination of computation from a single processing core to distributed computation engines, by permitting authorized hardware component cooperation. The design also disallows, on the hardware level, malicious components from being able to intercept or interfere with computations of incompatible security classifications. The modifications necessary to the components to make them CG-STA-compatible are minor, requiring little additional component engineering effort. Most importantly, the architecture is simple to understand and therefore simple to verify as information-secure. In high-security embedded SoC devices, the CG-STA design provides a non-bypassable, built-in security mechanism.

References

[1] S. E. Madnick and J. J. Donovan, "Application and analysis of the virtual machine approach to information system security and isolation," in *Proceedings of the workshop on virtual computer systems*. New York, NY, USA: ACM, 1973, pp. 210–224.

[2] G. M. Uchenick and W. M. Vanfleet, "Multiple independent levels of safety and security: high assurance architecture for msls/mls," 2005.

[3] R. Vanover, "Peripheral virtualization over ethernet," virtualizationreview.com/blogs/everyday-virtualization/2010/05/peripheral-virtualization-over-ethernet.aspx, 2010.

[4] Intel, "Intel virtualization technology for connectivity," www.intel.com/network/connectivity/solutions/virtualization.htm, 2010.

[5] T. Alves and D. Felto, "Trustzone: Integrating hardware and software security," www.iqmagazineonline.com/magazine/pdf/v_3_4_pdf/Pg18_24_custZone_Secur.pdf, 2004.

[6] Burroughs61, "The descriptor a definition of the b5000 information processing system," Bull. No. 5000-20002-P, 1961.

[7] D. A. Moon, "Architecture of the symbolics 3600," *SIGARCH Comput. Archit. News*, vol. 13, no. 3, pp. 76–83, 1985.

[8] E. F. Gehringer and J. L. Keedy, "Tagged architecture: how compelling are its advantages?" *SIGARCH Comput. Archit. News*, vol. 13, no. 3, pp. 162–170, 1985.

[9] M. Dalton, H. Kannan, and C. Kozyrakis, "Raksha: a flexible information flow architecture for software security," in *ISCA '07: Proceedings of the 34th annual international symposium on Computer architecture*. New York, NY, USA: ACM, 2007, pp. 482–493.

[10] C. Reis, A. Barth, and C. Pizano, "Browser security: lessons from google chrome," *Commun. ACM*, vol. 52, pp. 45–49, August 2009.

[11] S. Bandhakavi, S. T. King, P. Madhusudan, and M. Winslett, "Vex: Vetting browser extensions for security vulnerabilities," 2010.

[12] B. W. Lampson, "A note on the confinement problem," 1973.

[13] D. C. Latham, "Department of defense standard department of defense trusted computer system evaluation criteria," 1985.

[14] R. A. Kemmerer, "A practical approach to identifying storage and timing channels: Twenty years later," in *ACSAC '02: Proceedings of the 18th Annual Computer Security Applications Conference*. Washington, DC, USA: IEEE Computer Society, 2002, p. 109.

[15] W.-M. Hu, "Reducing timing charmers with fuzzy time," *Security and Privacy, IEEE Symposium on*, vol. 0, p. 8, 1991.

[16] D. E. Bell and L. J. Lapadula, "Secure computer systems: Mathematical foundations," 1973.

[17] D. Abramson, "Intel virtualization technology for directed i/o," Intel Technology Journal, 2006.

Parallelizing Electroencephalogram Processing on a Many-Core Platform for the Detection of High Frequency Oscillations

Gildo Torres, Paul McCall, Chen Liu, Mercedes Cabrerizo, Malek Adjouadi
Department of Electrical and Computer Engineering
Florida International University
Miami, Florida, USA
E-mail: {gtorr029, pmcca001, cliu, cabreriz, adjouadi}@fiu.edu

Abstract— **EEG high frequency oscillations, known as ripples, in subdural Electroencephalography (EEG) have been associated to the seizure onset zone (SOZ). Ripples, which can be visible in the frequency range from 80 to 250 Hz, are considered reliable biomarkers (like interictal EEG spikes) to identify the epileptic focus in the brain. Consequently, an automated detection method is proposed with the aim of identifying those electrodes that have a higher count of these events. The computational approach considered in this paper relies on processing the EEG records using the Intel Single-Chip Cloud Computer (SCC) platform. This new method preserves data coherency through the message-passing interface and utilizes dynamic voltage and frequency scaling (DVFS) capability of SCC, yielding both energy saving and performance benefit. The proposed SCC-based method for detecting high frequency oscillations (HFO) is validated by EEG experts at Miami Children's Hospital, and the location of the electrodes with higher counts will be compared with the 3-D source localization using interictal spikes to demonstrate the relation if any that exists between them.**

Keywords: Cloud computing, Intel SCC, EEG processing, High frequency oscillation, Power-aware computing

I. INTRODUCTION

Epilepsy is a common medical condition characterized by a predisposition to unprovoked recurrent seizures. A seizure is the manifestation of an abnormal, hypersynchronous discharge of a population of cortical neurons [1]. Affecting over 60 million people around the world, Epilepsy is the second most frequent neurological disorder other than stroke.

Advanced clinical techniques are used to diagnose epilepsy, such as computed tomography (CT), encephalogram (EEG), magnetic resonance imaging (MRI), positron emission tomography (PET), and functional MRI, along with others. While the aforementioned techniques yield a coarse approximation of the epileptogenic region, they otherwise lack either the spatial or temporal resolution necessary to accurately determine the seizure focus location. When this is the case, invasive recording techniques such as intercranial-EEG (iEEG) or Electrocorticography (ECoG), which are characterized by the placement of electrode arrays on the cortex of the brain, are performed.

During ECoG recording it is routine clinical practice to place multiple electrode arrays on different areas of interest to the neurologists. Multiple arrays act to both eliminate certain cortical regions of interest and designate cortical areas where more analysis may be needed. For example, it is customary for a patient undergoing ECoG to have sixty-five or more implanted electrodes.

Within Epilepsy research, many relevant algorithms are useful throughout the process of localization of the seizure onset zone (SOZ). These include interictal spike detection, seizure onset detection, artifact detection and elimination, high frequency oscillations (HFO) detection as well as many others [2-6]. For HFO detection, patients are monitored throughout the night and sleep ECoG is recorded for up to 10 hours. What's more, high resolution clinical-use EEG machines have an eclipsing sampling rate of 2KHz. Due to these characteristics, it can be seen that any extensive time recordings will yield large amount of data which requires enormous computing power.

In the last several decades, we have seen how microprocessor performance have been dramatically improved by increasing the operating frequency, from 5MHz of Intel 8086 to the astounding 5.2GHz of IBM z196 [7]. Unfortunately, in recent years, power-thermal issues have limited the pace at which processor frequency can be increased [8]. In an effort to utilize the abundant transistor real estate offered by the Moore's Law [9] and at the same time contain the power-thermal issues, current developments in microprocessor design favor increasing core counts over frequency scaling to improve processor performance and energy efficiency [10].

In the commercial field, it is common to have a 2, 4, 6 or even more cores housed in one chip nowadays; while the research community makes use of experimental many-core architectures containing tens or even hundreds of processors. Today, the challenge is not only how to develop powerful hardware architectures that satisfy the demands of high resources-consuming applications, but also the development of applications that could effectively explore the capabilities offered by many-core architectures. There are major benefits that can be obtained from parallel programs running on many-core platforms.

The Single-Chip Cloud Computer (SCC) experimental processor [10] is a 48-core 'concept vehicle' created by Intel Labs as a platform for many-core software research. This system allows the implementation and study of parallel applications by supporting a message-passing programming model for communication among the cores. The SCC also includes hardware elements which support dynamic voltage and frequency scaling (DVFS) for improving energy efficiency.

As we mentioned earlier, ECoG recording and HFO detection would not be easily processed by desktop computers or even specialized software. Due to the limitations of the processing platforms that are available to researchers in this field, currently only a sub-section of this data, ranging between a few seconds to a few minutes, will be subjected to analysis while the rest of the data is discarded. In this paper we will illustrate the benefits of utilizing the SCC as the platform of choice for EEG algorithm implementation, by demonstrating the energy savings and computational benefits associated with Intel's 48-core Single-chip Cloud Computer.

The rest of the paper is organized as follows. In Section II we present an overview of the SCC platform. Section III contains the methodology we propose for processing EEG data on the Intel SCC platform and Section IV describes the algorithm implemented for the detection of HFOs. In Section V we present and comment on the obtained results. We conclude in Section VI and give an outlook for future work.

II. Computation Platform

The Single-Chip Cloud Computer contains 48 Pentium™ class IA-32 cores on a 6×4 2D-mesh network of tiled core clusters with high-speed I/Os on the periphery [10]. There is a unique hardware feature called Message Passing Buffer (MPB), shared by every two cores, that is optimized to support message passing programming model to communicate among all the cores.

The SCC platform used in this research can be considered a computational benefit to almost any parallel application performed on it due to the fact that it possesses 48 cores. However, there are two main benefits that make the SCC platform suitable for EEG signal processing: its inter-core communication or message-passing abilities, and the capacity for DVFS. These two aspects of the SCC make EEG processing a promising application because they address two problems that are inherent to processing of this type.

The first difficulty that occurs when processing EEG data is that a significant amount of time and energy is consumed upon the access and distribution of the data. This problem is magnified within a parallel architecture [11]; while a few cores are accessing and loading the data, many of the cores are running at the same power levels without contributing to the overall progress. As shown in Figure 1, every two cores form a frequency island and every eight cores form a voltage island. The SCC allows the user to fully control these voltage and frequency islands that are present on the chip.

In this way we can act to minimize wasted energy by setting cores to lower power states while accessing large amounts of data. A similar approach has been demonstrated by the use of multiple voltage-frequency gears that run at different segments throughout the program in order to maximize performance while saving energy in a PC cluster setting [12].

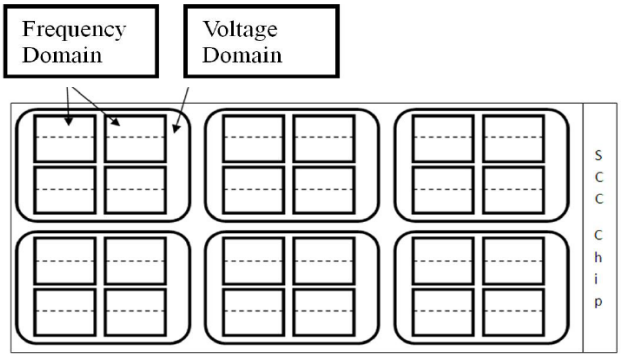


Figure 1. Frequency domains and voltage domains on SCC chip

The second issue with EEG data is that there tends to be dynamic global parameters for most algorithm implementations. This means that the processing of one electrode may be dependent on a parameter defined by another electrode or a group of electrodes. This is due to the aggregate nature of the EEG signal itself. The EEG signal stems from a summation of neuronal activity; therefore a single phenomenon may have components in many surrounding electrodes. A program which distinguishes a particular activity of interest, such as interictal spikes, may need information from numerous electrode signals in order to confidently detect their presence within the data set. This establishes a need for effective and user-controllable inter-core and thus inter-electrode communication. When such algorithms are run in a parallel manner, this becomes a more detrimental issue. Without explicit user defined communication protocols, there would potentially be cache coherency issues and/or memory allocation issues due to variables growing inside loops. The SCC allows the user, through use of the RCCE library [13] (an API library for message passing programming model specially designed for SCC), the ability to control and synchronize inter-core communication due to the message-passing benefits of the SCC platform.

III. Methodology

The proposed method for implementing EEG algorithms on the SCC platform is shown in Figure 2 and outlined below.

1. Electrode dependencies and any need for global parameters are identified. This knowledge will lead to an understanding of the appropriate inter-core communication that will be necessary for proper execution.
2. Once inter-core communication is understood, the program needs to be broken down into segments. These

segments are divided into two categories, communication-intensive and computation-intensive. Parts of code that accesses the data and distributes appropriate data to proper cores would be considered communication-intensive while parts of code that utilize many processing cores for filtering or detection of any kind would be considered computation-intensive. These segments would then be run using the appropriate voltage-frequency gear that would result in performance benefits while maintaining energy savings.

3. Once there is an awareness of the necessary communication between cores and the code has been segmented and assigned correct gears, the code needs to be modified with the correct RCCE library functions so as to be implemented and executed on the SCC platform.

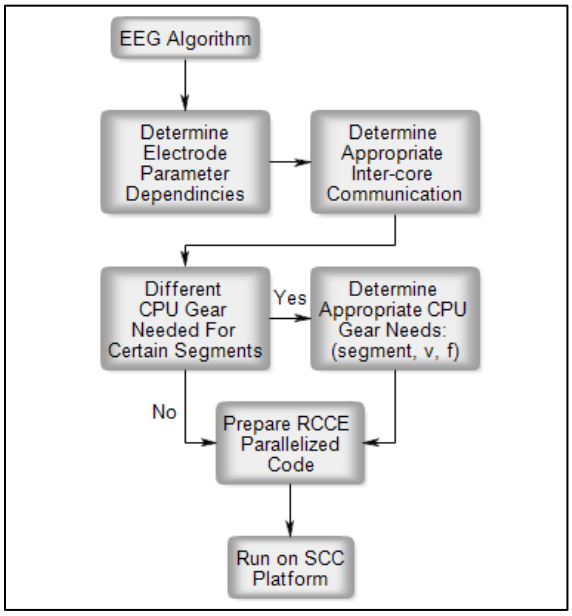


Figure 2. Proposed EEG-SCC Methodology

The EEG data and results that are presented within this paper are gathered from a patient who was monitored overnight with multiple electrode arrays placed on his cortex totaling sixty-five electrodes at the Brain Institute in Miami Children's Hospital (MCH) [14]. In the validation of this algorithm, 10-minute segments were analyzed consisting of 32 electrodes. This number of electrodes was analyzed for each run of the simulation because this is a preliminary analysis of EEG processing on the SCC platform. Processing of more electrode data will be an interesting avenue for future work

IV. HFO – EEG Algorithm

HFOs have been defined as spontaneous patterns in the range of 80 – 500 Hz that consists of at least 4 oscillations which can be distinguished from the background. However, this is not a quantitative definition, thus making accurate detection of HFOs is both difficult and subjective. HFOs can be visually marked, but this process tends to be highly time consuming, on the order of hours for the analysis of a few minutes of data [15]. Research has suggested that HFOs are possibly related with epileptogenesis [16]. Electrodes of interest, which correspond to the SOZ, have higher relative ripple counts when compared with electrodes that are associated with other cortex regions of normal neuronal activity [17, 18].

The definition used in this paper for the HFO detector is listed below and illustrated in Figure 3.

- HFOs are within the 80 – 250 Hz frequency band,
- A global threshold based on standard deviation of a selected electrode is determined,
- Three or more crossings of the global threshold within a 250 ms window will count as a HFO.

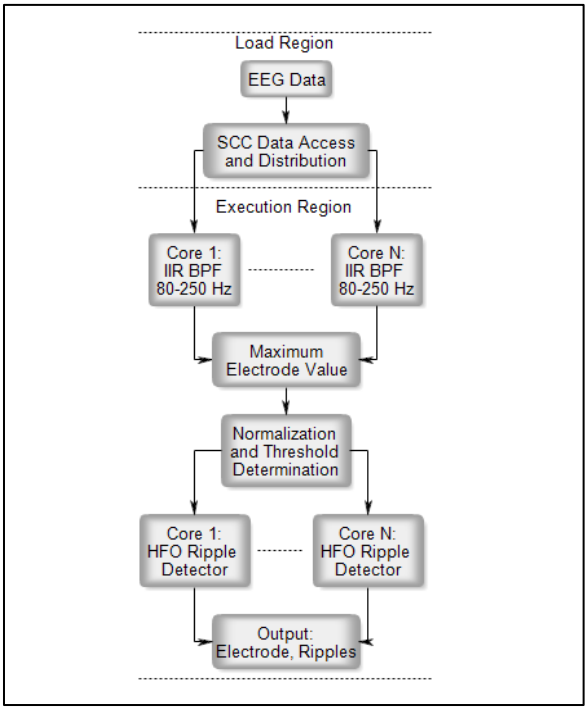


Figure 3. HFO Detection Program Executed on SCC

Raw EEG data, acquired at a sampling rate of 2 kHz, is passed into the SCC program, parsed up and distributed to the appropriate cores. Upon receiving the data, each core implements a 10th-order Butterworth IIR filter using cascaded Second Order Direct Form II sections. Once the filtering process is completed, a global maximum value is taken across all electrode signals. This electrode-max value is used to normalize the entire set of electrode readings. A sample segment of data is shown in Figure 4 after filtering and normalization.

After normalization, a standard deviation calculation is performed on the electrode signal that produced the electrode-max value. The global threshold defined for the program is calculated as a multiple of this standard

deviation value. This threshold parameter shows that even for the simplest EEG algorithms, such as a rudimentary HFO detector, there exists a fundamental need to efficiently process and pass data between cores while the program is analyzing the data.

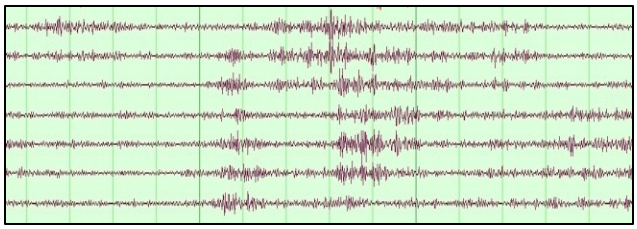


Figure 4. Filtered and Normalized Data

After the electrode data is normalized, the data is compared point-to-point with our global threshold, which is determined as three times the aforementioned standard deviation value. Threshold crossings are highlighted and stored as possible spikes. If three or more threshold crossings occur within any 250 ms window, that window is determined to include one high frequency oscillation. The program then moves to the end of the window and continues to examine the data for threshold crossings.

V. RESULTS

This work acts to demonstrate the feasibility of the HFO algorithm implementation for EEG analysis on the SCC platform. In order to verify the correctness of our implementation, relative HFO counts per electrode were compared with the findings previously validated by neurologists at Miami Children's Hospital and found to match, with all electrode signals of interest being identified. The results were further validated when electrode locations with higher HFO counts from the algorithm were compared with 3-D source localization using interictal spikes. This analysis was performed for 10-minute segments of data with all simulations taking less than 40-second of processing time. It is worth noting that when a replica of our algorithm is ran on Matlab, removing all Matlab optimized function calls, the processing time exceed thirty minutes for the same amount of data. When the program was executed including Matlab optimized function calls, the load region took in excess of 42 seconds, while the execute region took over 9 seconds. These results were obtained on a PC with an Intel core i5 processor running at 2.30 GHz and 8 GB of RAM.

As shown in Figure 3, the program is classified into the load region and the execution region. The load region includes the section of the program where the data file is being loaded and split by the master core. The data is then distributed to each processing core. This region was classified as a communication-intensive region, because the code spends most of the time accessing memory or transmitting information to other cores. Therefore a high processing frequency is not required.

The execution region, explained in the previous section, is where the ripple-detection algorithm is executed. We classified this region as a computation-intensive region, where a high processing frequency is required.

Gear	Voltage	Frequency
HIGH	1.1 V	800 MHz
LOW	0.8 V	533 MHz
MIX	0.8/1.1 V	533/800 MHz
XIM	1.1/0.8 V	800/533 MHz

Table 1. Voltage - Frequency Gears

We tested different setups of the SCC platform where the number of cores employed varied while four Frequency-Voltage configurations were used, as shown in Table 1. The frequency-voltage schemes tested were HIGH gear (800 MHz and 1.1V), LOW gear (533 MHz and 0.8V), the third and fourth ones were mixed approaches, where the low gear was applied to the load region and the high gear to the execution region in the third configuration that we called MIX, and for the fourth one the high gear was applied first to the load region and then low gear to the execution region, for what we called the XIM gear. For both mixed gears we dynamically changed the voltage and frequency values during the transition between regions. The numbers of cores tested were 1, 2, 4, 8, 16, and 32.

The elapsed time when switching between different levels of voltage and frequency is not significant compared to the time consumed by each region, therefore it has no impact on the timing results associated with either region or the total execution.

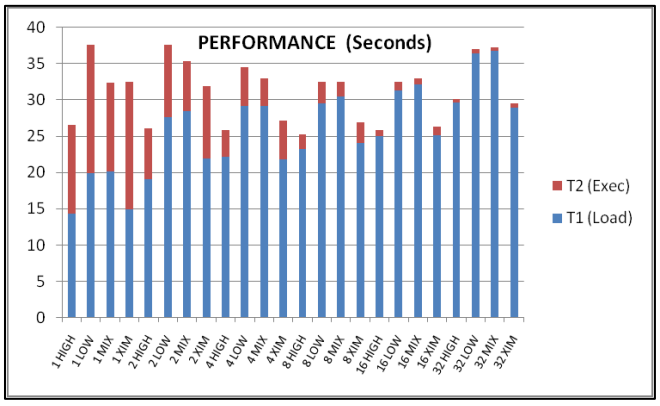


Figure 5. Performance Graph

It can be seen that total processing time for all simulation runs does not exceed 40 seconds. While this is a major improvement over serial Matlab implementation, it does not directly demonstrate the need for the parallel platform. This is because our proposed method is not specifically aimed at this algorithm implementation. The HFO algorithm is a simple computation task for the SCC, and immediate benefits appear in the DVFS and message-passing abilities

of the system. However, as the execution region becomes more burdensome, the benefits of parallelizing the program can be easily seen. We can anticipate that processing benefits associated with the SCC-based EEG analysis will become more substantial when running more complex algorithms on the platform.

From Figure 5 we can observe how significant the communication region is when compared to the total time for the simulations, with the exception of the simulations completed with one core where no communication overhead exists because all the data manipulation is local to one core. The results show the HFO detection program spends most of its execution time in communicating and transferring data among the cores. We can see as the number of cores increases, the communication overhead increases, and the time for completing the execution region decreases. Here the communication overhead has more weight than the computation overhead. Therefore, "the more the merrier" seems not work here. The implementation of the MIX gear tries to match the communication-intensive region with low power dissipation gear and the computation-intensive region with a high processing frequency in order to improve the power-energy efficiency without sacrificing performance. Based on the nature of the specific program we are running, we decided to try the XIM gear where we executed the communication-intensive region with high frequency and the computation-intensive region with the low frequency gear. We discuss our findings below.

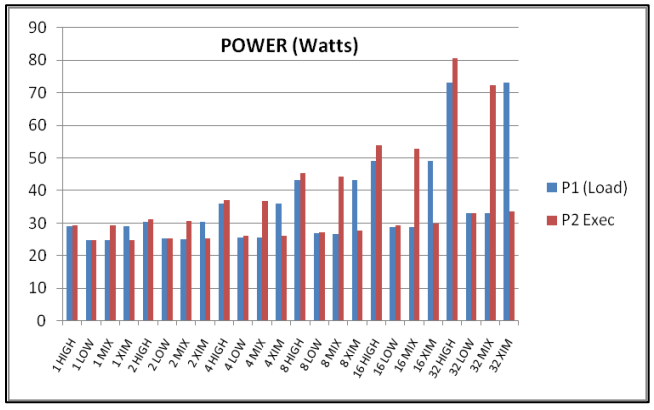


Figure 6. Power Graph

As we can see, the XIM gear provides a better performance than the MIX gear except the one-core case. The reason is for the parallel computing case, the communication overhead is so high when compared with computation time. Even though the XIM gear slows down the execution region a little bit, it reduces the time spent in load region significantly, thus having an advantage over the MIX gear overall. This is not the case for single core configuration because executing with one core is the only case where the load and the execution region performances are in similar order. This happens because in this case we avoid the overhead of data transmission and because of the prolonged time it takes for the execution region to be completed. Overall, for optimal performance, we see that the best configuration for the SCC is having 8 cores running with the HIGH gear.

Differences in power consumption for both regions of the program are demonstrated in Figure 6. HIGH gear always consumes more power than LOW gear due to higher frequency and voltage. In the MIX gear mode, the execution region consumes similar power as HIGH gear mode, while load region consumes similar power as LOW gear mode, due to the dynamic change of operating frequency. The opposite happens for the XIM gear. From this graph we can observe that the execution region always consumes more power than the load region, as more computation is required. As we increase the number of cores, there is a resulting increase in power consumption associated with both the HIGH and the LOW gear. If optimal power is required, either 1 or 2 cores running with the LOW gear might be the right choice.

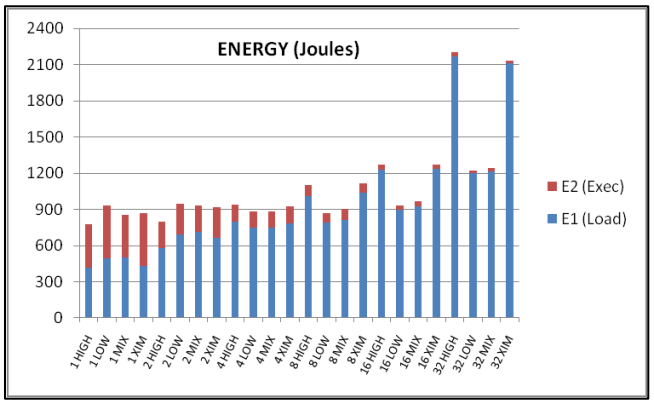


Figure 7. Energy Graph

The energy consumed by each region, as well as the total energy consumption for each simulation, is shown in Figure 7. This graph reaffirms that the energy consumed by a system is dependent on the balance between performance and power, not simply processor speed or power dissipation alone. From this graph we can conclude that for energy considerations, the HIGH gear running with 1 and 2 cores is the most energy-efficient one across all configurations. Because the communication overhead and power consumption are low at low core count, it is worth to run at highest frequency and still be energy-efficient. When running with 4-core or above, LOW gear and MIX gear always provide better energy readings. This is because in our HFO application, when we increase the core count, the program spends a large portion of its execution time in communicating and transferring data among the cores. As we saw from Figures 5 and 6, the communication overhead and hence the time required for completing the load region increases, as well as power consumption for more cores if they want to run at higher frequency.

The metric for measuring power-performance is very common. From the system point of view, minimizing the execution time may usually be the first priority. However, even if the total energy is minimized, the user may not be satisfied with extended system response time [19-20]. In Figure 8 we present the energy-delay product (EDP) for each used configuration. EDP metric takes into consideration both energy and execution time. This graph shows how both mixed gears generally have an EDP between the HIGH and the LOW gear as we anticipated. One thing worth mentioning is that in this case, similarly to the energy analysis, the HIGH gear running with 1 and 2 cores provides the best of both worlds, user experience and energy consumption.

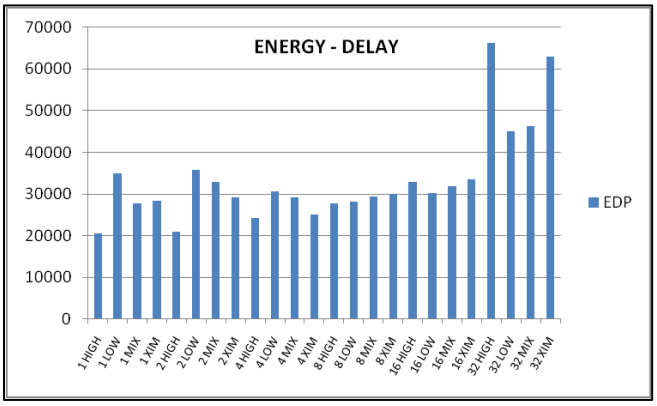


Figure 8. Energy-Delay Product Graph

From observing the presented performance, power, and energy results, we can observe how in most of the cases, mix gears offer varied results between the HIGH gear and the LOW gear when comparing the same number of cores.

VI. Conclusions and Future Work

The performance benefits of utilizing the SCC platform for HFO detection are substantial. The SCC can process data while implementing complex algorithms in short periods of time. In the area of HFO detection alone this is a remarkable advance. This would allow HFO analysis to be performed on extended durations of recordings, hours of data instead of minutes. This increase in data processing capability will act as an analysis tool for neurosurgeons and neuroscientists in order to define the SOZ with higher resolution and confidence.

The advantages associated with EEG processing on the SCC platform can be employed on a myriad of EEG signal processing algorithms. The SCC proves to be an ideal platform on which to process multiple electrode recordings in an energy-efficient manner, while increasing the performance of analysis as a whole. The DVFS capabilities of the SCC allow the user to have full control of energy usage, which can lead to total system energy savings when the code can be broken up into communication-intensive and computation-intensive segments and run with the appropriate voltage-frequency gears. The message-passing architecture, both at the hardware and software level of the SCC, allow for user-defined inter-core and therefore inter-electrode communication, which has been shown to be essential for EEG algorithms.

All blocks pertaining to the HFO-EEG algorithm are implemented and running on the SCC platform. This algorithm is not computationally burdensome to the SCC and is handled in a very efficient manner. As for future work, we expect to implement a program which is able to process 65 electrodes overall and more than 32 at a time. Further research can be done implementing more complex EEG-based detection algorithms. Other areas of interest would be systems and EEG algorithms that incorporate more message-passing or electrode-dependencies than are currently present in the HFO algorithm. The SCC platform allows for the development and implementation of more complex, data-dependent algorithms in which further neuronal phenomena can be examined. A system approach in which multiple algorithms are executed in parallel on the SCC would be beneficial to the field of neuroscience and epilepsy research as well.

Acknowledgment

This work is partly supported by the National Science Foundation and the Department of Defense (DoD). Paul McCall is supported through the National Defense Science & Engineering Graduate Fellowship (NDSEG) Program. The authors are also grateful for the support provided by the National Science Foundation under grants ECCS-1125762, CNS-0959985, CNS-1042341, and HRD-0833093. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or Department of Defense.

References

[1] S. S. Spencer, D. K. Nguyen, and R. B. Duckrow, *Invasive EEG in Presurgical Evaluation of Epilepsy, Chapter 53 of the Treatment of Epilepsy*, 3rd ed. Hoboken, NJ: Wiley, 2009, pp. 767–798.

[2] M. Ayala, M. Cabrerizo, P. Jayakar, and M. Adjouadi, "Subdural EEG Classification into Seizure and Non-seizure Files Using Neural Networks in the Gamma Frequency Band",Journal of Clinical Neurophysiology, Volume 28, Number 1, February 2011.

[3] M. Adjouadi, D. Sanchez, M. Cabrerizo et al., "Interictal Spike Detection Using the Walsh Trans.," IEEE Trans. on Biomed Eng., 51, 868-72, 2004.

[4] Weidong Zhou; Gotman, J.; "Removal of EMG and ECG artifacts from EEG based on wavelet transform and ICA," *Engineering in Medicine and Biology Society, 2004. IEMBS '04. 26th Annual International Conference of the IEEE* , vol.1, no., pp.392-395, 1-5 Sept. 2004.

[5] Smart, O.L.; Worrell, G.A.; Vachtsevanos, G.J.; Litt, B.; , "Automatic detection of high frequency epileptiform oscillations from intracranial EEG recordings of patients with neocortical epilepsy," *Technical, Professional and Student Development Workshop, 2005 IEEE Region 5 and IEEE Denver Section* , vol., no., pp. 53- 58, 7-8 April 2005.

[6] Gardner et al., 2007. Gardner AB, Worrell GA, Marsh E, Dlugos D, Litt B. Human and automated detection of high-frequency oscillations in clinical intracranial EEG recordings. Clin Neurophysiol 118: 1134–1143, 2007.

[7] Morgan, Timothy Prickett (July 23, 2010). "IBM's zEnterprise 196 CPU: Cache is king". The Register. September 7, 2010.

[8] J. Held, J. Bautista, and S. Koehl, "From a Few Cores to Many: A Tera-Scale Computing Research Overview," Research at Intel white paper, 2006.

[9] Moore, Gordon E.; , "Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff.," *Solid-State Circuits Newsletter, IEEE* , vol.20, no.3, pp.33-35, Sept. 2006.

[10] Howard, J.; Dighe, S.; Hoskote, Y., et al , "A 48-Core IA-32 message-passing processor with DVFS in 45nm CMOS," *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International* , vol., no., pp.108-109, 7-11 Feb. 2010.

[11] Valentini, G., Lassonde, W., et al.; An overview of energy efficiency techniques in cluster computing systems, Cluster Computing, pp. 1–13, issn 1386–7857.

[12] Hotta, Y., Sato, M., Kimura, H., Matsuoka, S., Boku, T., Takahashi, D.: Profile-based optimization of power performance by using dynamic voltage scaling on a PC cluster. In: Proc. of the 20th IEEE International Parallel and Distributed Processing Symposium (IPDPS) (2006), 8 pp.

[13] Van der Wijngaart, Rob F. and Mattson, Timothy G. and Haas, Werner, "Light-weight communications on Intel's single-chip cloud computer processor", SIGOPS Oper. Syst. Rev., vol. 45, Issue 1, pp. 73-83, February 2011.

[14] http://www.mch.com/page/EN/605/Medical-Services/Brain-Institute.aspx.

[15] Zelmann, R.; Mari, F.; Jacobs, J.; Zijlmans, M.; Chander, R.; Gotman, J.; , "Automatic detector of High Frequency Oscillations for human recordings with macroelectrodes," *Engineering in Medicine and Biology Society (EMBC), 2010 Annual International Conference of the IEEE* , vol., no., pp.2329-2333, Aug. 31 2010-Sept. 4 2010.

[16] J. Jacobs, P. LeVan, R. Chander, J. Hall, F. Dubeau, and J. Gotman, "Interictal high-frequency oscillations (80-500 Hz) are an indicator of seizure onset areas independent of spikes in the human epileptic brain", *Epilepsia,* vol. 49, pp. 1893-907, Nov 2008.

[17] J. D. Jirsch, E. Urrestarazu, P. LeVan, A. Olivier, F. Dubeau, and J. Gotman, "High-frequency oscillations during human focal seizures", *Brain,* vol. 129, pp. 1593-608, Jun 2006.

[18] E. Urrestarazu, R. Chander, F. Dubeau, and J. Gotman, "Interictal high-frequency oscillations (100-500 Hz) in the intracerebral EEG of epileptic patients", *Brain,* vol. 130, pp. 2354-66, Sep 2007.

[19] Hotta, Y., Sato, M., Kimura, H., Matsuoka, S., Boku, T., Takahashi, D.: Profile-based optimization of power performance by using dynamic voltage scaling on a PC cluster. In: Proc. of the *20th IEEE International Parallel and Distributed Processing Symposium* (IPDPS), 8 pp. 2006.

[20] Brooks, D.M.; Bose, P.; Schuster, S.E.; Jacobson, H.; Kudva, P.N.; Buyuktosunoglu, A.; Wellman, J.; Zyuban, V.; Gupta, M.; Cook, P.W.; , "Power-aware microarchitecture: design and modeling challenges for next-generation microprocessors," *Micro, IEEE* , vol.20, no.6, pp.26-44, Nov/Dec 2000.

A Study of CUDA Acceleration and Impact of Data Transfer Overhead in Heterogeneous Environment

Fahian Ahmed, Saddam Quirem
Byeong Kil Lee
The University of Texas at San Antonio, USA

Bum Joo Shin[1]
Pusan National University[1] Korea

Duk Joo Son[2], Young Choon Woo[2]
Wan Choi[2]
ETRI[2], Korea

Abstract—**Along with the introduction of many-core GPUs, there is widespread interest in using GPUs to accelerate non-graphics applications such as energy, bioinformatics, finance and several research areas. With a wide range of data sizes where the CPU has greater performance, it would be important that CUDA enabled programs properly select when to and not to utilize the GPU for acceleration. Algorithms that use dynamic programming like P7Viterbi algorithm of HMMER 3.0 (genetic application) show high parallelism in its code. Based on performance hotspot analysis, these parallel features were exploited through the use of CUDA and a GPGPU. The CUDA implementation of this algorithm being performed on the Tesla C1060 enabled a 10-15X speedup depending on the number of queries. In this paper, we focus on accelerating HMMER 3.0 - one of the genetic applications with GPUs as co-processors. Also we investigate the potential performance bottleneck in GPU-CPU environment with blowfish - a security application. Based on workload characterization and bottleneck analysis, we provide optimization methodologies to remove the bottleneck.**

Keywords - CUDA, GPGPU, Hotspot analysis, HMMER 3.0, Database, Viterbi Algorithm, blowfish, CUDA profiling.

I. INTRODUCTION

Along with the advance of integration technology, multi-core processors have provided a technical breakthrough to the computing community. This includes both general-purpose processor and application- specific processor domain. In addition to multi -core trend, the demand of high quality of graphics have the programmable GPUs (Graphics Processing Units) evolving into a highly parallel, multithreaded many-core processors. Even with powerful and massively parallel GPUs, it is difficult to achieve peak performance without the knowledge of graphics or graphics dedicated APIs (Application Programming Interface). However, with the introduction of new programming models such as Nvidia's CUDA (Compute Unified Device Architecture) that abstracts the GPU hardware, non-graphics users can easily map wide range of applications into many-core GPUs without having deep knowledge of graphics and GPU architecture [1][2].

The GPGPU (General-purpose computing on graphics processing unit) is a technique of using a GPU, which has high data-parallel processing capability and typically handles computation only for computer graphics, to perform the computation in general-purpose applications traditionally handled by general-purpose CPU. With the introduction of many-core GPUs, there is widespread interest in using GPUs to accelerate non-graphics applications such as bioinformatics, energy, finance and several research areas [1].

CUDA makes use of the massively parallel nature of NVIDIA's graphics processing units to accelerate both graphics computation and general purpose computations that can be performed in parallel. The Tesla series of GPUs represents the NVIDIA's line of HPC (High Performance Computing) oriented graphics processors, or GPGPUs. Each graphics processor on a Tesla card contains several graphics processing clusters (GPCs), which in turn contain multiple streaming multiprocessors (SMs), which contain dozens of CUDA cores. Each SM can simultaneously execute thousands of threads. This highly parallel environment can effectively reduce the cost of achieving higher levels of computational speed.

Here, we architecturally characterize some basic kernels and genetic applications and investigate performance hotspot function in HMMER 3.0 and blowfish. HMMER [3] is an application whose main use is ―searching sequence databases for homologs of protein sequences. It is among many bioinformatics applications whose algorithms can be easily accelerated on a GPU.

Even though the GPUs provide highly parallel processing capability, the interface between CPU and GPU could be a performance bottleneck due to heavy data transfer. In this case, if data transfer time is overwhelming the computation time on GPU, it would be better keep the computation on CPU instead of using GPUs. Thus, we aim to observe the borderline between CPU vs. GPU performance as well as the effects of using different types of memory. For this observation we use a security application name blowfish with different input size. The Security application includes several common algorithms for data encryption, decryption and hashing. Among others blowfish [4] is one of the popular security applications.

The rest of paper is organized as follows: section II describes related works. The workload characterization including hotspot analysis is explained in section III. Section IV shows CUDA implementation, and section V describes results and analysis. Finally, concluding remarks and future works are presented in the last section.

II. RELATED WORK

Major genetic applications known to make use of the GPU include Gromacs, NAMD, HMMER and most notably Folding@home. NVIDIA GPUs account for over 35% of Folding@home's native TFLOPS. HMMER itself is a database search application, and like many similar applications, GPUs have been applied for acceleration. Peter

Bakkum and Kevin Skandron of the University of Virginia have previously ported SQLite to CUDA resulting in at least 20x speedups in query time. For HMMER application, various types of coprocessors were utilized for acceleration. Perhaps most interesting acceleration is done with the FPGA by Steve Derrie and Patrice Quinton [5]. This is where the P7Viterbi algorithm was implemented in hardware as a set of MUXs and LUTs. This FPGA implementation achieved a 50x speedup in one case. A CUDA implementation of HMMER is also present. Walters *et al* accelerated HMMER by focusing on the P7Viterbi algorithm at the core of the application. Using a single Tesla GPGPU, GPU-HMMER was capable of a 30x speedup with a large HMM size (number of states). Likewise, an earlier implementation of HMMER utilizing streaming processors (which includes NVIDIA GPUs), known as ClawHMMER, took a similar approach by targeting the P7Viterbi algorithm [6]. While all the previous implementations are with old version of HMMER, we focus on newer version of HMMER with different memory allocation schemes.

A number of researchers have discussed bandwidth troubles that can arise with frequent or poorly managed data movement between devices. Schaa and Kaeli [7] examine multiple GPU systems and acknowledge that unless a full working set of data can fit into the memory on a GPU; the PCI Express will be a bottleneck. Owens et al. [8] express similar concerns. For this observation we use a security application name blowfish with different input size.

III. Workload Characterization

3.1 Scalability and bottleneck Analysis

We choose two kernels from CUDA SDK and three bioinformatics applications from SPEC CPU 2006, and characterize them on CPU only and CPU-GPU computational environment. The description of each benchmark is shown in Table 1.

Table 1: Benchmarks and description

Type	Benchmark	Description
Kernels	Matrix multiply	Two dimensional matrix multiplication with multiply-accumulation functions
	Histogram	an image processing algorithm that combines a stream of pixel light values into a series of bins that represent the distribution of light across an image
Applications	Namd	[SPEC 2006 FP] Simulates large biomolecular systems. The test case has 92,224 atoms of apolipoprotein A-I.
	Blowfish	[Mibench] Blowfish is an encryption algorithm. It is a symmetric block cipher that uses a variable-length key from 32 bits to 448 bits.
	Hmmer	[SPEC 2006 INT] Protein sequence analysis using profile hidden Markov models

In general, matrix multiply and histogram kernels consist of large amount of ALU computations, and computation of those applications can be accelerated with more parallel functional units. We tried to explore the *namd*, one of bioinformatics applications, that it shows proportional performance impacts on the variation of hardware configurations. Figure 1 shows IPC (Instruction per Cycle) a performance metric, for multiple configurations using simplescalar. As variable resources, we use instruction fetch queue, load store queue, and decode width, issue width, number of ALU, number of multiplier and memory width. We see the direct performance improvement in terms of the number of functional units. The width of fetch, decode and issue also affect the IPC because those components are aligned on the processor pipeline. Compared to the result (leftmost graph) with the smallest resource, the rightmost graph with the largest resource shows 63% improvement in IPC. Based on this simulation, we observe that the performance of *namd* application can be scalable with the number of functional units and the number of other resources.

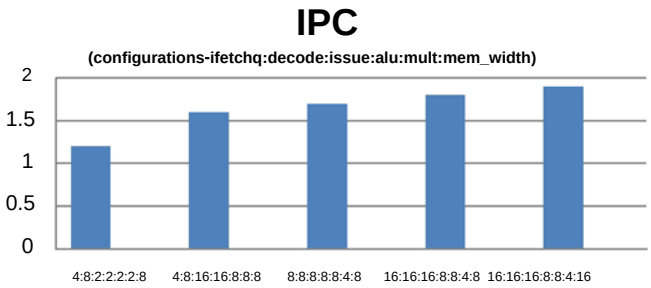


Figure 1: Impact on IPC (Instruction Per Cycle) with hardware variations (*namd*)

Data transfer time investigation: We investigate the memory transfer overhead costs for CPU/GPU performance comparisons. Our results show that data transfer time can be as significant as main kernel runtime if data size is big enough. Also, if the number of CUDA cores is increased and GPU dedicated memory size is increased; the data transfer overhead will be more critical to overall performance. Two CUDA SDK kernels are used for this matter. Nvidia CUDA profiler (cudaprof) is used for characterizing and breaking down the GPU time. Normally, GPU time consists of major kernel computation time and data transfer time. In CUDA profiler, data transfer time is divided into two components: *memcpyDtoH* and *memcpyHtoD*. The memcpyDtoH means Device to Host (data transfer from the GPU) and memcpyHtoD means Host to Device (data transfer to the GPU).

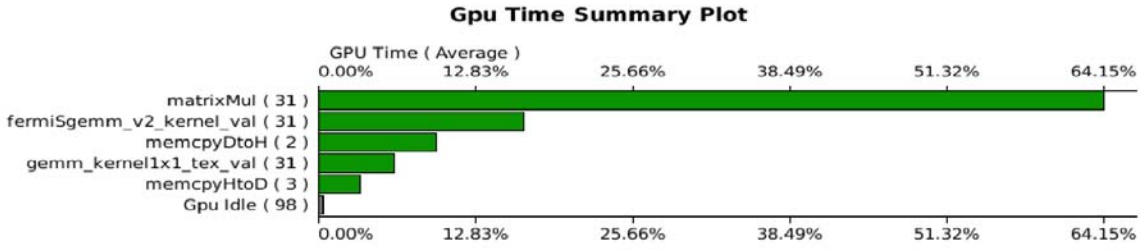
Figure 2 shows the experiment on Nvidia GTX 460. It shows that kernel computation time is major time-consuming part, and data transfer time is around 10% of the total GPU time for both Matrix multiply and Histogram. It is interesting to note that Matrix multiply shows more data transfer time from the GPU and Histogram shows more data transfer time from the CPU. The reason is coming from each kernel's algorithmic characteristics. The results from the Matrix multiply on GPU will be slightly bigger data size than the original data from the CPU, while the results from the Histogram on GPU will be smaller data size than the original data from the CPU.

3.2 Hotspot Analysis

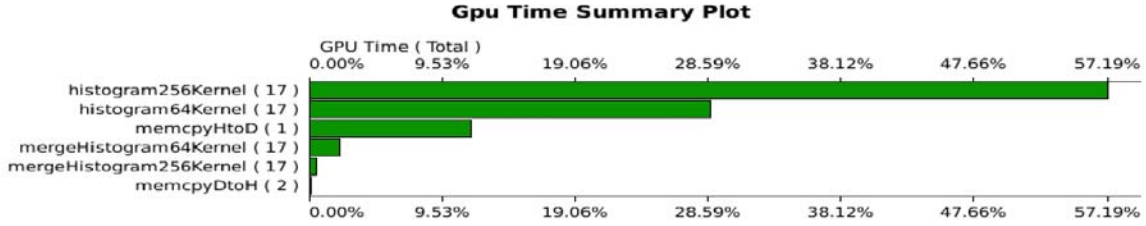
The Intel VTune Performance Analyzer provides information on the performance of code. The VTune analyzer shows the performance issues, enabling to focus tuning effort and get the best performance boost in the least

amount of time. The Hotspots analysis helps understand the application flow and identify sections of code that took a long time to execute (hotspots). A large number of samples collected at a specific process, thread, or module can imply high processor utilization and potential performance bottlenecks. With the help of some performance analyzer software like Vtune, we can figure out the hotspot. According to the specific hotspot modules, we can modify some modules for performance acceleration. This acceleration and flexibility of an application on GPU can be achieved by applying high level GPU programming languages such as CUDA.

There are several sub-programs in HMMER application like *phmmer, jackhammer, hmmbuild, hmmsearch, hmmscan, smmaligh*, etc. Each sub-program shows similar hotspot results, but here we only discuss about jackhammer. The *Jackhmmer* program is for searching a single sequence query iteratively against a sequence database. VTune Analyzer creates and run an activity that collects performance data of the application. An activity means lunching application onto Vtune profiler. Figure 3 shows the sampling summary view of the *Jackhmmer*.



(a) Matrix Multiply



(b) Histogram

Figure 2: GPU Time comparison of main computation and data transfer between CPU and GPU. Experiment is on Nvidia GTX 460 (336 cuda cores, 1GB GPU memory, 256-bit memory I/O)

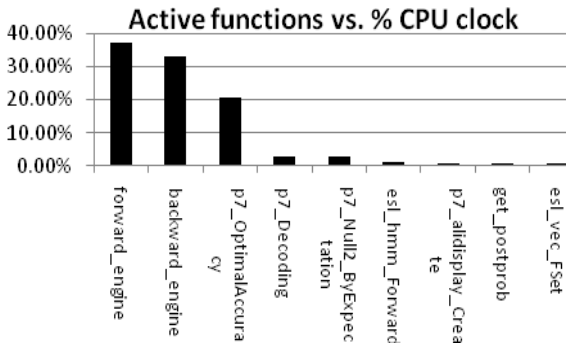


Figure 3: Hotspot analysis: the percentage of CPU clock for active functions (Jackhmmer)

IV. RESULTS AND ANALYSIS

4.1 CUDA Parallelization of p7Viterbi

With each implementation of the HMMER's P7Viterbi algorithm timers were utilized in order to properly compare results. For the CPU version, the elapsed time was simply measured between the start of the function and after the completion of the function. The GPU implementations utilize two timers that measure the total time taken to allocate memory storage and transfer memory onto the GPU and back, and the kernel execution time. The second timer simply measures the time taken to execute the CUDA kernels.

The P7Viterbi algorithm iterates through every observation, from 1 to L, and determines a score for each state from 1 to M. It is not possible to calculate the scores for each observation in parallel, because the scores of the next row of the dynamic programming matrix depend on the previous row. However, it is possible to calculate each column (query), independent of the other.

Figure 4 shows the path of the Viterbi algorithm through the dynamic programming matrix and what cells were computed in parallel. A set of kernels were written to properly convert the all the functionality of the P7Viterbi function. Because the data was already moved to the GPU, the initialization of the data was executed as a single-threaded kernel.

4.2 Accelerating Application with GPU

Based on our experiments, the CUDA implementation of the P7Viterbi algorithm showed an over 14x speedup over the original implementation of the P7Viterbi function, as shown in Figure 5. The speedup increased exponentially as the number of threads launched (number of queries) doubled. No more than 16,384 queries were tested due to the limitations of the GPU memory. The total speedup (CPU

Time/GPU Time), the increase in speed was significantly reduced.

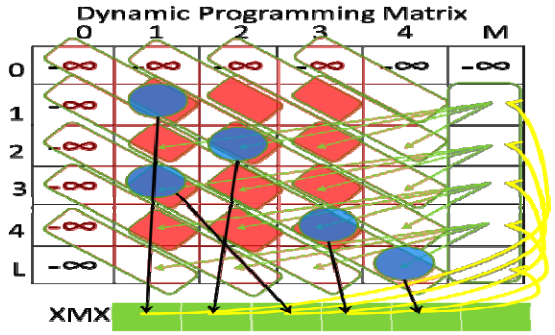


Figure 4: Viterbi Algorithm Executed in Parallel

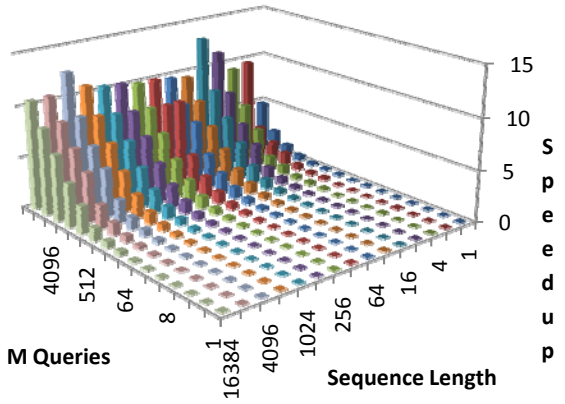


Figure 5: Total Time Speedup represented as CPU Time over GPU Time

Table 2: GPU time for three different input size

Method	GPU time(us) Large data set (3.1 MB)	GPU time (us) Large data set (12 MB)	GPU time(us) Large data set (120 MB)
BF_encrypt	104441	107406.1	200857
BF_cfb64_encrypt	52406.1	60189	98277.84
main	3854.3	4030.66	5814.89
memcpyHtoD	21102.7	37041.7	87981.46
memcpyDtoH	504.608	681.37	1945.3

4.3 Effect on Memory Transfer Overhead

Blowfish has two types of data set - large and small. For the experiment issue we modified large data set into three set so that we can get the clear picture of data transfer overhead. And then using Vtune we got the hotspot zone and improved the zone by converting source code to CUDA. Hotspot module name is BF_encrypt(). It uses a special function name BF_ENC() which is the main spot for higher number of clock ticks. Since Vtune only shows the CPU clock time so the improvement on GPU never reflects in Vtune. So we used CUDA profiler to get the GPU- CPU time and also the data transfer overhead. For three different large data sets summary table is given in Table 2.

From the Figure 6 graph (GPU time), we realize that as the data set increases both kernel execution and data transfer overhead increases. But data transfer from Host (CPU) to device (GPU) increases significantly. When data set increases from 12MB to 120MB, kernel execution time (GPU time) increases 2 times but data transfer time increases almost 4 times.

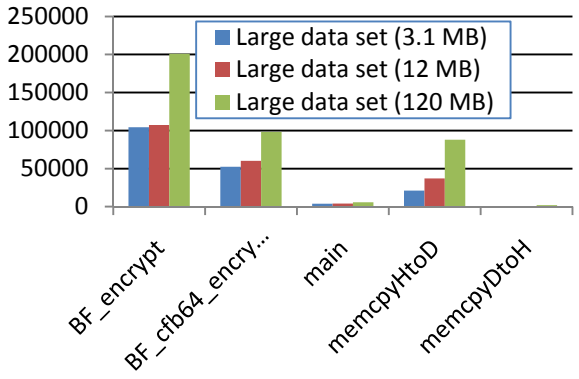


Figure 6: GPU time for three different data sets

V. Conclusion

A single application includes many kernels or functions, and some of them are heavily used for entire computation. Based on kernel's algorithmic characteristics, some functions show higher data transfer time than kernel computations on GPU, but others are not. Therefore, full CUDA modeling could not be an ideal solution. However, the amount of overhead can vary drastically depending on how a GPU kernel will be used in an application, or by a scheduler. Future work will include using this information to inform scheduling decisions about whether to run kernels on a GPU or on the CPU.

ACKNOWLEDGEMENT: This work was supported by the IT R&D program of MKE/KEIT. [10038768, The Development of Supercomputing System for the Genome Analysis]

References

[1] NVIDIA Corporation. Press Release: NVIDIA Tesla GPU Computing Processor Ushers In the Era of Personal Supercomputing, 20 June 2007.

[2] S. Ryoo, C. I. Rodrigues, S. S. Baghsorkhi, S. S. Stone, D. B. Kirk, and W. W. Hwu, "Optimization principles and application performance evaluation of a multithreaded GPU using CUDA," In Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 2008.

[3] Rob Finn and Jody Clements, "HMMER3: a new generation of sequence homology search software", http://hmmer.janelia.org, 28 March 2010.

[4] Matthew R. Guthaus, Jeffrey S. Ringenberg, Dan Ernst, Todd M. Austin, Trevor Mudge, Richard B. Brown "MiBench: A free, commercially representative embedded benchmark suite," IEEE 4th Annual Workshop on Workload Characterization, Austin, TX, December 2001.

[5] Steven Derrie and Patrice Quinton, "Parallelizing HMMER for Hardware Acceleration on FPGAs", IEEE 18th International Conference Application-specific Systems, Architectures and Processors, 2007.

[6] Daniel Horn, Mike Houston, and Pt Hanrahan, "ClawHMMER: A Streaming HMMer-Search Implementation", presented at Supercomputing 2005, Washington, D.C., 2005.

[7] D. Schaa and D. Kaeli, "Exploring the multiple-GPU design space," in International Parallel and Distributed Processing Symposium., May 2009, pp. 1–12.

[8] J. Owens, M. Houston, D. Luebke, S. Green, J. Stone, and J. Phillips, "GPU computing," Proceedings of the IEEE, vol. 96, no. 5, pp. 879–899,May 2008.

Session II: VLSI Design

The Impact of Technology Scaling in the SpiNNaker Chip Multiprocessor

Eustace Painkras and Steve Furber
University of Manchester
United Kingdom
Email: painkras@cs.man.ac.uk

***Abstract*—This paper focuses on technology scaling of the SpiNNaker Chip Multiprocessor (CMP). A detailed characterization of the SpiNNaker Processor Node has been carried out in 130nm and 90nm semiconductor processes with many variants of the process libraries exploring the area, power and performance aspects. Experimental results of processor node performance, power dissipation and silicon area are presented and have been used to predict the best technology for a future SpiNNaker CMP for high-performance computing.**

I. Introduction

CMOS technology has seen continuous and systematic rise in transistor density and processing performance for the last few decades in accordance with the self-fulfilling prophesy of Gordon Moore [1] and the scaling theory proposed by Dennard et al. [2]. Other consequences of technology scaling are increased power per unit area, due to transistor size scaling faster than the power per transistor, and lower die cost [3]. The aggressive scaling-down of transistor dimensions in technology generations has enabled designers to integrate a great many processor cores in a single die, thus building Chip Multiprocessors. The focus of this paper is the technology scaling aspects of the SpiNNaker CMP so that the SpiNNaker system can be scaled to a large-scale high-performance computing platform taking advantage of the process technology enhancements. For this, the impact of the underlying parameters and the trade-offs involving area, power and performance on SpiNNaker CMP are studied.

An exploration of the impact of technology scaling at 130nm and 90nm technology nodes for a chip multiprocessor is carried out to understand how embedded microprocessor CMPs are affected by technology scaling. In particular, this is a case study of the system-level performance/power analysis in a CMP with many embedded ARM968 cores [4]. The evaluation results pertaining to the technological options and design cost form the design basis for a future SpiNNaker CMP scaled to the appropriate technology node.

This paper is structured as follows. Related work in technology scaling, cortical simulators and CMPs is presented in Section II. Section III reviews the design of SpiNNaker and its components. We also describe the multi-core SpiNNaker chip, the on-chip and off-chip communication mechanisms and the connection of multiple chips to form a large-scale SpiNNaker system. Section IV is devoted to describe in detail the technology scaling experiments we have carried out on the SpiNNaker CMP with the associated results and analysis. After that, we focus on the major design considerations for the current and future SpiNNaker CMPs in Section V. The paper is concluded with pointers to what the future may hold for the SpiNNaker chip in Section VI.

II. Related Work

Huang et al. [5] apply analytical models, derived from the technology-scaling predictions of ITRS [3], to commodity processors of relatively large die sizes. Chung et al. [6] also use the ITRS road map to construct their scaling model. Our work, in comparison, uses experimental results to evaluate many-core design considerations and zero-in on the best choice of process technology for a future SpiNNaker CMP.

Most large-scale neural simulations [7], [8] utilize supercomputers. The Blue Brain project [7] uses BlueGene/P supercomputer [9] to simulate cortical columns. The BlueGene/P is not a custom architecture, but a general purpose massively parallel system. Ananthanarayanan et al. [8] also report using the Blue Gene/P machine for cat cortical column simulation as part of DARPA's Systems of Neuromorphic Adaptive Plastic Scalable Electronics (SyNAPSE) program.

In the CMP arena, quite a few many-core chips have been reported in literature which differ in a host of parameters like die size, process technology, number and type of processors, power consumption, operating frequency, flops etc. making comparison difficult. However, a few notable CMPs are mentioned below.

The Blue Gene/Q chip, the basic processing element for IBMs latest offering - the Blue Gene/Q massively-parallel scientific computer - employs 18 PowerA2 processor cores with floating-point units occupying a silicon real-estate of 359.5mm^2 with 1.47 billion transistors fabricated in a 45nm SOI CMOS process. Peak performance for the chip was specified at 204.8 GFLOPS with 55W power dissipation when operated at 1.6GHz with a 0.8V supply [10].

Truong et al. [11] have presented a 167-processor computational platform suited for DSP and multimedia applications built from simple programmable processors. Implemented in 65nm low-leakage ST Microelectronics CMOS process, the chip has 55 million transistors occupying a die area of 39.44mm^2. The power consumption is 47.5mW when operating at a clock frequency of 1.07GHz.

Intel's 80-core TeraFLOPS processor [12] consists of 80 tiles arranged as an 8x10 array and is designed to operate at 4GHz. Fabricated in a 65-nm 8-metal CMOS process, it occupies an area of 275mm^2, has 100 million transistors and achieves over 1.0TFLOPS while dissipating 97W at 4.27GHz.

Tilera's TILE64 processor [13], with 64 tile processors arranged in an 8x8 array, fabricated in a 9-layer 90nm triple-V_t CMOS process consumed 10.8W core power when running a deep-packet inspection application at 750MHz. Their latest TILE-Gx family [14] of 40nm many-core processors, designed for cloud computing datacentres, come with up to a hundred 64-bit cores operating at frequencies up to 1.5GHz. The 100-core version with 32MB of on-chip memory has a power consumption of 20W.

Compared to the above chips, the SpiNNaker CMP, with 100 million transistors in a 101.64mm^2 die, peak performance of 3.96GIPS and a power consumption of 1W at 1.2V when all the processor cores are functioning at 180MHz, is a customized architecture which is much more energy-efficient.

III. Overview of SpiNNaker

SpiNNaker [15] is a biologically-inspired, massively parallel computing architecture designed to facilitate the modelling and simulation of large-scale spiking neural network systems of up to a billion neurons and a trillion synapses in biological real-time. It is a generic and programmable platform for neuroscientists, psychologists and brain researchers to explore brain functions with software neuronal models. The architecture scales from a single chip with 18 processor cores in its smallest configuration to a system of 65,536 chips with 1,179,648 processors in a fully-fledged machine, delivering peak processing power of over 235 Dhrystone TeraIPS.

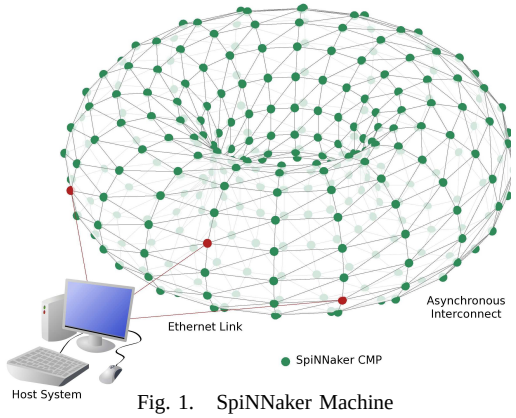


Fig. 1. SpiNNaker Machine

A. SpiNNaker System Architecture

The SpiNNaker machine is designed as a very large array of up to 2^{16} nodes, each node containing a CMP die and a 128MB SDRAM die, stacked and housed in a single 300-pin BGA package. Each CMP contains 18 processing cores, each capable of simulating up to 1000 spiking neurons. Instead of many large and fast processors, this design takes advantage of one of the most important features of embedded processor cores - their low power consumption - to deploy a low-power, massively-parallel architecture with many simple cores. Special emphasis is also placed on its fault-tolerance features. Fig.1 shows the connection of multiple chips to form a SpiNNaker machine and also the manner in which this system connects to the outside world.

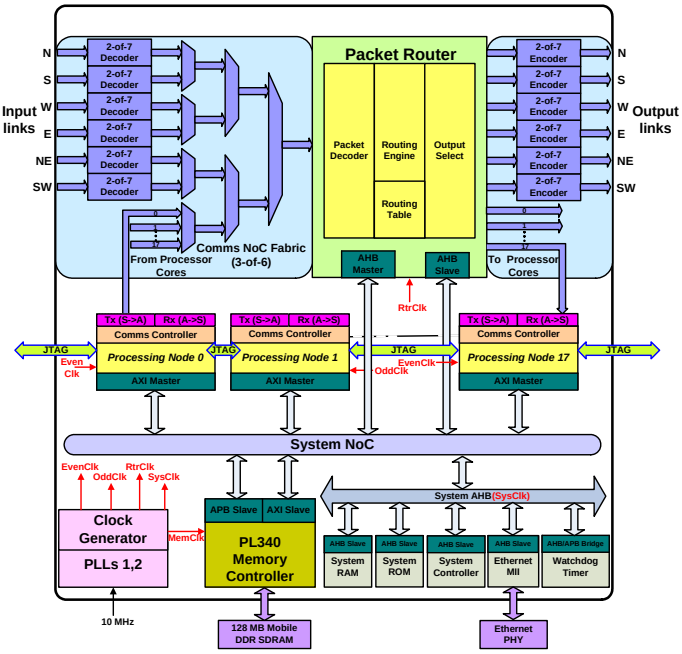


Fig. 2. SpiNNaker chip organization showing the CMP and the SDRAM

B. SpiNNaker CMP

The basic building block of the SpiNNaker machine is the SpiNNaker chip with the CMP and the SDRAM. The SpiNNaker CMP, shown in Fig.2, is a Globally Asynchronous Locally Synchronous (GALS) multi-processor SoC [16] with ARM968ES processing nodes residing in synchronous islands surrounded by a packet-switched asynchronous communications infrastructure. The GALS architecture not only simplifies timing closure in the SoC design [17] but also facilitates isolation of faulty processor nodes. Self-timed delay-insensitive on-chip interconnects based on CHAIN technology [18] are the backbone for on-chip and off-chip communications. System-wide communication is handled by the two separate hardware communications channels - the *Comms NoC* and the *System NoC*. The Comms NoC implements inter-processor communications between any processor to any other processor in the system. The off-chip communication is handled through a bespoke multicast router with its routing tables and six full-duplex links with *Transmit* and *Receive* interfaces connecting to neighbouring chips in the north, south, east, west, north-east and south-west directions forming a 2D toroidal triangular mesh. The System NoC provisions the chip-wide sharing of system resources, viz. 32KB System RAM, 32KB System ROM, System Controller, Watchdog Timer and Ethernet Controller Interface. It also provides access through a memory controller to 128MB off-die SDRAM, private to each CMP but global to its processors, housed in the same package. The

shared System ROM and RAM are used for loading and wave-pipelining the boot/application code and for inter-processor communication as a mailbox, when needed. Each CMP can communicate with the external world through a 100-Mbit Ethernet interface.

The System NoC implements Silistix's custom protocol with AMBA AXI adapters [19] whereas the Comms NoC has 2-of-7 NRZ for off-chip links and 3-of-6 RTZ for on-chip links. As opposed to typical synchronous bus interconnect, the asynchronous NoCs with their packet-switched fabric deliver scalable, high-bandwidth, power-efficient, multicast and low-latency communications [20].

C. SpiNNaker Processor Node

The internals of the SpiNNaker Processor Node are shown in Fig.3. The processor selected for the SpiNNaker machine is the 32-bit ARM968E-S processor which is a power-efficient, small-footprint core designed for low-power, data-intensive applications with a Dhrystone performance of 1.1 DMIPS/MHz [4]. The core is fully synthesizable, so it can be ported quickly and efficiently to different process technologies. The processor node performs both computation and communication functions. On-chip, each processor node has an ARM968 core, private, directly-connected 32KB Instruction Tightly Coupled Memory (ITCM) and dual-banked (for interleaved word access) 64KB Data TCM (DTCM) and peripherals such as the Timer, Vectored Interrupt Controller (VIC), Communications Controller (CC) and Direct Memory Access Controller (DMAC). The Timer generates simulation time step intervals for the neuronal models. The CC handles the packet-based traffic at the processing end. The custom-designed DMAC shares access to the TCMs with the processor core through a dedicated AMBA AHB-lite slave interface. Its main function is to offload communication tasks from the processor and provide transparent access to other system resources such as the System RAM, System ROM and SDRAM through the System NoC. The VIC handles up to 32 interrupt requests with programmable priority from the node peripherals as well as system resources and generates IRQ and FIQ interrupts to the processing core. A IEEE 1149.1-compliant JTAG port is also available for debugging purposes.

IV. Technology Scaling

In every process generation, the minimum channel length of the transistor is roughly scaled by a factor S = $1/\sqrt{2}$. In practice, a combination of both Constant Field Scaling and Constant Voltage Scaling is used to scale CMOS devices. Compared to the previous process generation, same-size die can now accommodate twice the number of transistors. Thus, the major benefit of scaling is the increased density of transistors thereby enhancing the capability to integrate more devices in the same area or added functionality. The side-effects are the ever-increasing power-density and the associated difficulties to dissipate the increased power. Therefore, to harness the benefits of Moore's law and deliver increased compute-power and performance while simultaneously keeping power

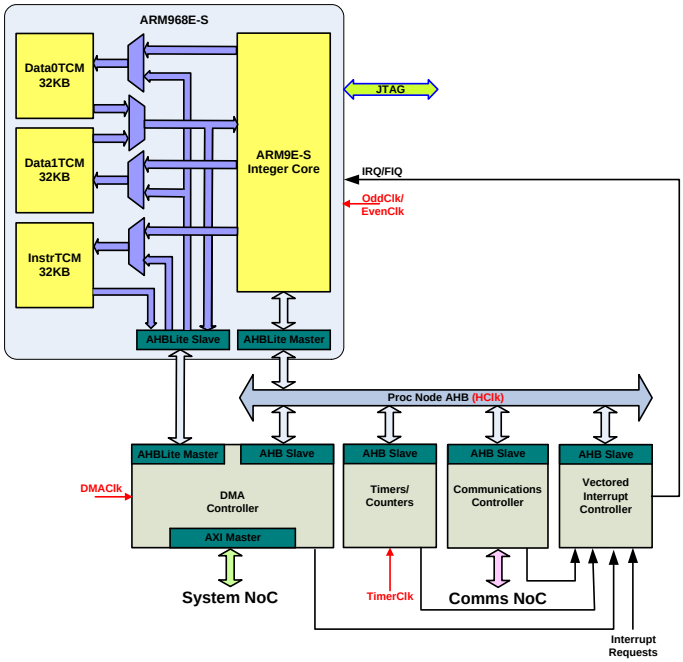


Fig. 3. Details of a SpiNNaker Processor Node

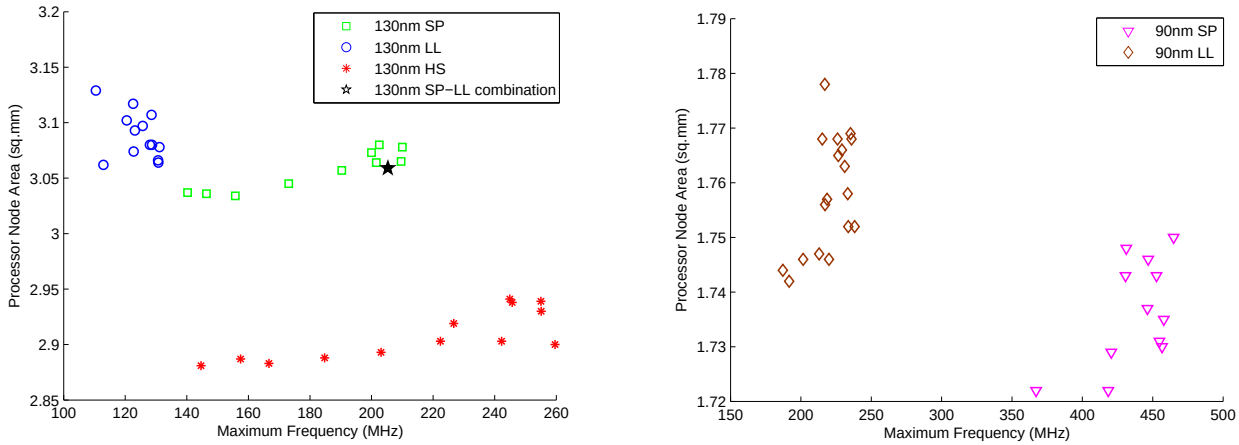
to a minimum is onerous. Also, the leakage power increases dramatically below the 130nm technology generation.

A. Process Libraries

Wafer fabs and independent foundries offer several process technology library variants at different technology nodes, each optimized for performance, power, supply voltage or threshold voltage. This selection of libraries enables the CMP designer to carry out design exploration from the power/speed tradeoff viewpoint and to incorporate power-management considerations throughout the design flow. For example, the choice could be between a high-speed high-leakage library or a low-speed low-leakage library. To reduce the leakage power in finer geometries, the threshold voltage (V_t) of the transistors is raised, but this results in a corresponding decrease in the transistor operating frequency. In the SpiNNaker CMP design, multiple-threshold libraries have been used in a concerted effort to meet the performance of timing-critical circuits and minimize the leakage power in non-timing-critical paths.

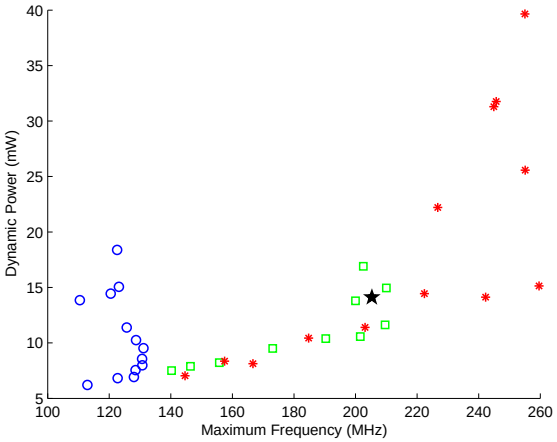
B. Framework, Tools & Metrics

We use the SpiNNaker CMP as the experimental platform for our research. The Synopsys Galaxy Design Platform, with tools for RTL simulation, logic synthesis, physical implementation, timing and power analysis, such as Design Compiler, IC Compiler, PrimeTime and PrimeRail, is used for the design and implementation tasks. Faraday standard cell libraries for UMC process technology at 130nm and 90nm nodes - accessible to universities for academic and research purposes - have been used for the experiments. System-level metrics envisaged are the area/design cost, throughput and power efficiency. The experimental results are analyzed and discussed below with a view of comparison in terms of area, power and performance.

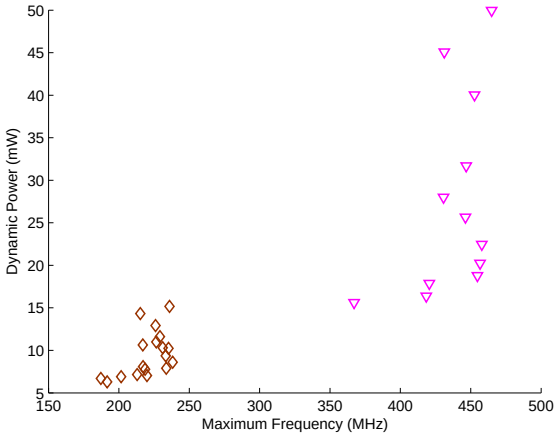


(a) Area in 130nm technology

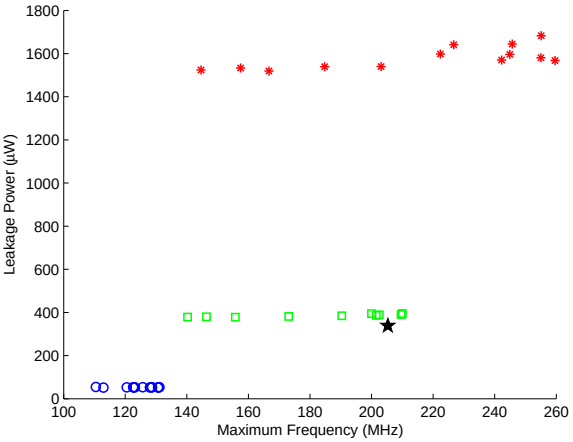
(d) Area in 90nm technology



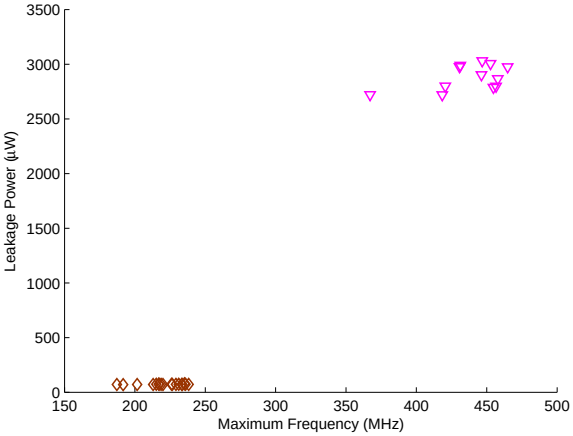
(b) Dynamic Power in 130nm technology



(e) Dynamic Power in 90nm technology



(c) Leakage Power in 130nm technology



(f) Leakage Power in 90nm technology

Fig. 4. Area, Dynamic Power and Leakage Power of SpiNNaker Processor Node for 130nm (SP, LL and HS libraries) and 90nm (SP and LL libraries) process technologies with the current implementation also highlighted with the pentagram

C. Experiments on Technology Scaling of the SpiNNaker Processor Node

Experiments on the physical implementation of the SpiNNaker Processor Node are carried out in 130nm and 90nm technology nodes with Faraday standard cell libraries. The silicon area, highest frequency attained and power-consumption figures are measured from these experiments. These figures may be extrapolated for future technology nodes or form the basis for system model creation for a typical CMP.

To get a comprehensive perspective of technology scaling

from 130nm to 90nm, we have evaluated three different standard cell library variants (High Speed (HS), Standard Performance (SP), Low Leakage (LL)) at the 130nm technology node and two variants (SP and LL) for the 90nm node. Not just the standard cells, but the RAMs also specifically designed for these library variants have been utilized in our experiments. The HS library cells are targeted for implementations where achieving highest possible performance is paramount. The SP library is characterized for general-purpose standard-performance implementations. The LL library has cells with high threshold voltage and therefore has low leakage-power dissipation.

It is worthwhile noting here that the experimental results presented below, for the sake of a deeper understanding of the key building block of the CMP and for carrying out design exploration on a small-scale problem, pertain to the Processor Node alone and not the entire CMP. The SpiNNaker Processor Node is synthesized with its operating frequency as the main design constraint for target frequencies ranging from 100 to 500MHz in numerous iterations and measurements taken of the worst critical path delay, maximum frequency achieved, dynamic power, leakage power and area. Only the synchronous portion of the processor node was subjected to the following experimentation as the asynchronous network interface part was hand-crafted to safeguard the timing assumptions. This is the reason that the processor node area in 130nm technology mentioned in the following section is about 3.0mm^2 whereas the fabricated processor node area is 3.75mm^2.

D. Experimental Results for 130nm Process Technology

Operating Frequency: As can be seen from Figs.4(a)-(c), depending on the library variant chosen, the frequency at which the processor node can be operated varies from a maximum of 260MHz with the HS library to a minimum of 110MHz with the LL library in 130nm technology. This corresponds to a critical path length of 3.85ns/9.05ns at the worst-case process corner of the respective libraries. The design targeted towards the SP library can achieve almost the same performance as the HS library in the lower range of the desired frequencies whereas the higher-end of the range is achievable only with the HS library with considerable overhead on the dynamic and leakage power as explained below. So, depending on the operating frequency desired, one of these libraries or combinations of up to two libraries (for e.g. SP-LL, HS-SP) may be chosen by the designer as the target library for implementation.

Silicon Area: Fig.4(a) shows the silicon area for the processor node in a 130nm process. The area ranges from 2.88mm^2 with the HS library to 3.13mm^2 with the LL library. This anomaly in standard cell area, wherein the HS design is much more compact than the LL design, can be directly attributed to the number of standard cells utilized in the design: 46,709 for HS as opposed to 66,886 for the LL design. Comparison between the same type of standard cell with the same drive strength, for e.g. NANDX8, from LL and HS libraries shows that the intrinsic delay for the LL-type standard cell is twice that for the HS-type cell. Therefore, in an attempt to meet the target frequency (design constraint), the tools introduce more cells in the slower paths to extract greater parallelism. The end-result is larger area for a much slower design. The SP design, as expected, is a medium-sized design with performance comparable to the HS design at the lower end of the operating frequencies achieved. It makes more sense to make use of the SP library for designs where the achievable operating frequency is not the main criterion, as the area overhead is very little (about 0.18mm^2).

Dynamic Power Consumption: Fig.4(b) shows the dynamic power consumption predicted by the tools for the processor node in a 130nm process. The range over which the dynamic-power consumed by the design varies within a single target-library is strikingly wide. For the HS design, the processor node consumes about 7mW @ 145MHz *vs* 40mW @ 255MHz. This huge variation can be explained by the increase in the standard cell count of the design at 255MHz (60,735) compared to the lower frequency (46,709). As stated in the above paragraph, the implementation tools try to match the design constraint set by the designer as closely as possible, which in turn leads to a bloated design at the extreme case scenario. Therefore, it is not advisable to use the HS library if the power budget is limited. The SP and LL designs, naturally, consume less power as they operate at lower frequencies in comparison to the HS design. However, there is an overlapping central region in the graph where the dynamic power is comparable for HS and SP designs. In this specific region, other metrics such as the area and leakage power come into play while evaluating the trade-offs involved. If total power was the overriding constraint, we would choose the LL library for the design implementation and our experimental results lend support to this.

Leakage Power Consumption: Fig.4(c) plots the static leakage power for 130nm designs at the various operating frequencies. As can clearly be seen, the LL library consumes the least (around 50μW) whereas the HS design has the highest leakage power (around 1600μW), with the SP design falling in the middle (around 385μW). The SP design has 75% less leakage power in comparison with the HS design over the central region of the graph with approximately the same dynamic power consumption. It is important to note that as the cells in the LL library are designed with the intention of keeping leakage power to the minimum, this observation validates our design. Also, leakage power accounts for about 10% of the total power consumed under full-load conditions. This constitutes a major drain on the power resources for systems in which portions of the circuitry have to be put in sleep-mode for a considerable amount of time.

E. Experimental Results for 90nm Process Technology

We had to make do with the two variants of the standard cell libraries and SRAMs that were available for the 90nm process: SP and LL. For the sake of completeness, the experiments should have included 90nm HS library. This was unavailable and therefore, could not be included in the evaluation.

Operating Frequency: A noticeable difference in operating frequency is inferred from Figs.4(d)-(f) - the SP design at about 465MHz is twice as fast as the LL design at 238MHz. Compared to the 130nm design, it is to be noted that the operating frequency attained by the 90nm LL design is almost the same as that achieved by 130nm SP and HS designs. Overall, the 90nm LL library is a much better option for a future design, if the aim is to operate at around 230MHz, as it comes with added benefits of smaller area, lower dynamic and leakage power, compared to the 130nm HS library. Albeit, if the specification is to operate at around 400MHz, the 90nm SP library is the way forward.

Silicon Area: The area for 90nm SP and LL designs are plotted in Fig.4(d). For the processor node, the area ranges from 1.72mm^2 to 1.75mm^2 with the SP library and from 1.74mm^2 to 1.78mm^2 with the LL library. We can infer from this metric that there is not much difference in the area irrespective of the library variants - the maximum variation is only about 3% - and therefore insignificant at 90nm. However, progressing from 130nm to the finer 90nm technology allows the integration of roughly double the function on a similarly sized chip. This can be confirmed by the fact that the silicon area has shrunk by 43% from approximately 3mm^2 in 130nm to about 1.7mm^2 in 90nm. It is to be noted that the RAM area, which forms 80% of the Processor node area, has itself reduced from 0.8mm^2 to 0.47mm^2, a 60% shrink. So, it can be inferred that the scaling to the next technology generation, at least in this particular technology, affects both the RAM macros and the standard cells uniformly.

Dynamic Power Consumption: Fig.4(e) is an interesting plot in that it shows that the dynamic power consumed by the 90nm SP design varies from 15.62mW @ 367MHz to 49.97mW @ 465MHz. This is too drastic a variation, a three-fold increase, which proves that this library is only to be used if we are pushing for the highest operating frequency and are not bothered by the power consumed, which might be wasteful when the operating costs are factored in. However, if the specification calls for approximately 360MHz, the 90nm SP design is indeed useful as the dynamic power is comparable to most of the other library designs, disregarding the leakage power at this point. 90nm LL design is favorable in the 200-250MHz region due to the reasonable value for the dynamic power consumed. The dynamic power variation at 90nm is quite similar to that at 130nm. The dynamic power consumption of 90nm SP design is slightly higher than 130nm SP design with improvement in operating frequency from 250MHz to 450MHz. By capitalizing on the higher operating frequency of 90nm SP design, higher performance may be achieved at smaller technology scale but with a slight power overhead. Comparing the 130nm and 90nm LL designs, Figs.4(b)&4(e), the enhanced throughput that can be garnered with the increased operating frequency while maintaining the same power envelope makes the power savings that can be made by migrating to 90nm even more evident.

Leakage Power Consumption: Fig.4(f) also shows a marked difference between the leakage power for 90nm LL (about 70μW) and SP (about 3mW) library designs. This in itself could be a deciding factor when faced with the choice of these libraries. In comparison with the 130nm library variants, the leakage power has almost doubled in the 90nm generation. This further reinforces the fact that the leakage power which was negligible in technology nodes 130nm and higher has become a significant factor in the 90nm node.

The above experimental results confirm that scaling down from a technology node results in a higher integration level, which, in turn, translates to more functionality in a similarly-sized chip or a smaller footprint for the same design. It boils down to making an informed decision based on how the area savings and performance gain are to be balanced against power savings. Considerable power savings translate to an overall reduction in the total cost of ownership over the lifetime of the system, which is significant for high-performance computing systems as the running cost matches, or even surpasses, the acquisition cost.

F. Power Efficiency

In this section, we embark on evaluating the processor node designs based on another metric, their power efficiency. We have used Dhrystone MIPS divided by the total power, which is the sum of dynamic and leakage power, to calculate this metric. The higher the value of DMIPS/mW, the more power efficient the design is. It can be inferred from Fig.5 that the 90nm LL library stands out from the rest in terms of power efficiency. This is a direct consequence of the low dynamic power and very low leakage power consumption of the 90nm LL design while operating at a reasonable frequency of 200MHz. In the current SpiNNaker CMP, quite a lot of effort has been put in to optimize the processor node to deliver a good computational rate per Watt. The 130nm SP-LL combination has been used to fix this figure in the central region of the plot. When moving to the 90nm technology node, disregarding cost aspects, 90nm LL seems the most appropriate (*greenest*) choice of design library so that the machine itself can be limited to a strict power budget. On the other hand, the 90nm SP library design achieves power efficiency comparable to that of the current 130nm design.

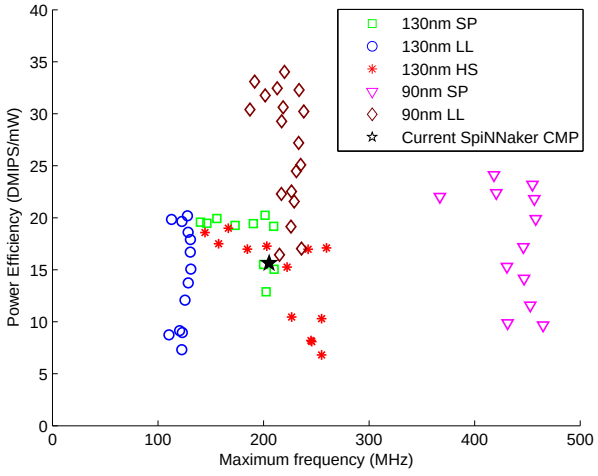


Fig. 5. Power Efficiency of SpiNNaker Processor Node

G. Performance Density

Yet another metric we have used for evaluation is the Performance Density of the various designs. This metric relates to the throughput in relation to the silicon area. We have added 2 more scenarios in this experiment where in the 90nm library designs the Data TCM sizes have been doubled to 128KB. The reasoning behind that is that, at the increased frequency of operation in the 90nm node, the processor should be able to achieve higher throughput, but requires a larger RAM in proportion to this throughput. Fig.6 plots the experimental results of this phase of design exploration. It can be seen from the plot that the added RAMs have lowered the performance density figures with little effect on the maximum frequency achieved. To explore this further, we need to take into account System NoC bandwidth and latency, which we have not done at this stage. Still, the 90nm LL library design is quite attractive in terms of area, performance and power, if we decide to migrate to the 90nm node.

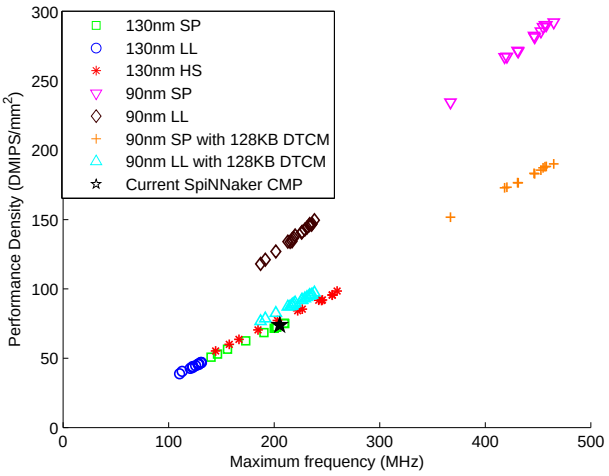


Fig. 6. Performance Density of SpiNNaker Processor Node

V. Design Considerations

A. Balancing Area, Performance, and Power

High-performance and low-power devices are often considered to be at opposite ends of the design spectrum. Finding a balance is a challenge, but several fundamental features helped SpiNNaker achieve the goal of reducing size and also saving significant power while achieving adequate performance. Speed/power/area trade-off analysis with the results of the experiments described above ensured that the design balances throughput with attractive area and power. For coordinated power saving, both the leakage as well as dynamic power dissipation need to be kept within limits. Concurrently, it is desirable to achieve a good-enough performance with minimal area overhead. In our evaluation, we have experimented with implementing the Processor Node with the target libraries individually and based on the results, made a judicious choice for the Processor Node design implemented in the fabricated CMP. So, for the fabricated 130nm SpiNNaker CMP design, a combination of cells from SP (57%) and LL (43%) libraries have been chosen for an area-efficient, low-power design. The ensuing processor node design is highlighted in the Figs.4(a)-(c) with the pentagram. It occupies about 3.75mm^2 of silicon area and functions at about 180MHz with a power consumption of about 20mW.

B. Power Optimization

To optimize the competing goals of throughput and power efficiency, trade-offs are needed. In order to minimize the operating costs of the SpiNNaker machine, the main strategy employed is to reduce the power consumption. The SpiNNaker system is built out of low-power embedded processors and mobile DDR SDRAMs. Rather than employing large high-performance power-hungry processing cores, small cores have been chosen for the system, amortizing the area cost across multiple cores. The SpiNNaker processor nodes operate at a relatively low frequency of about 200MHz. Therefore, they consume much less area and power and it is possible to pack 18 of these processors in a single CMP. The ARM968 processors implement 32-bit fixed-point arithmetic as opposed to the floating-point operations available in general-purpose processors, thereby sacrificing performance for the sake of an energy-efficient architecture. In addition, mechanisms have been built in to power-off idle nodes of the machine, put the processors into *sleep* mode when they are not used for computation and *wake* them up when the need arises.

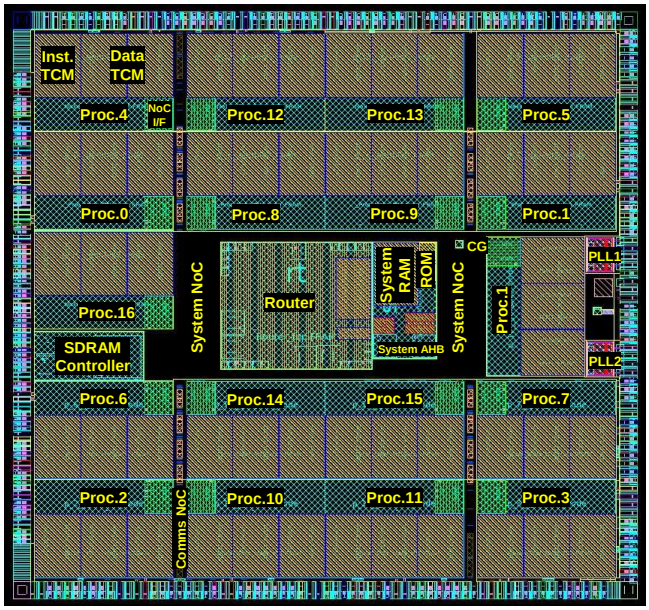


Fig. 7. Plot of the SpiNNaker CMP

C. CMP-Memory Packaging

The SpiNNaker chip is packaged, using wafer-stacking technology, by stacking the SDRAM die on top of the CMP die as seen in Fig.8. Each die is implemented individually: the 128MB SDRAM die is acquired from the memory manufacturer as a Known Good Die and the CMP die is manufactured using a conventional 2D IC process. Coarse integration stacks these two dies in a 300-pin BGA package. The ensuing advantages are improved system performance,

reduced interconnect power dissipation due to smaller parasitic RC than that of conventional on-board (off-chip) wiring, cost-effective packaging/PCB due to integration within the same package and, in particular, reuse of the 2D CMP design with the latest commercially available, possibly higher capacity and speed, SDRAM dies. In future, with the emerging Wide I/O JEDEC standard, a highly parallelized Wide I/O interface with a relatively low memory frequency may be utilized to connect the logic die with the memory die with Through-Silicon Vias. This 3D integration will bring about a dramatic reduction in I/O power, lowering the power consumption of the SpiNNaker CMP still further. Also in tandem with the progress in the DRAM market, a higher capacity SDRAM die, possibly 1GB, may be incorporated into the chip for a higher memory bandwidth to cater for the increased number of faster processor cores that can be crammed in a smaller geometry process in a die of similar size.

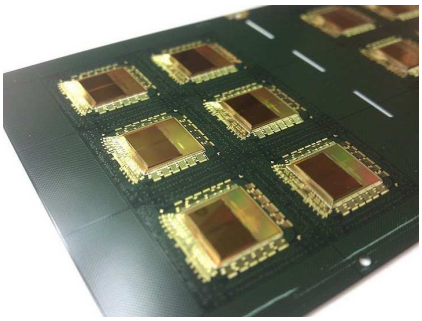


Fig. 8. SpiNNaker chip package substrate with SDRAM

D. Fabricated SpiNNaker CMP

The Processor Node is implemented in two variants consistently tuning to achieve the best performance/power figures. The variants are then replicated nine times each to achieve a fairly regular layout. Fig.7 is a layout picture of the SpiNNaker CMP with its major components highlighted. The standard cell libraries have been augmented with custom-designed asynchronous logic cells and macros. The CMP is fabricated in UMC 130nm L130E process technology and has over 100 million transistors and a power consumption of 1W at 1.2V when all the processor cores are operating at 180MHz. The CMP achieves a peak performance of 3.96 GIPS.

VI. Conclusion and Future Work

Based on the above results, a combination of cells from SP and LL libraries have been chosen for the fabricated 130nm SpiNNaker CMP. By scaling the SpiNNaker CMP down to 90nm, power efficiency is enhanced while maintaining its performance at reduced die size. On the other hand, 130nm process continues to be competitive in terms of manufacturing and implementation cost compared to 90nm. The experimental methods and results presented in this paper, though for a proprietary CMP, are easily extensible to other CMPs, thereby giving valuable insight before making implementation technology decisions. With a view to extending the current design exploration, future research will concentrate on aspects such as memory requirement and bandwidth limitations, design cost and technological options for a future SpiNNaker CMP scaled to an even finer process geometry (possibly 65nm, 45nm and 32nm). The impact of process variability and the resulting reliability issues in finer technology nodes is another possible avenue of research.

Though SpiNNaker is an application-specific architecture, it can still be used for running applications such as ray-tracing, protein folding etc. which are outside the purview of neuroscience applications. For the neural applications, major advantage is gained in the flexibility afforded by the software implementation of neural models in the processor core and the asynchronous communications infrastructure.

Acknowledgements

The SpiNNaker project is supported by the Engineering and Physical Sciences Research Council of the UK, through Grants EP/D07908X/1 and EP/G015740/1, and also by ARM and Silistix. The authors appreciate the support of these sponsors and industrial partners. The die photo in Fig. 8 is courtesy of Unisem Europe Ltd.

References

[1] G. Moore, "Cramming More Components onto Integrated Circuits," *Electronics*, vol. 38, no. 8, pp. 114–117, April 19, 1965.
[2] R. Dennard *et al.*, "Design of Ion-implanted MOSFETs with Very Small Physical Dimensions," *IEEE JSSC*, vol. SC-9, no. 5, pp. 256–268, 1974.
[3] "ITRS 2010 Update," *http://www.itrs.net/Links/2010ITRS/Home2010.htm*.
[4] "ARM968E-S," *http://www.arm.com/products/processors/classic/arm9/arm968.php*.
[5] W. Huang *et al.*, "Scaling with Design Constraints: Predicting the Future of Big Chips," *Micro, IEEE*, vol. 31, no. 4, pp. 16–29, July-Aug. 2011.
[6] E. Chung *et al.*, "Single-Chip Heterogeneous Computing: Does the Future Include Custom Logic, FPGAs, and GPGPUs?" in *Proc. 43rd Annual IEEE/ACM Int. Symp. Microarchitecture*, 2010, pp. 225–236.
[7] H. Markram, "The Blue Brain Project," *Nature Reviews Neuroscience*, vol. 7, no. 2, pp. 153–160, Feb. 2006.
[8] R. Ananthanarayanan *et al.*, "The Cat is Out of the Bag: Cortical Simulations with 10^9 Neurons, 10^{13} Synapses," in *Proc. Conf. High Performance Computing Networking, Storage and Analysis*, 2009, pp. 63:1–63:12.
[9] IBM Blue Gene Team, "Overview of the IBM Blue Gene/P Project," *IBM J. Research & Development*, vol. 52, no. 1/2, pp. 199–220, 2008.
[10] R. Haring *et al.*, "The IBM Blue Gene/Q Compute Chip with SIMD Floating-Point Unit," *Hot Chips 23*, 2011.
[11] D. Truong *et al.*, "A 167-Processor Computational Platform in 65nm CMOS," *IEEE JSSC*, vol. 44, no. 4, pp. 1130–1144, Apr. 2009.
[12] S. Vangal *et al.*, "An 80-Tile Sub-100W TeraFLOPS Processor in 65-nm CMOS," *IEEE JSSC*, vol. 43, no. 1, pp. 29–41, Jan. 2008.
[13] S. Bell *et al.*, "TILE64 Processor: A 64-Core SoC with Mesh Interconnect," in *Dig. Tech. Papers. IEEE Int. SSC Conf.*, 2008, pp. 88–598.
[14] Tilera, "TILE-Gx Processor Family," *http://www.tilera.com/products/processors/TILE-Gx_Family*.
[15] S. Furber and S. Temple, "Neural Systems Engineering," *J. Roy. Soc. Interface*, vol. 4, no. 13, pp. 193–206, Apr. 2007.
[16] L.Plana *et al.*, "A GALS Infrastructure for a Massively Parallel Multiprocessor," *IEEE D&T Computers*, vol. 24, no. 5, pp. 454–463, 2007.
[17] L. Plana *et al.*, "SpiNNaker: Design and Implementation of a GALS Multicore System-on-Chip," *ACM J. Emerging Technologies in Computing Systems*, vol. 7, no. 4, pp. 17:1–17:18, 2011.
[18] J. Bainbridge and S. Furber, "CHAIN: A Delay-Insensitive Chip Area Interconnect," *IEEE Micro*, vol. 22, no. 5, pp. 16–23, 2002.
[19] ARM, "AMBA AXI Protocol Specification, Rev.1.0," *http://www.arm.com/products/system-ip/amba/*.
[20] L. Plana *et al.*, "An On-Chip and Inter-Chip Communications Network for the SpiNNaker Massively-Parallel Neural Net Simulator," in *Proc. ACM/IEEE Int. Symp. Networks-on-Chip*, 2008, pp. 215–216.

A Unique Design methodology to generate reconfigurable Analog ICs with simplified Design Cycle

G. Kapur, S. Mittal, C.M.Markan, V.P.Pyara

Abstract: **We propose a new design methodology, which simplifies the analog design cycle and introduces reconfigurable and accurate prototype of any analog IC. A comprehensive design methodology is being developed that it makes Analog IC customizable and adaptable by a field user. For example we have considered a most fundamental analog element, the operational amplifier (op-amp). The basic design of a two-stage compensated op-amp is firstly analyzed to estimate the basic circuit configuration which verifies its functionality. Secondly using its small signal equivalent model, specifications are being derived in terms of threshold voltage of transistors, present in the circuit. Indeed, with the help of floating gate transistors, which has a feature of post fabrication programmability of its threshold voltage, an accurate and reconfigurable prototype of a design can be obtained. The circuit is simulated using BSIM3 level49 MOSFET models in T-Spice 0.35µm CMOS process. Specifications such as slew rate, gain, CMRR, PSRR, gain bandwidth, input range and offsets are being derived and estimated their sensitivity with respect to threshold voltage of respective floating-gate transistors. The simulated results demonstrate that by programming threshold voltages, fine tuning of specifications with wide spectrum can be achieved. The designs fabricated using our methodology can be adaptive to any desired value of specification with very high precision (about 13bit programming precision can be attain). It can also make Analog ICs immune to most drawbacks like processes variations, device degradation by introducing new possibilities such as self correction and adaptability.**

Keywords: **Accuracy, Floating-gates, Field Programmable analog array, Operational Amplifier, Reconfigurable, Specifications, Threshold voltage.**

I. INTRODUCTION

ANALOG circuits from long time are getting overshadowed by digital designs however real world consists of analog, all electronic systems ultimately have to interface at input and output with the analog signals. The portable electronic devices also require higher level of integration with lower power consumption, which pervades the need of analog. Moreover, with analog devices there would be no use of huge ADCs/DACs, decision box (filters) and can have easy interfacing with real world. But still analog devices are less preferred because analog circuits are not easy and well defined like digital as there is no unique answer. Moreover, there is a knowledge mismatch between books and industrial training kits for analog circuit designs because teaching courses focus on analysis of analog circuits rather than on their designing. Design is actually reverse of analysis. As in industry one start with the answer, which are the specifications, and one has to work back to what circuit configuration to begin with and what component values to use. This mismatch of knowledge in books and industry create problems to fresh design graduates in industry. Dr. R.D.Middlebrook, Prof. of Electrical Engineering, California Institute of Technology, Pasadena, also states that "I believe design can be integrated into analysis at a much fundamental and detailed level" [1]. Indeed analog design cycle consists of many iterative steps. Analog designers have to maintain a tradeoff between fabrication time and costs. Designers also have to maintain a tradeoff between accurate and optimized prototype of a design. Thus any analog design turnaround time is about three months. As well as, each new derivative (i.e. the same design with new value of specification) requires going through the complete design cycle again. Such limitations with analog IC design can be outshined by introducing adaptability and reconfigure ability in the designs. Recent efforts have introduced a fair degree of design automation and field programmability in the form of Field Programmable Analog Arrays (FPAA) [2, 3, 4 and 5]. These FPAAs have an array of analog components over a range of specifications that can be switched in/out of a design on the same lines as digital devices are in a FPGA. These chips can be programmed very close to desired functionality; besides the inbuilt device redundancy also helps to make it fault tolerant to a certain extent.

Despite the phenomenal progress a basic question still eludes analog ASIC designers. Can we build generic analog devices that can be customized to desired specification by the user in a field programmable manner? Therefore we would like to propose a design methodology for analog ICs which integrate designing and analysis at a very fundamental level. As, we start designing any circuit at very basic level that is, start with a simple equivalent small signal model and some basic quantitative relationships that can establish the required functionality of the design with the help of sequence of steps. On the basis of design functionality, specifications of the design will be derived in terms of threshold voltage of the respective transistors used in the design. The design with the basic details of sizing and biasing condition can be fabricated. Moreover with the help of post fabrication programmability of transistor's threshold voltage in floating gate transistors, we can produce accurate prototype of the design which are customizable to desired specification by a field user [6, 7 and 8]. Prof. Keith Hipel, a system designer said, "Any new methodology is required to solve application and application in turn is used to prove the proposed methodology. Therefore to prove our methodology we have considered an Operational amplifier, a fundamental building block in analog integrated circuit design. Various programmable op-amps with programming circuitry exists [9, 10 and

11], however, we proposes a methodology to develop a field programmable, reconfigurable and adaptive op-amp. Using our design cycle we have analyzed, designed and estimated the circuit functionality with basic circuit configuration and its specifications in terms of threshold voltages of respective floating-gate transistors. Furthermore these specifications can adapt with high precision to any desired value by field user. Next section will illustrate our proposed design cycle with brief description of field programming of floating-gate transistor. Consequently with an example of op-amp's basic design, we propose a design methodology for op-amp designing in the section III, followed by its simulation results.

II. New Design Methodology

The new design methodology to introduce field programmability, reconfigure ability and adaptability in analog circuit designs, using concept of floating gate transistors leads to a new and simpler analog design cycle. The flow chart of the new design cycle is shown in Figure 1. Algorithm of the design flow is being illustrated in next section, however field and indirect programmability of floating gate transistor is demonstrated in the next sub-section.

A. *Proposed New Analog Design Cycle*

The usual design flow for analog designs consists of many iterative steps and verification tools. For desired specification, designing starts from first cut design using classroom equations. To optimize the circuit and to derive optimized W/Ls for individual transistors before the circuit is ready for fabrication sophisticated simulation tools with accurate yet complex simulation device models are required. Wherein, analog designers have to sustain a tradeoff between the prescribed design specifications with design sizing and biasing conditions while maintaining its functionality. Time and cost are very important gradient in such tradeoff, especially in determining the accurate and optimized prototype of a design. Therefore we propose an alternative paradigm in design of analog devices wherein focus would be to reduce the overdependence on determining accurate W/L of individual transistors prior to fabrication. Our proposed design cycle starts with the designing analog circuits from fundamental level, defining circuit specifications in terms of threshold voltage of the transistors and programming it after fabrication to develop an accurate and stable prototype. Hence it proves to pave a way of analog-friendly environment in fundamental teaching, graduate level and at industry level. Algorithm of the proposed design cycle is as follows:

Steps:

1. Analyze circuit with the help of block diagram to establish desired functionality.
2. Simulate the circuit to check its functionality with basic sizing and biasing conditions.
3. Design the equivalent small-signal model of the circuit.
4. Derive specifications in terms of threshold voltage of transistors.
5. Analyze sensitivity of each specification with respect to respective thresholds.
6. Simulate the circuit to check the sensitivity of each specification.
7. Layout creation and verification of the circuit with basic sizing and biasing conditions.
8. Then after testing and extraction, fabricate the design.
9. Program the transistor's thresholds to adjust the desired specifications with huge accuracy.

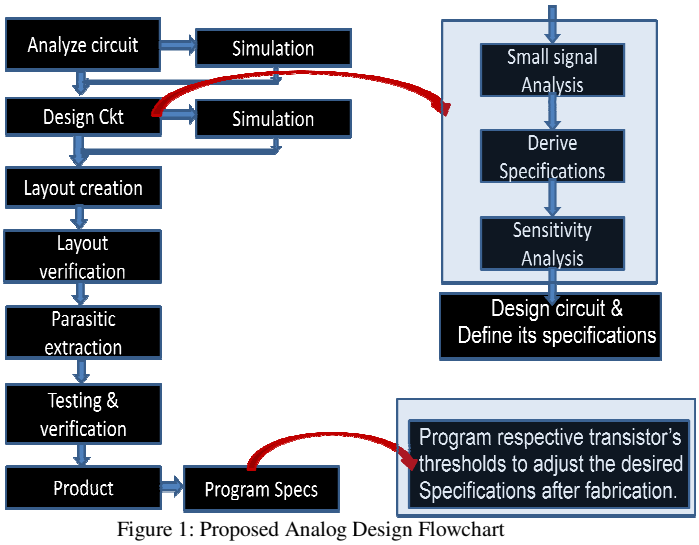


Figure 1: Proposed Analog Design Flowchart

B. *Floating-gate Transistor*

Floating-gate MOS transistors are conventional MOS transistors wherein memory is stored in the form of charge trapped on floating-gate, affecting its threshold voltage. Two antagonistic quantum mechanical transfer processes, viz. injection and tunneling, alter the trapped charge on a floating gate. As these processes can occur during normal operation (indirect programming [13]), it leads additional attributes to the FGMOS transistors such as non volatile analog memory storage on floating-gate, locally computed bidirectional memory updates and memory modification during normal transistor operation.

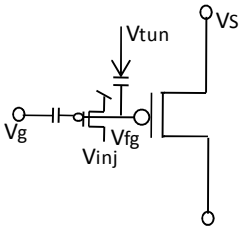


Figure 2: Symbolic representation of a normal MOS with indirectly programmable floating gate using injection and tunneling

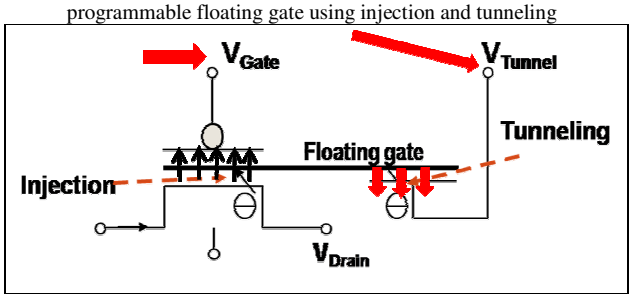


Figure 3: Pictorial representation of non-volatile, high precision, indirect and on-chip programming of Floating-gate MOS.

Tunneling

Charge is added to the floating gate by removing electron from it by means of Fowler-Nordheim tunneling across oxide capacitor. This shifts the curve (Figure 4) to the right or in other words threshold voltage of the transistor increases.

Injection

Charge is removed from the floating-gate by adding electron on it by impact-ionized hot electron injection from the channel to the floating gate across the thin gate oxide. This shifts the curve (Figure 4) to the left or in other words threshold voltage of the transistor decreases.

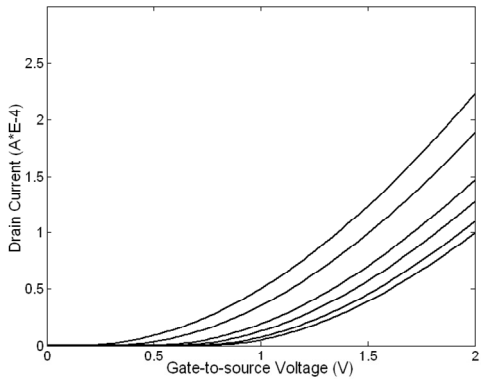


Figure 4: Output Characteristics of FGMOS transistor

With such simpler design cycle and field programmability feature of FGMOS, any analog circuit can be designed and customized at the field user end. It can reduce design time and reconfigurable designs reduce costs of fabrication. Optimization of the design can be performed at the layout creation as only basic design is to be fabricated. Moreover, accurate prototype of the design can be obtained after fabrication. Hence it can produce highly accurate and optimized designs without any compromise. Indeed using our proposed design cycle, a design methodology can be developed for any analog circuit, i.e. which specification is most sensitive to which FGMOS threshold programming. Consequently such derived and estimated design methodology for any circuit can be used to generate accurate as well as reconfigurable prototypes of the design. To verify our design cycle we have designed various applications. For simplicity we are representing the design flow on the most fundamental analog element and demonstrated its derived design methodology.

III. Application

A. *Analyses of Basic Op-amp Design*

Operational amplifier is a fundamental building block in analog integrated circuit design. A diagram of the two stage compensated op-amp with output buffer is shown in Figure 5, where first stage of an op-amp is a differential amplifier. This is followed by another gain stage, such as a common source stage, and finally an output buffer. The open loop gain of an op-amp is infinite so a compensating feedback loop which can make gain of op-amp finite and easily controlled by the resistance is used in the feedback path of op-amp and current biasing of differential amplifier is done to reduce the common mode gain and improves the CMMR ratio of op-amp. Op-amp is a voltage controlled voltage source in which output voltage depends upon the voltage difference in between inverting pin and non-inverting pins of an op-amp and with the help of op-amp various applications can be developed. To verify our proposed design cycle we have analyzed the op-amp circuit design with basic circuit configuration (sizing and biasing conditions) for desired basic functionality using simulation results as shown in Figure 6. It shows voltage gain equals to 34db and 3db bandwidth equals to 30 kHz (represented by red curve in Figure 6). Subsequently the circuit is designed using its equivalent small signal model.

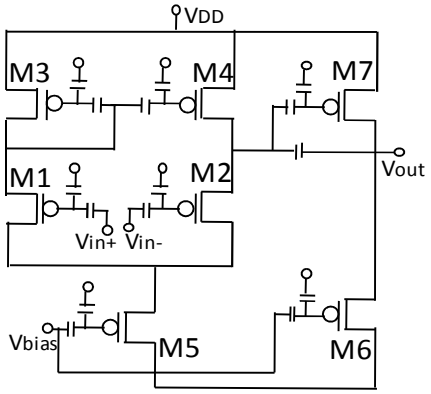


Figure 5: Proposed Op-amp Circuit using FGMOS

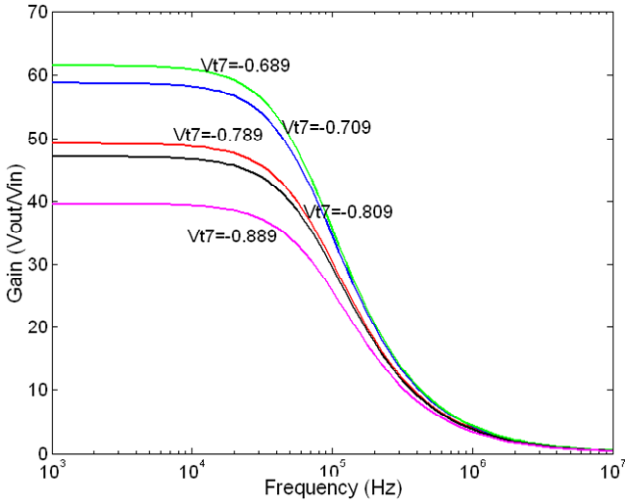


Figure 6: Plot representing voltage gain of proposed op-amp which shows programmability with FGMOS M7 (Vt7, M7 threshold voltage)

B. *Design of Proposed Op-amp Specifications*

Designing of proposed op-amp is demonstrated with the help of its equivalent small signal model. The circuit consists of a differential pair (M1 and M2), current mirror (M3 and M4) for biasing and a biasing MOS M5 along with output buffer (M6 and M7). From small signal equivalent model on applying nodal analysis, voltage gain is derived in terms of MOS transconductances and output resistances, which in turn can be substituted to generate gain in terms of MOS threshold voltages (V_{t1} and V_{t7}). The voltage gain is given by:

$$A_v = \frac{g_{m1}(g_{m7} - S_{cc})R_1R_2}{R_1R_2S^2(C_1C_2 + C_C(C_1 + C_2)) + S\begin{pmatrix}R_1R_2C_Cg_{m7} + R_1(C_1 + C_C)\\ +R_2(C_2 + C_C)\end{pmatrix} + 1} \quad (1)$$

Transconductances in equation 1 can be expressed in terms of square root of their respective saturation drain currents: $g_m = \frac{k_n' W/L}{2} \left(\sqrt{\frac{I_d}{k_n' /2W/L}}\right)$ (2)

However the slew rate is given by:

$$\frac{I_{D5}}{C_C} = \frac{K_{n5}(V_{GS5}-V_{T5})2}{2C_C} \quad (3)$$

This shows direct dependency of slew rate on Vt5 i.e. threshold voltage of FGMOS M5. Moreover, the common mode rejection ratio is the differential gain by common mode gain which is derived as:

$$CMRR = \frac{2g_{m1}g_{m2}}{(g_{o3}+g_{o1})g_{o5}} \quad (4)$$

Indeed these transconductances can be substituted in terms of the respective threshold voltages. Hence CMRR shows dependency on $V_{t1,}$ V_{t2} and $V_{t5.}$ The PSRR can be written as

$$PSRR = \frac{A_v(V_{dd}=0)}{A_{dd}(V_{in}=0)} \quad (5)$$

However for positive PSRR two stage op-amp is connected in the unity-gain mode with an ac ripple of V_{dd} on the positive power supply. It is given by:

$$+ve\ PSRR = \frac{2g_{m2}g_{m3}g_{m6}}{(g_{o2}+g_{o4})(2g_{m3}g_{o7}-g_{m6}g_{o5})} \quad (6)$$

Similarly negative PSRR depends on where the voltage V_{bias} is connected and is given by:

$$-vePSRR = \frac{g_{m2}g_{m6}}{(g_{o2}+g_{o4})g_{o6}} \quad (7)$$

The 3db bandwidth or gain bandwidth of the circuit is given by:

$$w_{3dB} = \frac{g_{m1}}{A_v C_C} \quad (8)$$

Poles of gain:

$$P_1 = \frac{-1}{C_C R_1(1+g_{m7}R_2)}, P_2 = \frac{-g_{m7}C_C}{C_2C_1+C_2C_C+C_CC_1} \quad (9)$$

Zero of gain:

$$z = \frac{g_{m7}}{C_C} \quad (10)$$

Output impedance form the model is being derived and is given by:

$$Z_{out} = \frac{R_2(1+R_1S(C_1+C_C))}{S^2R_1R_2\big(C_1C_2+C_C(C_1+C_2)\big)+S(R_1R_2C_Cg_{m7} +C_2R_2+C_CR_2+C_1R_1+C_CR_1)+1} \quad (11)$$

C. Derive Sensitivity of each Specification

The design of proposed op-amp using floating gate transistors is shown in Figure 5. To introduce field programmability in the design, all transistors in design are being replaced by indirectly programmable floating gate transistors as represented symbolically in the Figure 3 and represented symbolically in Figure 4. However basic characteristics of the design, mentioned in last section, can be expressed in terms of threshold voltage by replacing transconductance of the respective transistors with their threshold voltages. As the threshold voltage of these floating-gate transistors can be programmed on-chip after fabrication (as illustrated in section II), hence basic characteristics of the design can be adjusted after fabrication. Thus, such design methodology save several simulation steps and can produce accurate prototype of the op-amp design with specific characteristics. To obtain its design methodology, sensitivity of each specification with respect to respective threshold voltages is derived. Using equation (1) voltage gain shows dependence on V_{t5} and V_{t7}. Sensitivity of voltage gain with respect to threshold voltage of FGMOS M1 is given by:

$$S_{Vt1}^{Av} = \frac{-V_{T1}}{(V_{GS1}-V_{T1})} \quad (12)$$

Sensitivity of gain w.r.t V_{t7} is given by:

$$S_{Vt7}^{Av} = \frac{-V_{T7}}{V_{GS7}-V_{T7}-SC_C} - \frac{SR_1R_2C_CV_{T7}K_7}{S^2R_1R_2\big(C_1C_2+C_C(C_1+C_2)\big)+S\begin{pmatrix}R_1R_2C_C(V_{GS7}-V_{T7})K_7\\+R_1(C_1+C_C)+R_2(C_2+C_C)\end{pmatrix}+1} \quad (13)$$

Sensitivity of pole P_1 of voltage gain with respect to V_{t7}:-

$$S_{Vt7}^{P1} = \frac{R_2V_{T7}K_7}{1+K_7(V_{GS7}-V_{T7})R_2} \quad (14)$$

Sensitivity of pole P_2 of voltage gain with respect to V_{t7}:-

$$S_{Vt7}^{P2} = \frac{-V_{T7}}{V_{GS7}-V_{T7}} \quad (15)$$

Sensitivity of zero Z of voltage gain with respect toV_{t7}:-

$$S_{VT7}^{Z} = \frac{-V_{T7}}{V_{GS7}-V_{T7}} \quad (16)$$

The Figure 6 illustrate that the gain can be programmed using FGMOS M7 as plot demonstrates that the gain varies at different M7 threshold voltage, V_{t7}. With decrease in threshold voltage, $V_{t7,}$ gain of the proposed op-amp increases. Similarly rest of the important specifications like slew rate, CMRR, offsets have been derived in terms of threshold voltage of the respective FGMOSs and sensitivity of each will be analyzed. The output impedance from equation (11) is now considered and re-derived with respect thresholds and derives sensitivity w.r.t V_{t7} and is given by:

$$S_{VT7}^{Zo} = \frac{SR_1R_2C_Ck_7V_{T7}}{S^2R_1R_2\big(C_1C_2+C_C(C_1+C_2)\big)+S\begin{pmatrix}R_1R_2C_CK_7(V_{GS7}-V_{T7})+\\R_1(C_1+C_C)+R_2(C_2+C_C)\end{pmatrix}+1}$$

Sensitivity of Slew rate with respect to V_{t5} is given by:

$$S_{VT5}^{SR} = \frac{-2V_{T5}}{V_{GS5}-V_{T5}} \quad (18)$$

Similarly sensitivity of CMRR with respect to V_{t1} is given by:

$$S_{VT1}^{CMRR} = \frac{-V_{T1}}{V_{GS1}-V_{T1}} \quad (19)$$

And sensitivity of CMRR with respect to V_{t3} is given by:

$$S_{VT3}^{CMRR} = \frac{-V_{T3}}{V_{GS3}-V_{T3}} \quad (20)$$

Moreover sensitivity of 3-dB Bandwidth with respect to V_{t1} is given by:

$$S_{VT1}^{w3dB} = \frac{-V_{T1}}{V_{GS1}-V_{T1}} \quad (21)$$

Sensitivity of negative PSRR with respect to V_{t2} is given by:

$$S_{VT2}^{nPSRR} = \frac{-V_{T2}}{V_{GS2}-V_{T2}} \quad (22)$$

Similarly sensitivity of negative PSRR with respect to V_{t6} is given by:

$$S_{VT6}^{nPSRR} = \frac{-V_{T6}}{V_{GS6}-V_{T6}} \quad (23)$$

And sensitivity of positive PSRR with respect to V_{t2} is given by:

$$S_{VT2}^{pPSRR} = \frac{-V_{T2}}{V_{GS2}-V_{T2}} \quad (24)$$

Therefore the derived specifications of the proposed op-amp design can be programmed using floating gate transistors after fabrication. Simulation results illustrating such programming or sensitivity of respective specification while considering each FGMOS individually. The plots in Figure 7, 8, 9 and 10 represent the effect of each threshold voltage on all considered op-amp specifications. Figure 7 represents sensitivity of specifications with respect to V_{t1} and Figure 8 represent sensitivity of specifications with respect to $V_{t3.}$ Moreover, Figure 9 represent sensitivity of specifications with respect to V_{t5} and Figure 10 represent sensitivity of specifications with respect to $V_{t7.}$ Similarly sensitivity of specifications with respect to each FGMOS threshold can be obtained using simulation results. From such plots a design methodology (steps for on-chip programming) for the design can be developed which illustrate that which specification can be adjusted with which FGMOS threshold. Or in other words design methodology states that the most sensitivity pair of each specification with respective FGMOS threshold. The most sensitive pairs along with compensation of rest of the FGMOS thresholds determine the steps to program a specific op-amp specification after fabrication.

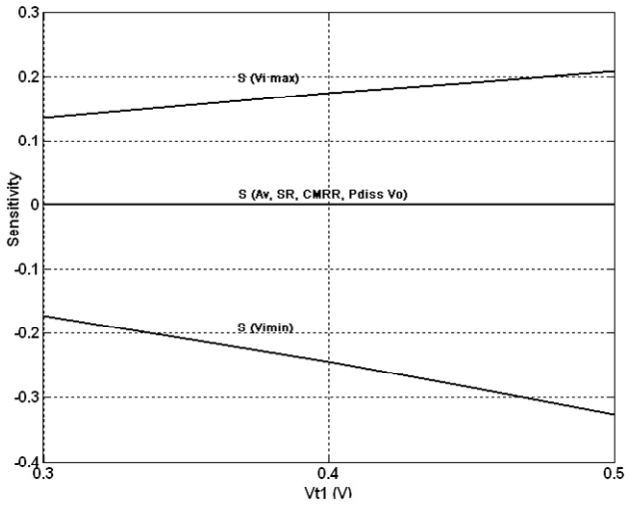


Figure 7: Plot representing sensitivity of all specification with respect to V_{t1}

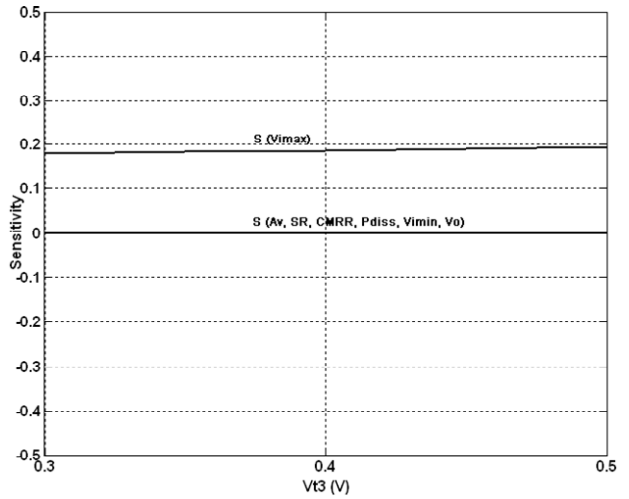


Figure 8: Plot representing sensitivity of all specification with respect to V_{t3}

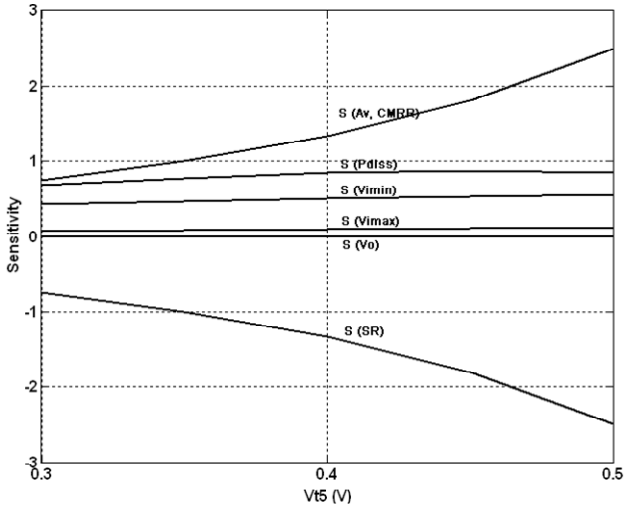


Figure 9: Plot representing sensitivity of all specification with respect to V_{t5}

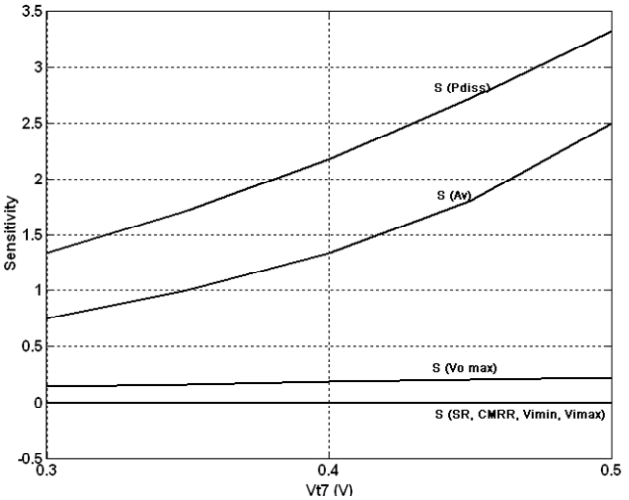


Figure 10: Plot representing sensitivity of all specification with respect to $V_{t7.}$

D. Proposed Op-amp Design Methodology

With the help of sensitivity equations as illustrated in last section and sensitivity plots with respect to each FGMOS, a design methodology for proposed op-amp have been developed. It is observed that each specification shows dependency on more than one FGMOS threshold. So with some design modification as well as iterative simulations each specification can be estimated with respect to only one FGMOS threshold, compensating rest all the FGMOS thresholds. Like voltage gain is sensitive to V_{t5} and V_{t7} however most sensitive to V_{t7} while compensating $V_{t5.}$ However slew rate shows most sensitivity with $V_{t5.}$ Similarly with analyzing sensitivity of each specification a design methodology for the proposed op-amp is developed. The design methodology is as shown below:

Identified V_T - Spec (most sensitive) pairs

- Gain - V_{T7}
- Slew Rate - V_{T5}
- Poles and zeros of gain - V_{T7}
- CMRR - V_{T1}
- Output impedance - V_{T7}
- PSRR - V_{T2}
- Input Range $V_{i(min)}$ - V_{T1}
- Input Range $V_{i(max)}$ - V_{T3}

- Offset voltage - V_{T2}
- Output Range - V_{T6}

The most sensitive pair of specification with respective FGMOS threshold keeping sensitivity of rest of the FGMOS threshold's constant, a specification can be programmed after fabrication. Thus the proposed op-amp design using FGMOSs can be fabricated with basic circuit configuration. And its specifications can be adjusted to desired value with accuracy of about 13 bit programming resolution (as claimed in [12]). In addition to it, any op-amp design with new values of specification can be reconfigurable on the same design. Moreover, design variations due to noise, temperature change, parasitic, etc. can also be adjusted adaptively in the design after fabrication.

IV. CONCLUSION

The proposed op-amp design using indirectly programmable floating gate transistors is simulated using BSIM3 level49 MOSFET models in T-Spice, 0.35µm CMOS process. Using small signal analysis of the design, voltage gain, poles and zeros of voltage gain, 3-db bandwidth, offset voltages, output resistances, CMMR, slew rate, output voltage range are derived. Sensitivity of each characteristic with respect to threshold voltage of the respective transistors is derived, while considering each one individually. The specifications are obtained from simulations and verified with theoretical results. The graphs showing programming for voltage gain, poles and zeros, 3-db bandwidth, output resistance, offset voltage, slew rate, CMRR with respect to respective dominant floating gate transistor programming have been generated. Sensitivity analysis for each characteristic has been illustrated with the help of graphs. Hence, characteristics of our proposed Op-amp design can be tuned after fabrication in small range but with very high precision (about 13bit programming resolution can be obtained using floating gate transistors). An accurate and reconfigurable op-amp design can be fabricated with less design time and costs and without any additional programming circuitry. Moreover, this methodology to introduce field programmability can be extended to various analog circuit designs. Instead of multiple steps of simulations, testing and verification, accurate prototype design can be obtained by tuning the specifications after fabrication with the help of floating-gate transistors. However, there is some limitation of range of tuning but still such systematic approach can bring drastic revolution in analog circuit design.

V. REFERENCES

[1] R.D.Middlebrook, "Analog Design needs a change in perspective", an article in *Electronic Engineering Times*, 17 Dec, 1990.

[2] E.K.F Lee, P.G Gulak, "Field programmable analogue array based on MOSFET transconductors", *Electronics Letters*, Vol. 28, Issue 1, pp. 28 – 29, 1992.

[3] Analog Integrated Circuits and Signal Processing, "Special Issue on Field-Programmable Analog Arrays", *Kluwer publishers*, Vol. 17, Numbers 1-2, September 1998.

[4] B. Pankiewicz, M. Wojcikowski, S. Szczepanski, and Y. Sun, "A field programmable analog array for CMOS continuous-time OTA-C filter applications," *IEEE J. Solid-State Circuits*, Vol. 37, no. 2, pp. 125–136, Feb. 2002.

[5] J. Becker, F. Henrici, S. Trendelenburg, M. Ortmanns, and Y. Manoli, "A field-programmable analog array of 55 digitally tunable OTAs in a hexagonal lattice," *IEEE J. Solid-State Circuits,* Vol. 43, no. 12, pp. 2759–2768, Dec. 2008.

[6] D. W. Graham, E. Farquhar, B. Degnan, C. Gordon, and P. Hasler, "Indirect programming of floating-gate transistors," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, Kobe, Japan, pp. 2172 – 2175, May 2005.

[7] K. Rahimi, C. Diorio, C. Hernandez, M.D.Brockhausen, "A Simulation model for floating gate MOS synapse transistors," *IEEE International Symposium on Circuit and Systems,* Vol. 2, pg. 532-535, Aug, 2002.

[8] T.S.Hall, C.M. Twigg, J.D.Gray, P.Hasler, D.V.Anderson, "Large Scale field programmable analog array for analog signal processing." *IEEE trans. on circuit and systems-I, Regular paper*, Vol.52, No.11, pg. 2298-2308, Nov, 2005.

[9] J.M Shin, K.S. Yoon, "Design of programmable slew rate op-amp", *published in proceedings of 37th Midwest symposium on circuit and systems,* Vol.1, pp.142-146, Aug, 1994.

[10] R. Hogervorst, S.M. Safai, J.P.Tero, J.H.Huijsing, "A programmable 3-V CMOS rail-to-rail op-amp with gain boosting for driving heavy resistive loads." *published in proceedings of IEEE international symposium on circuit and systems, ISCAS1995*, Vol.2, pp.1544-1547, May 1995.

[11] S.C. Delacruz, M. Delos Ruges, T.C. Gaffud, T.Abaya, M. Gusad, M.D. Rosaler, "Design and implementation of operational amplifier with programmable characteristics in a 90nm CMOS process", *published in proceedings of European IEEE conference on circuit theory and designs, ECCCTD 2009*, pp. 209-212, Aug, 2009.

[12] Y. L. Wong, M.H Cohen, P. A. Abshire, "A 1.2 GHz adaptive floating gate comparator with 13-bit resolution", *published in proceeding of ISCAS 2005, IEEE(CAS)*, pg 6146-49, Vol.6, 2005.

An Automated Design Approach of Dependable VLSI Using Improved Canary FF

Ken Yano[1]
Fukuoka University
Fukuoka, Japan
yano0828@fukuoka-u.ac.jp

Takahito Yoshiki
Fukuoka University
Fukuoka, Japan
td102017@cis.fukuoka-u.ac.jp

Takanori Hayashida
Fukuoka University
Fukuoka, Japan
thayashida@fukuoka-u.ac.jp

Toshinori Sato[1]
Fukuoka University
Fukuoka, Japan
toshinori.sato@computer.org

Abstract— **The demand of power saving and highly efficient LSI has increased by the miniaturization of semiconductor technology and the spread of portable device such as a mobile phone. We propose an automated design approach of dependable VLSI that address the timing error caused by the variation in the element characteristic in a deep submicron domain, aging and soft error. The improved canary FF described in this paper reduces about 8% of power consumption compared with the original canary FF. Using the existing standard cell library, the canary FF is mapped automatically to gate cells and its influence on chip area and power consumption is investigated.**

Keywords-***Flip-flops, timing error, dependable system, design automation***

I. INTRODUCTION

Due to the miniaturization of semiconductor technology and the spread of portable devices such as a mobile phone, further improvement of speed and power-saving LSI has come to be called for. The design method which takes the worst case scenario makes the design margin too large because of the parameter variation of the elements in the deep submicron domain has bad influence for performance and power consumption. Moreover aging and soft error would cause the timing error which is not assumed in the design phase and it has become one of the main factors of malfunction of an integrated circuit.

In this paper, the design technique of dependable VLSI which uses canary FF (CFF) by concentrating on the typical case is proposed. First, the technique of limiting the positions where to replace conventional DFF with CFF by considering the timing error information acquired from the worst case design is described. The proposed method is evaluated on two sample microprocessors. We also propose improved canary FF (iCFF) which is power saving and requires smaller area is introduced by optimizing transistor level circuit design. It is evaluated that the improved canary FF can decrease power consumption by about 8% that of canary FF. This paper further examines area and power overhead by canary FF by introducing a novel cell mapping technique to implement canary FF. Note that the improved canary FF is not used for the analysis of are and power overhead since the cell layout is under development. Canary FF can be used for reducing power dissipation in combination with DVS (Dynamic Voltage Scaling) [2] or for timing error detection like Razor FF[4] or for soft error protection. It is studied that in nanoscale CMOS domain, soft error will just not impact SRAMs but latches/flip-flops and combinational logic as well [7].

After introducing the related research of timing error detecting FF in Section II, Section III describes the transistor level circuit structure of improved canary FF. Detailed implementation method of canary FF is described in Section IV and in Section V the area and power overhead by canary FF are examined. Lastly further works and direction of possible studies are described.

In the following discussion, we use "CFF" for canary FF and "iCFF" for improved canary FF when the meaning is not ambiguous from the context.

II. TIMING ERROR DETECTING FLIP FLOP

Many researches have been done for detecting a timing error of integrated circuit. There are mainly two methods in order to improve the reliability of a circuit, one uses spatial redundancy and the other uses time redundancy. This paper describes a design approach for detection of timing error using CFF which adopts spatial redundancy and the method to integrate dependable LSI by utilizing it. We have so far reported the technique of reducing design margins by the variation of the element on LSI by using canary FF[2]. In recent years the degree of complexity of semiconductor process is increased and highly efficient and low power consumption LSI is demanded. The problem of variation in the device characteristic on a chip is emerging and the

[1] CREST, Japan Science and Technology Agency

TABLE I
Rohm0.18 micron standard cell library

Vdd	Temp.	Process
1.6V	85	Max
1.8V	25	Typ
2.0V	-40	Min

design method which takes the variation into consideration is indispensable. The change of the device characteristic by aged deterioration which is difficult to measure in the design phase and soft error are also serious issues. Flip-flops which detect a timing error such as Razor Flip-Flops (RazorFF) [4],

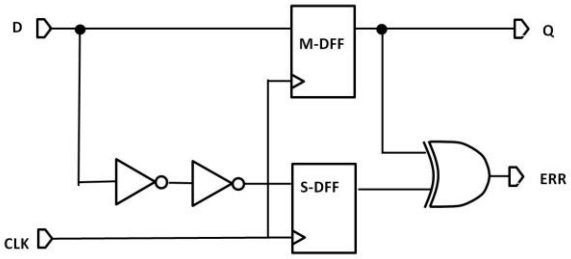


Fig. 1. Canary FF

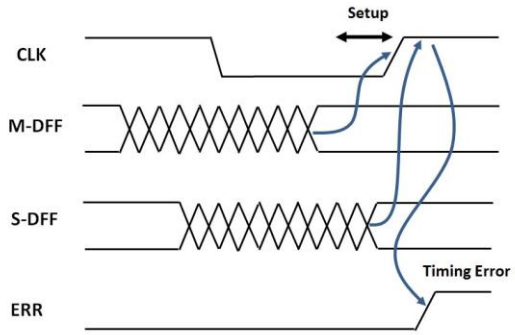


Fig. 2. Conceptual timing diagrams showing timing error detection

RazorII [3], Phase-adjustable Error Detection Flip-Flop (PEDFF), and Delay-Compensation Flip-Flops (DCFF) have been proposed. Moreover against soft error, redundant FF such as Built-In Soft Error Resilience (BISER) [7] and Bistable Cross-coupled Dual Modular Redundancy [5] are proposed. There are problems when using redundant FF, such as the increase of the power consumption and the cell area. It has been reported that the timing error detecting FF requires two to three times cell area compared with the conventional FF because it consists of shadow FF(Latch), a delay element and an error judging circuit in addition to main FF. On the other hand, there is a proposal to reduce the power consumption of the whole chip by utilizing the timing error detecting FF in combination with DVS[2].

III. IMPROVED CANARY FLIP FLOP

Canary FF has been proposed for detecting timing error and its circuit block is shown in Fig.1. In this paper, in order to consider implementation of LSI using canary FF, the

TABLE II
Power analysis of DFF,CFF and iCFF
(Clock cycle: 20ns)

	DFF	CFF	iCFF
Avg.PWR[mW]	0.025	0.068	0.063
Max.PWR[mW]	5.0	5.3	4.9

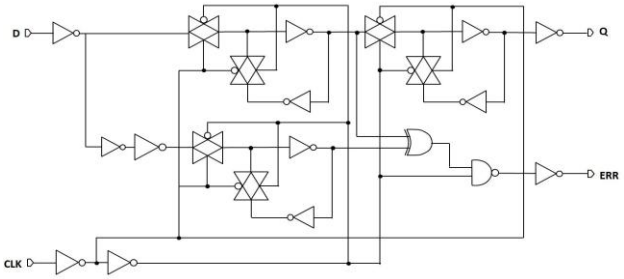


Fig. 3. Improved Canary FF circuit

transistor level circuit of CFF and cell mapping of CFF are examined. As evaluation, ROHM 0.18μm Kyoto University version standard cell library [1] is used. The cell library generated using three design corners by varying temperature and supply voltage: Max(also called worst), Typ(also called typical), and Min(also called best). The three different process corners are shown in Table I. The propagation delay of cell is longest at "Max" condition and shortest at Min "condition".

Fig.1 shows the circuit level schematic of CFF. CFF consists of main FF and shadow FF and delay buffer and XOR gate to detect a timing error. Phase synchronized clock is provided to the main and shadow FFs. Delay buffer is inserted at the input of shadow FF, hence the timing requirement of the shadow FF is severer than the main FF. Timing error is detected by comparing the outputs of two FFs. If two values are equal, system is safe and can scale down supply voltage or scale up frequency. If the two values are different, system is unsafe and about to fail to meet timing condition, so error signal is asserted to alert the error. Then the system might scale up supply voltage or scale down frequency. Fig.2 describes the conceptual timing diagrams of CFF showing timing error detection. From Fig.1, it is expected easily that it requires large cell area and the power consumption will be more than doubled since two equivalent FFs are needed in addition to the delay buffer and error detection logic. In order to reduce the area and power overhead, we try to optimize its transistor-level circuit design. When a timing error occurs, it is possible to detector the error at the timing when the data has been latched by master latch of the master and the shadow FF, hence the slave latch of shadow FF can be removed. The proposed circuit of iCFF is shown in Fig. 3.

Power analysis by HPICE simulation of three FFs:

TABLE III
# of FF at the end of timing error path

RTL	# of FF	# of FF Timing err.	Per.
MeP	3732	60	1.6%
miniMips	1967	228	11.6%

DFF,CFF and iCFF, is shown in Table.II. As expected, the average power of CFF is more than doubled than DFF. It is confirmed that the iCFF can save 7.8% of average power consumption and 7.6% of maximum power consumption of CFF.

A. Strategy of replacing with improved canary FF

If only high reliability is pursued, it is possible to replace all the FFs with iCFF, but the conventional microprocessors use thousands to tens of thousands number of FFs, it is not a practical technique when considering the area and power overhead by iCFF. Moreover, it would become serious issue how to collect timing error signal reported from all iCFFs. Hence, we have proposed that only a small number of DFF which has a small timing margin should be replaced with CFF [8]. We follow almost same approach for the selective replacement method described in [8]. Selective replacing method also used in [3] to replace DFFs of critical paths to Razor FF, however the detailed criteria and its implementation are not described in both studies. Hence in this paper we try to show our replacing method and its implementation in more detail. For the evaluation of the proposed method, we use RTL of Toshiba MeP processor [9] and miniMips processor [10].

The detail of selective replacement method is as follows: First, the RTL description is synthesized into the structured netlist by using Synopsys Design Compiler (D-2010.03-SP5). Then the delays of critical paths are analyzed. The minimum clock cycle is measured by using "Typ" condition of cell library. The minimum clock cycle is obtained by varying the clock length so that under such clock cycle no paths barely reports timing errors.

It must be confirmed that when setting the clock cycle to the measured minimum value with the process condition of cell library either with "Typ" or with "Min", no timing errors are ported. On the other hand, setting the same clock cycle length and using the library of process condition with "Max", some of the circuit paths must be ported as timing errors. This is because the "Max" process condition is the worst case of the three and the propagation delay of each cell is longest.

In Table. III, the number of DFF reported as timing error is shown. It turns out that 1.6% out of the whole DFF for the MeP and 11.6% out of the whole DFF for the miniMips are estimated to cause timing errors. Although the number of path reported as timing error is dependent on the clock cycle length, we set the smallest clock cycle under which there is no timing error is chosen when the process condition is "Typ" as described before, and select the DFFs at the end of paths reported error for replacement when using library with process condition "Max". That means, under that minimum cycle length timing errors will not occur in normal operating condition, however the possibility that timing errors will occur rise in change of environmental conditions such as sudden voltage drop, aging deterioration etc. By carrying out this strategy, it becomes possible to limit the number of DFFs which need to be replaced with CFF significantly.

B. Replacement Automation of Canary FF

The detailed procedure of replacement of DFF to iCFF is

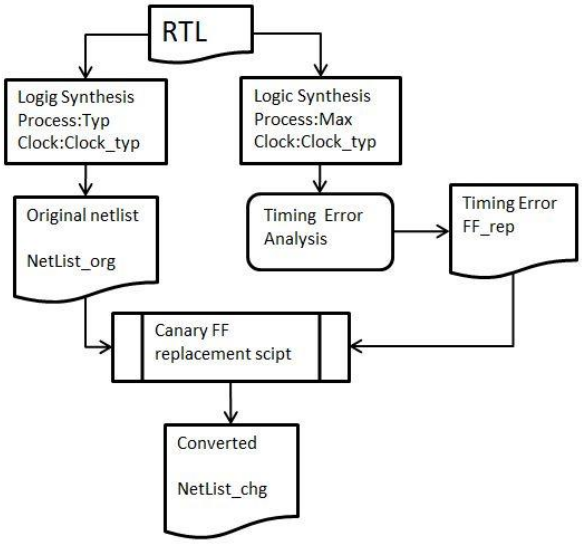


Fig. 4. Canary FF replacement procedure

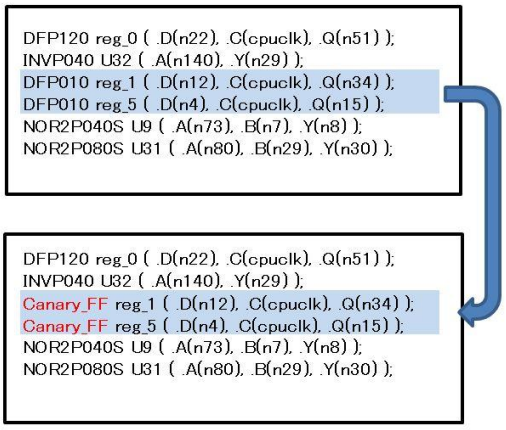


Fig. 5. Example of original netlist(top) and netlist(bottom) after some DFFs are replaced with iCFF.

shown in Fig.4. First the logic synthesis is performed by setting process condition to "Typ" and decides the smallest clock cycle length {clk_typ} from the critical path. Note

that this is an iterative process until the smallest clock cycle {clk_typ} is detected under which no timing errors are reported.

The generated netlist is saved as {Netlist_org}. Next the logic synthesis is performed again by setting process condition to "Max" and uses {clk_type} as the clock cycle. By analyzing the report from the synthesis tool, the timing error paths are extracted and the FF at the end of each path is saved in {FF_rep} for later process. After the analysis is done, the FF replacement script reads {Netlist_org} and {FF_rep} as input files and then replaces the DFF cell to iCFF cell when the instance of DFF is found in {FF_rep}. When all DFFs which might cause timing errors are replaced to iCFF, the converted netlist is outputted as {Netlist_chg}.

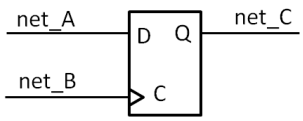
Fig.5 shows the example of original netlist and modified netlist after some DFFs are replaced with CFF. In this example, the DFF instances reg_1 and reg_5 are replaced to the iCFF since these DFFs are registered in {FF_rep}. As for the selective replacement method just proposed, we assume that iCFF is registered in the standard cell library as a custom cell and is ready for synthesis and placement & routing. However, this is not the case when the cell library is provided from third party and it is not permitted to modify or customize the library. In such cases, the iCFF has to be implemented by utilizing existing standard cells. Moreover the implementation of iCFF still underway, we propose our implementation method of CFF using existing standard cells in the next section.

IV. IMPLEMENTAION OF CANARY FF

The previous section explained the method of transforming a netlist on the assumption that we already have the cell library of iCFF. Since it is under development and the cell library is not always modifiable, we describe the implement method of CFF using existing standard cells. The same technique can be using when the iCFF is build using existing standard cells. Here the CFF is implemented using the existing standard cell and placement and routing are performed.

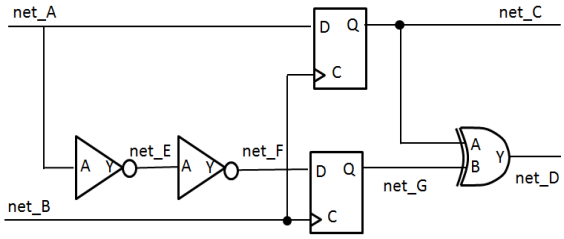
A. Cell mapping of canary FF

To help understanding following discussion, original circuit




```
ROHM18DFP010  reg_1  ( .D(net_A), .C(net_B), .Q(net_C) );
```

Fig. 6. DFF and netlist
(Before replacing to canary FF)




```
ROHM18DFP010  reg_1  ( .D(net_A), .C(net_B), .Q(net_C) );
ROHM18DFP010  UU_1  ( .D(net_F), .C(net_B), .Q(net_G) );
ROHM18INV010  UU_2  ( .A(net_A), .Y(net_E) );
ROHM18INV010  UU_3  ( .A(net_E), .Y(net_F) );
ROHM18XOR2P010  UU_4  ( .A(net_C), .B(net_G), .Y(net_D) );
```

Fig. 7. canary FF and netlist
(After replacing to canary FF)

block and netlist are shown in Fig.6. Here we consider changing reg_1 which is an instance of DFF into CFF. The instance name and the signal name are changed into the intelligible name for explanation, and as for net_a, net_b,

```
Netlist after 1st parsing

module  module_A(a, b, c);
 input    a;
 input    b;
 output   c;
 wire      net_A,   net_B,  net_C;
 @wire_decl@
   .
   DFF  reg_1 ( .D(net_A), .C(net_B), .Q(net_C));
   DFF  UU_1 ( .D(net_F, .C(net_B), .Q(net_G));
   INV  UU_2 ( .A(net_A),  .Y(net_E) );
   INV  UU_3 ( .A(net_E),  .Y(net_F) );
   XOR  UU_4 ( .A(net_C),  .B(net_G), .Y(net_D) );
   .
end module
```

```
Netlist after 2st parsing

module  module_A(a, b, c);
 input    a;
 input    b;
 output   c;
 wire      net_A,   net_B,  net_C;
 wire      net_D,   net_E,  net_F,  net_G;
   .
   DFF  reg_1 ( .D(net_A), .C(net_B), .Q(net_C));
   DFF  UU_1 ( .D(net_F, .C(net_B), .Q(net_G));
   INV  UU_2 ( .A(net_A),  .Y(net_E) );
   INV  UU_3 ( .A(net_E),  .Y(net_F) );
   XOR  UU_4 ( .A(net_C),  .B(net_G), .Y(net_D) );
   .
end module
```

Fig. 8. Netlist after 1st parsing(top) and after 2nd parsing(bottom)

net_C, it is necessary to extract the actual signal name used by each FF which needs to be replaced to CFF in the FF replacement script. Although FF replacement script to be used is almost the same as that of what was explained in Section 3, it is not simply to replace DFF to CFF, but to replace it by the cell group which constitutes the CFF. The details are stated following.

The circuit block and netlist after transformation is shown in Fig. 7. Since INV and EXOR cells are already registered into the standard cell library currently used, each gate is mapped to those cells. All the cells to be used adopt those of the minimum drive capability. UU_1, UU_2, UU_3, and UU_4 are the added instance name of the generated cells and net_D, net_E, net_F, and net_G are the added signal names. In order to avoid the duplication of name which are used by the original netlist, it is necessary to create these unique added names by combining a suitable prefix name and consecutive numbers. Since these names must be unique only within each module definition, consecutive numbers are reset when FF replacement script analyzes the syntax of the start part of a module definition, and they are incremented whenever a new name is generated. About the newly added signal name, it needs to be declared in the definition part of the module for which it is used. However, in the stage in which FF replacement script parses the module definition part, since it is not clear which signal names should be declared, it is decided to make a signal declaration by using a double parsing system. By the first parsing, while performing the processing which replaces applicable DFF to a CFF, the place holder {@wire_decl@} is described as a mark into the portion which makes a signal declaration in the head portion of module declaration as explained in Fig.8. The place holder is replaced by the declaration of the added signal names by the 2nd parsing. The added signal names are managed using the associative array referred to from the module name to which it is scheduled to be declared by the first parsing.

V. Power and Area Overhead by Canary FF

In this section, power and area overhead by CFF is investigated. Placement and routing (P&R) are performed using Synopsys IC Compiler (D-2010.03-ICC-SP2-1). Note that the layout of iCFF is not implemented yet, so the original CFF circuit is used and the netlist is generated by the method described in section IV. We use four different sets of configuration to estimate the overhead by CFF. Each configuration is described as follows,

(1) T : P&R is performed using cell library (“Typ” case) and no DFFs are replaced by CFFs
(2) TC: P&R is performed using cell library (“Typ” case) and some DFFs are replaced by CFFs using selective replacement method
(3) M : P&R is performed using cell library (“Max” case) and no DFFs are replaced by CFFs,
(4) TCA : P&R is performed using cell library (“Typ” case) and all DFFs are replaced by CFFs

Cell area and power estimates are obtained from the result of P&R by IC Compiler. Fig.9 and Fig.10 show power and area overhead for minimips and MeP processor respectively. Area is normalized based on config. (M) for both cases. For minimips, power of config. (TC) is 19.11[mW], which is increased by 26% from config. (M) and by 9.8% from config. (T). It turns out that power overhead by CFF is relatively large. In case of config. (TCA), power is estimated to 34.18[mW] which is 2.25 times larger than that of config. (M). Hence the selective replacement of CFF is deemed an effective method to decrease the power overhead by CFF. The power of config.(T) is larger than that of config.(M). This might be because in cell library “Typ”, the supply voltage is defined as 1.8V and in cell library “Max”, it is defined as 1.6V, hence the net total power consumption of config.(T) is slightly larger than that of config.(M).

On the other hand, the area of config.(TC) is decreased by 23% from config. (M) and increased by 3.6% from config. (T). This means that area overhead by CFF is much less than the cell area estimated by considering worst case “Max”

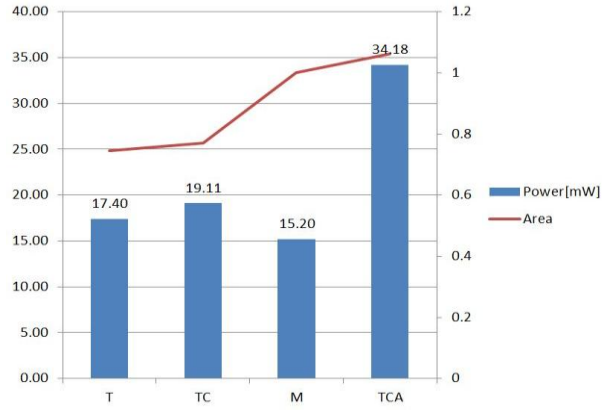


Fig. 9. miniMips
(Power and area overhead by CFF)

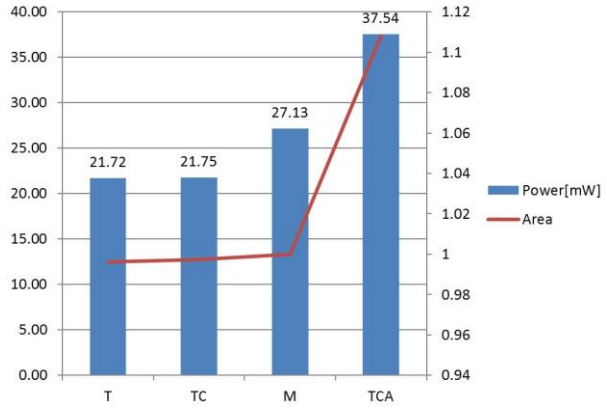


Fig. 10. MeP
(Power and area overhead by CFF)

condition. The cell area of config.(M) is relatively larger than that of config.(T). This is because in case of config.(M), gate cells with bigger drive strength, so larger cell area, are selected to compensate for longer propagation delay. Hence the area overhead by canary FF is deemed very small.

As for MeP, area overhead by CFF are very small since the chip area is mostly occupied by instruction and data caches. Instruction and data caches are implemented as hard macros using library with "Typ" condition. Actually the area of config.(TC) is decreased by 0.3% from that of config.(M) and increased by 0.1% from that of config.(T). Increase rate of area of config.(TCA) is relatively larger compared with minimips case. This might be because the number of DFFs is much larger than minimips. The power of config.(TC) is decreased by 20% from config.(M) and is increased by 0.1 % from config.(T). The power of config.(M) is much larger than config.(T). This might be since the chip area of MeP is occupied with large cache area, so the power consumed by cache is not negligible.

The difference of the two results comes from different architecture of the two microprocessors. MeP is an off the shelf commercial processor and uses state-of-art technology and contains large cache area to increase the performance (IPC); however the minimips is an open architecture processor and contains no cache. The chip area of Mep is about 7 times larger than minimips and the power consumption is about 1.25 times larger than minimips.

CFF can be used not only for microprocessor but also any sequential circuits which require timing error detection mechanism. The area and power overhead by CFF can be suppressed low by selecting DFFs for replacement by analyzing the critical paths under system timing requirement.

VI. CONCLUSION AND FUTURE WORKS

By miniaturization of semiconductor technology, the timing error caused by process variation of within-die or intra-die and aging deterioration is considered serious issue especially in deep submicron domain. The importance of the technique avoiding the defect of LSI during operation is increasing. In this paper, an automatic design method of reliable LSIs with canary FF is proposed which concentrates on typical case to ease the design margins incurred by worst case analysis. We show that by selectively replacing DFFs with canary FFs, the area and power overhead by canary FFs can be suppressed very small. The selection of DFFs is done by analyzing critical paths from worst case based on the results of typical case.

The future remaining studies regarding canary FF are as follows. First, we have to build the cell library of the improved canary FF. Since the CMOS circuit is complicated, it will be designed as double height cell. When the library is built, power and area overhead by iCFF can be measured in more detail and the comparison with CFF can be discussed. It also necessary to consider the method of collecting error signals when timing error is detected from CFFs and utilization of that signal. If there is N CFFs and when the collection of error signals is constituted from OR gates of two ports, an error signal will travel $\log_2 N$ piece of OR gate. Then the wiring delay of error signal cannot be disregarded. The error signal could be used to trigger DVS or DVFS to control the supply voltage and the clock frequency. Moreover, it is also necessary to examine the amount of delay buffer inserted at the shadow latch. When the amount of delay buffer estimates to be large excessively, timing error information occurs more than needed, and it will affect the performance. Conversely, when it is estimated too small, the possibility of overlooking timing errors becomes high and system reliability falls down. Furthermore, testing and verification of proposed method are not yet done and needs to be addressed more.

ACKNOWLEDGMENT

This study is supported in part by CREST project "Fundamental technologies for dependable VLSI system" of Japan Science and Technology Agency. The cell library used on this research was developed by Tamaru/Onodera laboratory, Kyoto University, and is released by Prof. Kobayashi of Kyoto Institute of Technology. This work is supported by VLSI Design & Education Center (VDEC), the University of Tokyo [11] in collaboration with Synopsys, Inc. and Cadence Design Systems, Inc.

REFERENCES

[1] H. Onodera, A. Hirata, T. Kitamura, K. Tamaru, "P2Lib: Process Portable Library and Its Generation System", IPSJ Journal, Vol.40, No.4, pp.1660-1669, 1999.

[2] T. Sato, Y. Kunitake, "Canary: A Variation Resilient FF to Eliminate Design Margin for Energy Reduction", IPSJ Journal, Vol.49, No.6, pp.2029-2042, 2008.

[3] D. Blaauw, S. kalaiselvan, K. Lai, et al., "RazorII:In Situ Error Detection and Correction for PVT and SER Tolerance", International Solid-State Circuits Conference, pp.400-622, 2008.

[4] D. Ernst, N. S. Kim,S. S. Das, et al., "Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation", 36th International Symposium Microarchitecture,pp.7-18,2003.

[5] J. Furuta, C. Hamanaka, K. Kobayashi, and H. Onodera, "A 65nm Bistable Cross-coupled Dual Modular Redundancy Flip-Flop capable of protecting soft errors on the C-element", IEEE Symposium on VLSI Circuits, 2010.

[6] K. Hirose, Y. Manzaw, M. Goshima, and S. Sakai, "Delay-Compensation Flip-Flops for Timing-Error Tolerant Circuit Design", International Conference on Solid State Device and Materials,pp.440-481,2007.

[7] S. Mitra, N. Seifert, M. Zhang, Q. Shi, K. S. Kim, "Robust System Design with Buit-In Soft-Error Resilience", IEEE Computer, pp. 43-52, February, 2005.

[8] Y. Kunitake,T. Sato,S. Yamaguchi,H. Yasuura, "Insertion-Point Selection of Canary FF for Timing Error Prediction", IEICE Technical Report, DC2008-42, 2008.

[9] Toshiba MeP Processor, http://www.semicon.toshiba.co.jp/

[10] miniMips processor, http://opencores.org/project,minimips

[11] VDEC, the University of Tokyo, http://www.vdec.u-tokyo

Session III: Computer Architectures

MODIFIED AVR CODING FOR TEST DATA COMPRESSION

Sruthi.P.R [1]
IInd MTech VLSI
Amrita Vishwa Vidyapeetham
Coimbatore
sruthirk@gmail.com

Dr. M.Nirmala Devi [2]
Associate Professer
Amrita Vishwa Vidyapeetham
Coimbatore
m_nirmala@cb.amrita.edu

Abstract: One of the major challenges in testing a System-on-a-Chip (SOC) is dealing with the large test data size. Several test data compression techniques have been proposed to reduce the volume of test data. This paper presents a test data compression approach, which reduces the test data volume by encoding runs of both 1's and 0's as many other codes, but here both runs share the same code word for the same run-length Further an extension to this code considering the relationship between two consecutive runs is proposed. The proposed approach is based on the use of alternating variable run-length (AVR) codes. The AVR codes can efficiently compress the data composed of both runs 0s and 1s. The decompression architecture is also presented in the paper. Experimental results were performed on ISCAS'89 benchmark circuits showed that the proposed method greatly improved the compression ratio.

Key words : Automatic test equipment (ATE), encoding, AVR codes, EFDR codes.

I. INTRODUCTION

Due to the advancements in process technology, the emphasis on placing larger and more complex devices in smaller areas is becoming more prominent. With the increase in integration density, testing has become a critical part of design process. The testing of devices after fabrication has become a major problem for both designers and test engineers. A major challenge in testing the complex designs is dealing with the enormous test data volume. In pattern storage testing, all the test vectors and test responses are stored on an external tester like automatic test equipment (ATE). But the cost of ATE grows significantly with the operating frequency, channel capacity and memory size. The amount of time taken to test a particular chip depends on the amount of data that can be transferred on to the chip and speed at which the data can be transferred known as the test data bandwidth. The test data bandwidth between the conventional testers and the chip is relatively small and hence is a bottleneck in testing a chip [1].

Several approaches were addressed in past to overcome the problems in external testing. Most of them are either a) external only approaches or b) internal only approaches. The external approaches include test data compaction techniques [2]. Though this method reduces the test data volume effectively, it does not overcome the bandwidth limitations of ATE. The internal methods are based on built-in-self test (BIST) [3]. BIST eliminates the need for external tester storage. This is very useful in performing self-test in the field when there is no access to a tester. But, these do not provide high fault coverage due to the presence of random pattern resistance faults. To increase the fault coverage in these cases, techniques such as test point insertion are required [4]. This involves modifying the functional logic which can degrade the system performance. Some other techniques modify the test pattern generator, but this tends to result in large silicon area.

A solution to the ATE problem that does not introduce any performance penalties is test data compression which is a test resource partitioning variant. This arises as a possible solution to reducing the speed, channel capacity and memory requirements of ATE. By introducing an on-chip decompressor it reduces the load on the ATE and therefore simplifies the channel capacity and speed requirements. Test data compression compresses test data losslessly and hence preserves the fault coverage unlike other techniques [5].

The three test data compression environment (TDCE) parameters are area overhead, test application time and compression ratio. Satisfying all these parameters simultaneously is found to be a difficult task. The existing approaches trade off some of these parameters against others. In this paper, a new test data compression scheme is formulated. The compression scheme is based on code based compression which achieves sufficient compression ratio leading to a reduction in chip area and hence an overall reduction in test cost [6].

The compression scheme reduces the test data volume by encoding runs of both 1's and 0's as many other

codes, but here both runs share the same code word for the same run-length Further an extension to this code considering the relationship between two consecutive runs is proposed. The proposed approach is based on the use of alternating variable run-length (AVR) codes. The AVR codes can efficiently compress the data composed of both runs 0s and 1s.

The rest of the paper is organized as follows. We present an analysis on the existing run-length codes, the EFDR (extended frequency directed) code and the AVR code in Section 2. In Section 3, the modified AVR test architecture for data compression procedure, and the decompression architecture is presented. Experimental results for the large ISCAS'89 benchmark circuits in Section 4. Section5 summarizes the paper.

II. PRELIMINARY

A. EFDR CODES

The EFDR code is a data compression code that maps variable length runs of 0s or variable length runs of 1s to a variable length code word. The don't-cares in test vectors are mapped to either 0 or 1 before coding. For example consider a run of seven 1s (0s). The run belongs to group A3 and it is mapped to the code word 1110000 (0110000). A detailed discussion for EFDR code is given in [7]. These codes were found to have better compression ratio than frequency directed (FDR) codes though an extra bit is used to indicate the type of run. Fig.1 shows an example of encoding using FDR and EFDR codes.

Illustration:

T_D = {0000001111111111110111000111111100000001} (40bits)

T_{FDR} = {110000 00 00 00 00 00 00 00 00 00 00 00 00 1001 110001 00 00 00 00 1001} (52 bits)

T_{EFDR}= {01011 1110101 11000 001 11011 01011} (30 bits)

Fig.1 An example showing encoding using FDR and EFDR codes

In Fig.1, T_D is the input data stream and T_E is the encoded data. The input data contains both runs of 0s and 1s. In FDR coding it can be seen that the size of the encoded test set is larger than the size of uncompressed test set. Hence, FDR codes are inefficient for data streams that are composed of both 0s and 1s. It can be observed that the compression ratio increases by using EFDR codes. During the analysis it is observed that the compression ratio of s35932 has increased from 19.359% to 57.6530% after using EFDR code based compression. It can be noted that there exists a scope of reducing the code-word length in these codes which will further increase the compression ratio. This was exploited in Alternating Variable Run-length codes (AVR codes).

B. AVR CODES

AVR code is a variable-to-variable length code. This code helps in reducing the code word length by including two parts – the group prefix and the tail. Fig.2 shows an example of encoding using AVR codes for the same input sequence which was earlier used for EFDR codes. It can be seen that the compression ratio has improved further as the encoded bits has reduced further to 26 bits.

Illustration:

T_D ={0000001 1111111111110 1110 001 1111110 0000001} (40bits)

T_{AVR}= {00100 11011 100 011 00101 00101} (26 bits)

α= 0 α=1 α=1 α=0 α=1 α=0

Fig.2 An example showing encoding using AVR codes

A detailed discussion for AVR codes is given in [8]. The AVR code consists of two parts – the group prefix and the tail. The prefix identifies the group in which the run-length lies and the tail identifies the member within the group. The presence of these two parts distinguishes AVR code from other run-length codes. Table I illustrates the encoding for the AVR code.

TABLE.I EXAMPLE OF ALTERNATING VARIABLE RUN-LENGTH (AVR) CODE.

Group	Run Length	Group Prefix	Tail	Code word
A1	1	01	0	010
	2		1	011
	3	10	0	100
	4		1	101
A2	5		00	00100
	6		01	00101
	7	001	10	00110
	8		11	00111
	9		00	11000
	10		01	11001
	11	110	10	11010
	12		11	11011
A3	13	0001	000	0001000
	:		:	:
	28	1110	111	1110111

A_1, A_2, A_3….A_K represents the different groups, where k is the longest run-length L as in [8]. Group A_j is calculated using (1).

$$j = \log \lceil \log_2(L+4) - 2 \rceil \quad (1)$$

Existing AVR codes can be further improved if relationship between two consecutive runs can be incorporated. It can be observed that if two consecutive runs have the same run-length, then the second run can be encoded using a small code word. This is done in this paper, where a code that efficiently compresses both runs of 0s and 1s after considering a relationship between two consecutive runs is proposed. This code is discussed in detail in the next section.

III. MODIFIED AVR CODE

As stated earlier, modified AVR codes exploits the relationship between two consecutive runs. It can be observed that if two consecutive runs have the same run-length, then the second run can be encoded using a small code word. If the second run is runs of 0 then it is encoded as 000 else encoded as 101. If there are more number of consecutive runs having the same run then a combination of the previous codes can be used. Fig. 3 shows an example of encoding using modified AVR codes. Table II shows an analysis of modified AVR codes. In this section we describe in detail the compression procedure and the decompression architecture.

A. ENCODING PHASE

The overall flow of the encoding phase is depicted using a flow chart in Fig. 4. After obtaining the fully specified test set, the encoding stage is performed. For this, the run length of the original bits is to be counted from the beginning. 0 run is defined as number of 0s followed by 1 and 1 run is defined as number of 1s followed by 0. Initially the run-length, cnt is assumed to be 0. For each occurrence of the same bit as the previous, this count, cnt is incremented until the occurrence of the opposite bit. For example, if the first occurring bit in the sequence is 0 (1), the next bit is checked then, if both the bits are found the same 0 (1), the count is incremented by 1. This is continued until the bit 1 (0) is seen.

After this the next consecutive run-length is checked. If both the run-lengths are same (cnt = pre_cnt) then the run type is noted, if it is runs of 0 then it is coded using 000 else it is coded as 101 (Type 2). In situations where the run-lengths are not same, then a normal AVR encoding (Type 1) is done. This can be explained using an example.

Illustration:

T = {0xx0xx1 111xxxxxxxxx0 11x0 001 xxxxx10 xx0xxx1} (40 bits)

T_D ={0000001 1111111111110 1110 001 **1111110 0000001**} (40bits)

T_{MAVR} = {00100 11011 100 011 **00101 000} (**24 bits)

α= 0 α=1 α=1 α=0 α=1 α=0

Fig.3 Example showing the modified AVR encoding technique

It can be observed from the example that the last two runs of the original set T_D have the same run-length and hence they can be encoded using the Type 2 coding technique. Since the second run-length is runs of 0 it is encoded as 000. The other run-lengths are not found to have any similarity with the previous bits and hence they are encoded using Type 1 coding. It can be observed that the amount of compression achieved has been improved when compared to the other existing techniques.

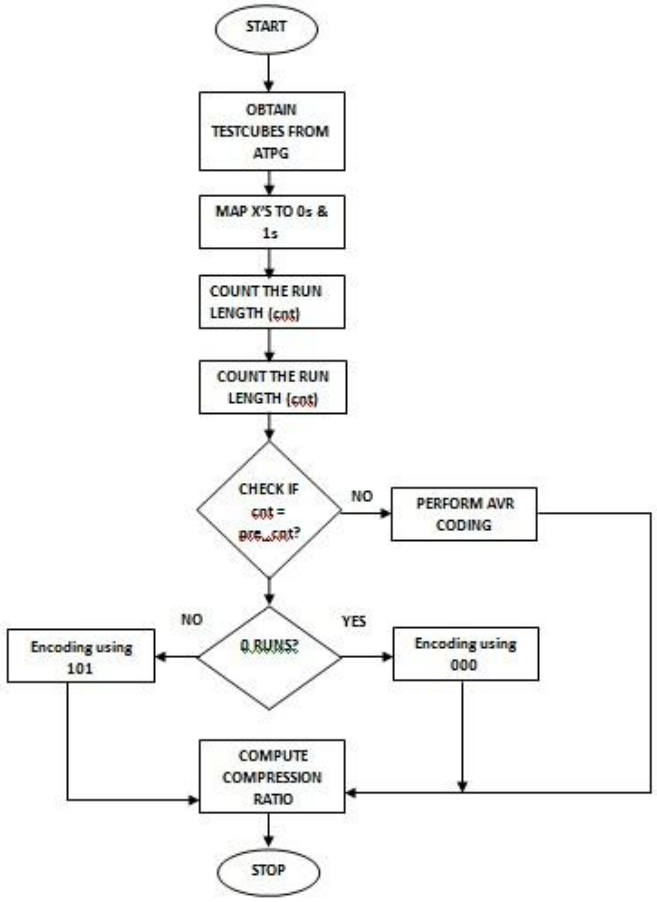


Fig.4 Flowchart showing the encoding procedure

B. DECOMPRESSING PHASE

This is carried out on-chip with the help of an on-chip decoder. It decompresses the encoded test set T_E and produces the primary set T_D. The decoder architecture for the modified AVR code is shown in Fig. 5. The decompressor architecture is simple and it is independent of the pre-computed test set and the CUT. This decompressor architecture is slightly modified from conventional AVR coding [8] after considering the relationship between two consecutive runs. For this, an extra counter in included in the architecture to keep an account of the previous run-length.

If the previous run-length was found equal to the present run-length, then the output is directly fed as 000 or 101 according to the run type. Though the architecture was modified with the inclusion of an extra counter, this does not add much to the area overhead and hence is acceptable. If L_{max} is the longest run of 0s or 1s, then k is given by (2).

$$k = \lceil \log_2 (L_{max} + 4) - 2 \rceil \quad (2)$$

The decoder consists of a (k+1)-bit counter, a $\log_2$ (k+1)-bit counter a finite-state machine (FSM), a T flip-flop and an exclusive OR gate.

- *bit_in* is the input to the FSM and *en* is the enable signal used to control the input encoded data when the decoder was ready.
- *shift* signal is used to control the prefix and the tail of the code word to the (k+1)-bit counter. *dec1* and *rs1* are the decrement and reset pins respectively. *rs1* indicates the reset state of the counter.
- A $\log_2$ (k+1)-bit counter was used to count the length of the prefix and the tail which helps in identifying which group it belongs to. *inc* and *dec2* are used to increment and decrement the counter respectively. *rs2* is another signal which indicates whether the counter has finished counting.
- A T-flipflop is included which is controlled by the *out* signal of the FSM. Signal *v* is indicated when the output was valid. This is used to control the scan clock signal *scan_clk.*

The operation of the decoder can be explained as below:

Step1: In the initial state, the T flip-flop was reset and *v* was reset to 0. Signal *en* became high to indicate the readiness of the decoder to receive *bit_in*.

Step 2: The FSM fed the prefix to the (k+1)-bit counter. The end of the prefix is indicated by a separator 0 or 1 according to the type of the codeword. The signals *en, shift,* and *inc* were kept high until the separator was received. The signal *out* gives out 0 until the *rs1* signal goes high. This occurs when the prefix is fully read into the (k+1)-bit counter.

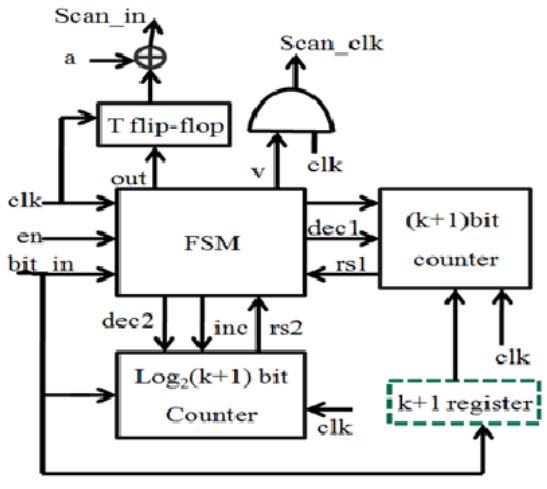


Fig.5 The decoder architecture of modified AVR code

Step3: The 0s given out by *out* signal is xor-ed with the α bit to produce the first part of the original set. Now, the *en* signal goes high and the (k+1) bit counter receives the bits until the *rs2* signal of the $\log_2$ (k+1)-bit counter goes high. This is when the value of *dec2* becomes 1 less than the prefix count.

The FSM for AVR code has 8 states. The S0-S1-S2-S3 process was related to the prefix decoding with "0….01" and S0-S6-S7-S3 was related to the prefix decoding of the codeword with prefix "1…10". The states S0-S1-S3 and S0-S6-S3 processes were related to the prefix decoding with "01" and "10", respectively. The S4-S5 process was related to the tail decoding. Fig. 6 shows the state diagram of the FSM.

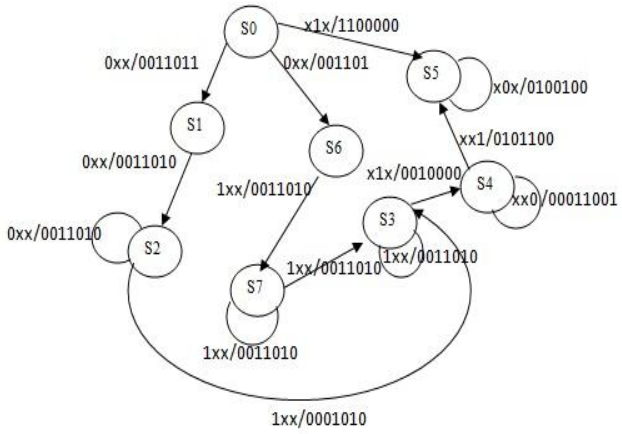


Fig.6 State diagram of the FSM

IV. POWER CONSUMPTION ANALYSIS

The impact of test set encoding on power consumption during scan testing is examined. For CMOS circuits, power consumption can be classified as either static or dynamic. Static power consumption, which is caused by leakage current is negligible and therefore ignored. Dynamic power constitutes the predominant fraction of CMOS power consumption.

For scan vectors, the dynamic power consumption during testing depends on the number of transitions that occur in the scan chain as well as on the number of circuit elements that switch during the scan in and scan out operations. Power estimation models based on the switching activity of the circuits have been presented in [10][11]. Weighted transition metric (WTM) introduced in [11] is used to estimate the power consumption during scan vectors.

The WTM models the fact that the scan in power for a given vector depends not only on the number of transitions in it but also on their relative positions. Consider a scan chain of length l and a scan vector $t_j = t_{j,1}, t_{j,2}, \ldots t_{j,l}$,with $t_{j,1}$ scanned in before $t_{j,2}$ and so on. The weighted transition metric for t_j, denoted WTM_j, is given by (3).

$$WTM = \sum_{i=1}^{l-1} (l-i) * (t_{j,i} \oplus t_{j,i+1}) \quad (3)$$

If T_D contains n vectors $t_1, t_2 \ldots t_n$ then the average scan in power P_{avg} and peak scan in power P_{peak} are estimated using

(4) and (5) respectively.

$$P_{avg} = \frac{\sum_{j=1}^{n}\sum_{i=1}^{l-1}(l-i)*(t_{j,i} \oplus t_{j,i+1})}{n} \quad (4)$$

$$P_{peak} = \max_{j\in\{1,2,3,..n\}}\{\sum_{i=1}^{l-1}(l-i)*(t_{j,i} \oplus t_{j,i+1})\} \quad (5)$$

V. EXPERIMENTAL RESULTS

To validate the efficiency of the proposed method, experiments were performed on ISCAS'89 benchmark circuits. The test sets used in these set of experiments were obtained using MinTest ATPG. The experiments were performed on Intel ® core i5 2.30 GHz workstation with 4.00 GB RAM. FDR [9], EFDR [7], AVR [8] and the proposed modified AVR were implemented in C. First the proposed compression method is analyzed from the point of view of compression ratio. It should be noted that all the proposed schemes were applied on the same test set after performing mapping as proposed.

Fig.7 shows an analysis of the number of runs in a test set. The number of 1s in a test set is as equally important as the number of 0s in the test set. This can be clearly visualized from the figure**.** In circuits like s35932, the number of runs of 1s exceeds the number of runs of 0s and this is the main reason for a low compression ratio obtained in these circuits when only 1s coding or 0s coding alone was done.

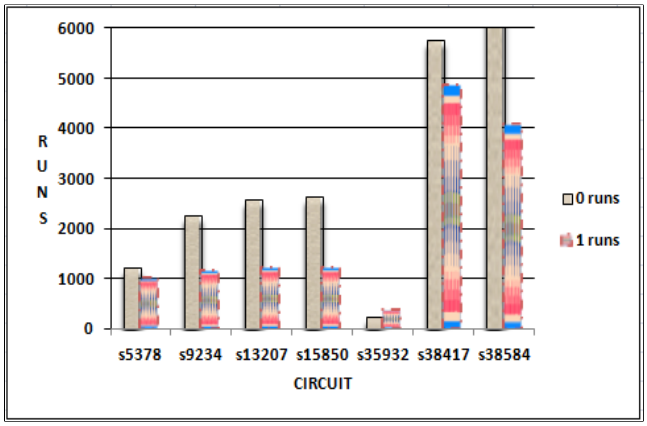


Fig. 7 Distribution of the runs both 0s and 1s in various sequential circuits

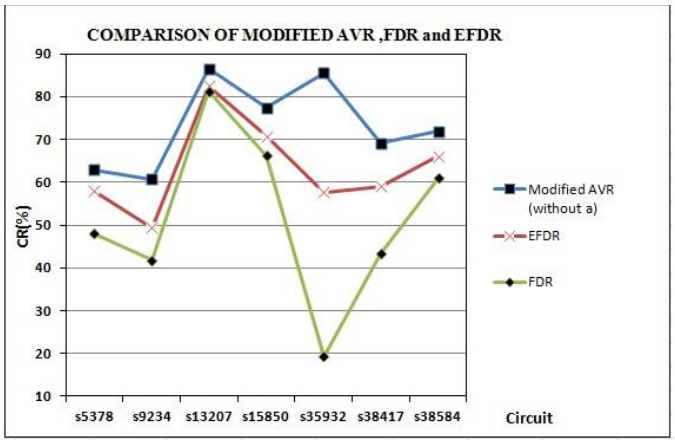


Fig.8 Comparison between Modified AVR coding and other existing code based schemes

TABLE. II. COMPARISON BETWEEN AVR CODES AND MODIFIED AVR CODES.

Circuit	Size of (T_D)	AVR(A) (CR%)	Modified AVR (A) CR(%)	AVR(B) (CR%)	Modified AVR (B) CR(%)
s5378	23754	61.94325	62.93062	55.0307	56.49575
s9234	35479	59.36692	60.63751	52.34364	53.25967
s13207	165200	85.61481	86.44055	83.91014	84.70418
s15850	76986	75.9469	77.43804	72.82694	73.3335
s35932	28208	83.90585	**85.52589**	82.3177	**84.59389**
s38417	164739	69.06491	69.15181	64.42979	65.43867
s38584	199106	71.77032	71.90542	67.98188	68.58006

Table. II shows an analysis of modified AVR codes. The decompressor architecture for the modified AVR code contains an FSM, a (k+1)-bit counter, $\log_2(k+1)$ bit register, a T-flip-flop, and an extra register. The FSM was coded in verilog HDL using Model Sim 6.2c and was synthesized in Quartus II.

It can be clearly observed that there exists an increase in compression ratio after considering the relationship between two consecutive runs. AVR (A) represents the AVR encoding with α parameter is not included and AVR (B) the parameter α is included. The compression ratio obtained after including α is found less than the coding where α is not included. But, the compression ratio in both the cases is more than the existing encoding techniques.

TABLE III COMPARISON AVERAGE POWER (mW)

Circuit	Average power (mW)			
	FDR	**EFDR**	**AVR**	**Proposed**
s5378	3185	2987	1565	1599
s9234	5329	5321	1348	**956**
s13207	12173	8026	6703	**6381**
s15850	17446	13611	8931	8969
s38417	184743	93225	72341	**72294**
s38584	203104	133420	61265	61469

TABLE IV COMPARISON PEAK POWER (mW)

Circuit	Peak power (mW)			
	FDR	**EFDR**	**AVR**	**Proposed**
s5378	7842	7867	7864	7852
s9234	10082	8543	7682	**7694**
s13207	84342	64875	76958	**64341**
s15850	65368	64450	37753	46082
s38417	45897	124374	24671	**24384**
s38584	453352	674923	398934	**381158**

Fig.8 shows a comparison between the modified AVR code with other compression schemes. The achieved compression ratio is maximum for modified AVR codes. A comparison of the peak power and average power obtained using the different coding schemes is shown in Table III and Table IV. It can be observed that there exists sufficient amount of reduction in some circuits like s13207,s38584etc.

VI FUTURE WORK

The timing calculations for the proposed scheme is to be analysed. The proposed scheme will be feasible only if it consumes less power and also does not add to the testing time. The power calculations for this scheme is found to be similar to or has only a marginal increase for some circuits like s13207 etc. Therefore techniques such as reordering of test vectors can be applied to reduce the power consumption further. These two parameters of the test data compression environment (TDCE) has to be analysed in detail.

VII CONCLUSIONS

An efficient code based compression scheme is presented which is found to have a higher compression ratio than any other existing schemes. Decoder architecture for the modified scheme has been proposed. Experiments were performed on all sequential circuits - ISCAS'89 benchmark circuits and are found to have an increase in compression ratio.

ACKNOWLEDGEMENTS

We thank Prof. Nur A Touba of Duke university for providing MinTest ATPG vectors for performing the experiments.

REFERENCES

[1] N. Tauba, "Survey of test vector compression techniques", *IEEE Transaction Design & Test of Computers*, pp. 294–303, 2006.

[2] Hamzaoglu and J. H. Patel, "Test set compaction algorithms for combinational circuits", in *Proceedings International Conference on Computer-Aided Design (ICCAD)*, pp. 283–289, Nov. 1998.

[3] D. Dsa and N. A. Touba, "Reducing Test Data Volume Using External/LBIST Hybird Test Patterns", in *Proc. Test International Conference*, pp. 115 -122, 2000.

[4] A. Chandra and K. Chakrabarty, "A Unified Approach to Reduce SOC Test Data Volume, Scan Power and Testing Time", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 352-363, March 2003.

[5] V. Iyengar, K. Chakrabarty, and B. Murray, "Deterministic built-in pattern generation for sequential circuits", *Journal of Electronic Testing: Theory and Applications*, vol. 15, pp. 97–114, August/ October 1999.

[6] A. Chandra and K. Chakrabarty, "Test resource partitioning for SOCs," *IEEE Design & Test of Computers*, vol. 18, pp. 80-91, September-October 2001.

[7] A. El-Maleh , "Test data compression for system-on-a chip using extended frequency-directed run-length code," *IET Comput. Digital Tech.2(3),* pp.155–163, 2008.

[8] Bo Ye, Q.Zhao, D.Zhou, X.Wang and M. Luo, "Test data compression using alternating variable run-length code", *Integration, the VLSI journal*, pp.167-175, Nov. 2010.

[9] A. Chandra and K. Chakrabarty, "Test Data Compression and Test Resource Partitioning for System-on-a-Chip Using Frequency-Directed Run- Length (FDR) Codes," IEEE Trans. Computers, vol. 52, no. 8, Aug. 2003, pp. 1076-1088.

[10] [10] H. Fang, C. Tong, and X. Cheng, "RunBasedReordering: a novel approach for test data compression and scan power," in *Proceedings of the Conference on Asia South Pacific Design Automation (ASP-DAC '07)*, Yokohama, Japan, January 2007.

[11] U. S. Mehta, K. S. Dasgupta, and N. M. Devashrayee," *Research Article* Weighted Transition Based Reordering, Columnwise Bit Filling, and Difference Vector: A Power-Aware Test Data Compression Method," Hindawi Publishing Corporation VLSI Design, Volume 2011, Article ID 756561, 8 pages.

[12] L Zhang J.S Kuang and Z-Q. You, "Test Data Compression Using Selective Sparse Storage*," J ElectronTest*,June2011.

[13] Nourani M, Tehranipour M," RL-Huffman encoding for test compression and power reduction in scan application," *ACM Trans Design Autom. Electron Syst* , vol.10 pp.91–115, 2005.

[14] Jas A, Gosh-Dastidar J, Ng M, Touba N," An efficient test vector compression scheme using selective Huffman coding," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst* (22)6:797–806, 2003 .

A Study on Performance Impact of Network Delay in Chip Multi Processors

Monobrata Debnath , Ankil Patel , Byeong Kil Lee
Department of Electrical and Computer Engineering
The University of Texas at San Antonio
San Antonio, Texas
mhl494@my.utsa.edu , ankilpatel@hotmail.com , byeong.lee@utsa.edu

***Abstract*—Next generation CMPs are expected of having hundreds of cores per chip. As the number of cores increases, performance dependency of CMPs also varies. Cache Miss is always being crucial on the overall performance of both multi core and unicore systems. But with the growing number of cores per chip, future CMPs will face new performance challenges. One of such challenges is on chip communication. Network on chip or NoC is an emerging and efficient way of solving on chip communication for future CMPs. But its network delay and power consumption remains crucial so far. In this paper, we observe that the domination on overall performance of CMPs will gradually shift from cache miss to NoC delay with the growing number of cores.**

I. INTRODUCTION

With the growing transistor count per chip, computer architects have been given the opportunity to explore new design paradigm chip multiprocessing (CMP). Performance of CMPs in terms of speed and power consumption is also satisfactory and this has lead computing industry to move towards increasing cores for future computing devices The Network on Chip (NoC) has gained enormous popularity in the recent years and has shown the potential to be an efficient alternative for CMPs [1][2][3]. Despite of NoC being very promising, current design aspects of NOC still prevents CMPs from reaching its maximum potential. The recent study [3][4][5] encourages multi module last level memory systems for large number of core CMPs (Figure1). Unfortunately, the multi module memory system also uplifts traffic congestion in the NOC significantly. As the number of cores expected to go higher and higher in the near future, NOC will be emerging as a major performance bottleneck in the multi module memory architecture. Kim et. al performed in [6] address this issue of network congestion in CMPs. But researchers miss the fact that there exists a correlation between the network latency and memory controller (or the cache coherency protocol) when the number of cores goes high [7][8][11]. This correlation becomes stronger as the number of cores increases. In this study, we observe that the lower order performance trend strongly follows L1 cache miss and loosely follows L2 miss and network latency. There has been very little work done to propose any integrated and mutually supporting solution to control the network congestion treating memory coherency protocols and network parameters inter-reliant of each other. We also notice that the network latency becomes more dominant on the overall performance of the CMP, as the number of cores increases. We have simulated 4, 8 and 16 core CMPs with the state of the art MOESI token and directory based protocols and three different network topologies namely point-to-point, torus2D and hierarchical switches.

The current state of the art widely used coherency protocols are simply migrated from lower order CMPs to higher order CMPs. The nature and the density of network congestion vary from one CMP to another if number of cores increases. The coherency protocol adopted for higher cores may show almost no issue in terms of scalability but fails to control the network congestion and hence results in performance degradation and excess power consumption. This happens due to the fact that at any point of time the cache controller receives no information about network congestion. In this paper, we see that the overall performance would be dominated by NoC delay, if coherency protocols are adopted without necessary modification from a lower order CMP. This may dilute the effects of the advance features of each core and state of the art memory architectures.

We proceed with our discussion by presenting the coherency protocol chosen for this study in section II, network topologies in section III, simulation environment and methodology in section IV. Section V describes the simulation results, and we conclude with section VI.

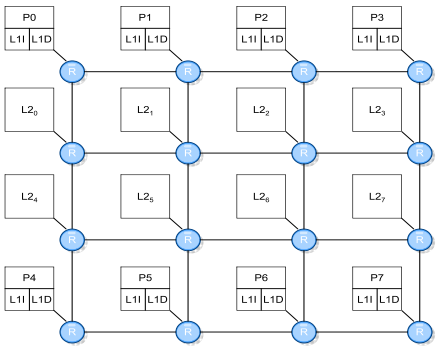
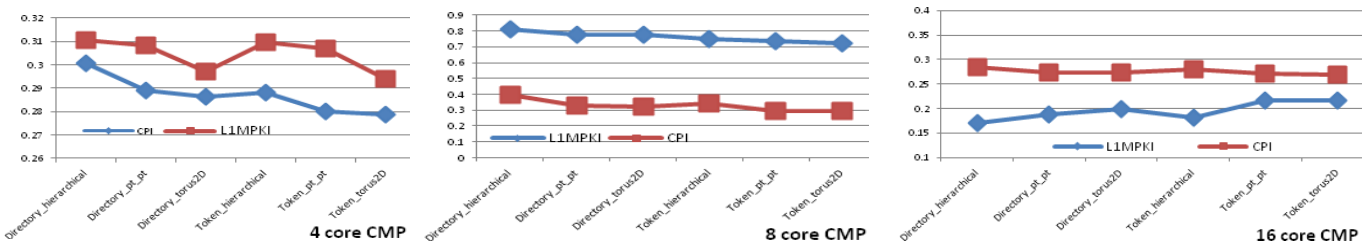


Figure1: Multi module memory Arrangement

II. CACHE COHERENCY PROTOCOL

MOESI Coherency Protocol

We choose MOESI [9] protocol for our experiment. MOESI Coherency Protocol was proposed by combining MESI [10] and MOSI protocols. We further divide MOESI as



(a) (b) (c)

Figure 2: Correlation between CPI and L1 miss in (a) 4-core (b) 8-core (c) 16-core CMPs

directory based and token based. MOESI directory based protocols maintains a directory at the home node. Directory protocol sends all requests to its home node, which adds an indirection to critical path of cache-to-cache misses. In addition to that, it also escalates burden on NOC by sending redundant requests. On the other hand, token based coherency protocol [14] provides low-latency interconnection without indirection. But cache to cache miss prevention is not as efficient as directory based protocols. Wang H. et. al [13] have combined the benefits of directory based and token based protocols by using Sharing Relation Cache .This approach finds an intermediate way to retain the benefits of both directory and token based protocols.

III. Network Topologies

The on-chip network topology often adopted from off chip networks determines the physical layout and connections between nodes and channels in the network, the physical layout and connections between nodes and channels in the network [14]. A topology determines the number of hops (or routers) a message must traverse as well as the interconnect lengths between hops, thus influencing network latency significantly. In our experiment, we have considered three different topologies: (1) Hierarchical switch, (2) Torus 2D and (3) Point to Point. These topologies consist of a set of system components; each one being connected directly or indirectly to a processing element. A processing element could be a processor, cache or a cache controller. Each component usually has a switch (or router), which handles message communication among components. Each switch/router has direct connections to its neighbor one. If the connection between the routers to the processing element through a direct connection is called as direct topology else it is known as an indirect topology. Point-to-point and torus2D are direct topologies where as hierarchical switch is an example of indirect topology.

Network topologies and cache coherency protocol

The network topologies and cache coherency protocols are considered as individual independent entity. P. Foglia et. al [12] have shown that if cache coherency protocols and network topologies combined wisely has positive impact on the overall L1 miss latency and cache misses and hence thereby on overall performance. In our study, we re-explore the characteristics of NoC topology and cache coherency protocol as a single entity and its impact on the overall performance when the number of cores increases.

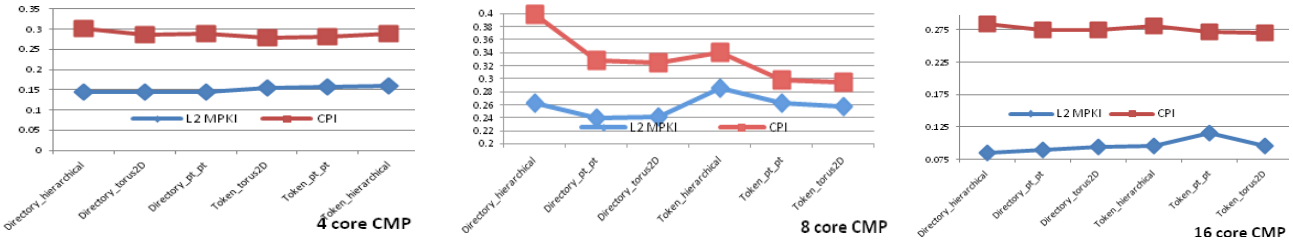
IV. Simulation Environment and Methodology

We performed the full-system simulation using Simics [15]. We simulated 4, 8 and 16 CPU UltraSparc III plus CMP system with Sun Solaris 10 operating System, each CPU using in-order issue, running at 75 MHz .We also used GEMS [15] in order to simulate the memory hierarchy and coherence protocols. For NoC measurement we applied Garnet [16] and Orion [17] for network power estimations. We conduct our simulation against complete set of Parsec 2.1 multithreaded benchmark suite [19] with native inputs. One billion instructions are executed with fast forwarding first 1 million instructions. The cache system is two levels with L1I and L1D as private caches to each core and L2 as a shared resourse. The cache architecture is a non uniform (NUCA). Size of L1 cache is 64 KB 4-way set associative with block size 64 Byte. L2 is 16MB 4-way set associative with number of modules kept equals to number of processors. The cache replacement policy is Pseudo-LRU [21]. The router is a 5 stage pipelined with X-Y routing algorithm and with 5 virtual channels and flit size is of 16 byte.

V. Simulation Result

We have simulated at least 1 billion instructions for each of Parsec benchmark suite with three different topologies and two different coherency protocols. We choose Cycles Per Instruction (CPI) as the reference performance indicator. Also, L1, L2 misses per thousand (Kilo) instructions (MPKI), network latency and network power consumption as other indicators. Arithmetic Mean of L1 MPKI, L2 MPKI, network latency and CPI of all the thirteen Parsec benchmarks are used throughout this section. In addition to arithmetic mean we used logarithmic values (base 10) of network latency in the plots throughout this paper.

As shown in Figure 2, it is clear that L1 miss per thousand instructions is correlated with CPI in 4-core CMP. On the other hand, when the number of cores is doubled from 4 cores to 8 cores, CPI is now loosely following L1 miss trends. In case of MOESI Token and torus2D combination, this observed behavior is completely opposite. For an 8 core CMP, L1 miss has significantly less impact on CPI than that of 4 cores. But when the number of cores is quadrupled, the correlation is completely lost and behaves in a way opposite to that of 4 core CMPs.



(a) (b) (c)

Figure 3: Correlation between CPI and L2 miss in (a) 4-core (b) 8-core (c) 16-core CMPs

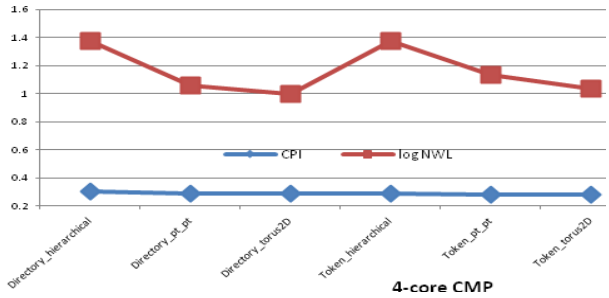


Figure 4: CPI and N/W latency for 4 core

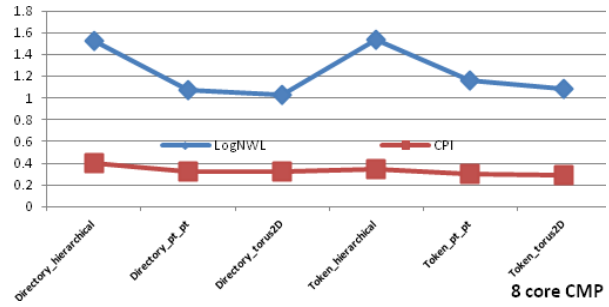


Figure 5: CPI and N/W latency for 8 core

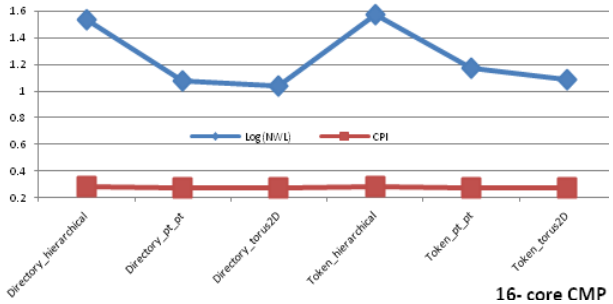


Figure 6: CPI and N/W latency for 16 core

However, cache misses cannot improve performance hence it is well understood that with increasing number of cores the performance bottleneck is no longer L1 cache miss. If L2 miss is the performance indicator when order of the CMP is more, then token coherency with point to point (token_pt_pt) combination would have been the slowest among all the six combinations. But as shown in Figure 3(c), it is clear that token pt_pt has the second least CPI Since the token based protocols has less network latency compared to directory based protocols even though the cache miss in token based protocol has more. So it is clear that L2 miss is not the only one where bottleneck exists. This indicates for a third possibility: network delay in higher order CMPs. Cache misses in a 16 core CMPs have very less significance on the overall performance than the network delay. It would be legitimate to assume that with the growing number of cores the network delay will be the most crucial. The network latency and CPI is shown in Figure 4.

To further investigate the correlation between CPI and network latency we increase number of cores from 4 to 8 (Figure 5). The CPI is following average network latency in way which is more profound than that of 4 cores. For 16 cores (Figure 6), CPI is following network latency. In 8 cores CMP CPI follows neither network latency nor L1 miss. L2 misses impaction on CPI is more in case of an 8 core CMP. For 4 core it is L1 miss rate and for 8 cores it is L2 miss which dominates the overall performance and network latency dominates the overall performance for 16 core CMP.

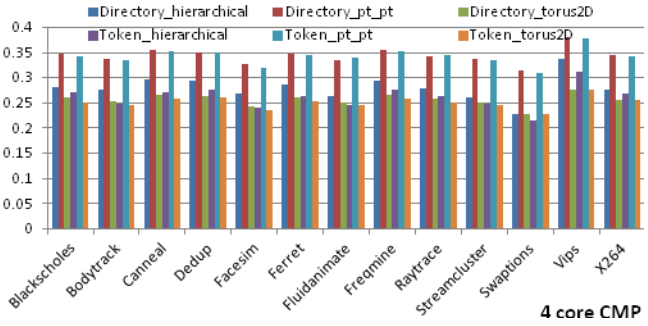


Figure 7: Total Network Power (W) for 4 core CMP

Power Consumption

Early stage estimation of NoC power has become crucially important in chip multiprocessing. We calculate total network power as the summation of router and link power. Figure 7, 8 and 9 represents the total network power for 4, 8 and 16 core CMPs respectively. Point to point topology consumes more power than any other network topology in all three CMPs. The *Vips* consumes highest power in all 6 combinations for 4 and has highest number of L1, L2 misses and network delay in 16

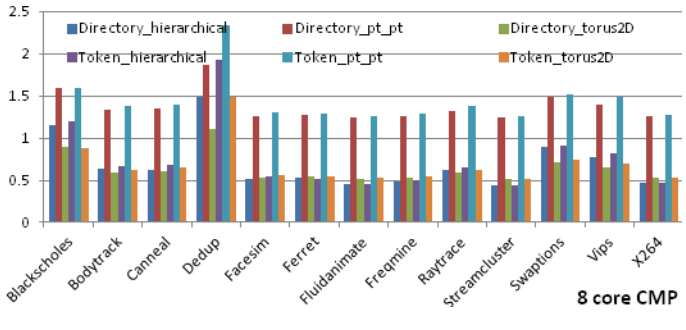


Figure 8: Total Network Power (W) for 8 core CMP

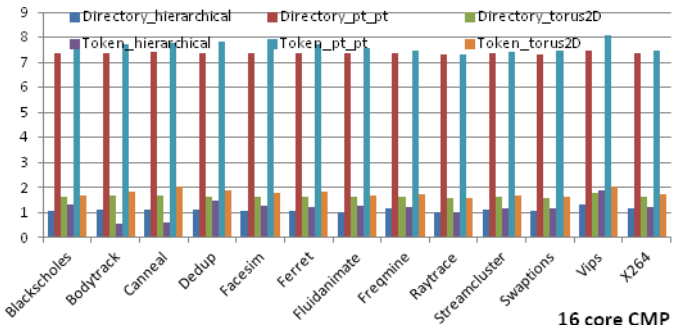


Figure 9: Total Network Power (W) for 16 core CMP

core CMPs. The *Vip*s is an image processing application and respective CMPs. But in the case of 8core, *Dedup*, a kernel application, consumes more power. It also has the highest L1, L2 miss and network delay. We see from the power statistics that the network delay and cache misses are the primary influences of network power consumption.

VI. CONCLUSION

Performance of CMPs depends on the memory and its accessing capabilities. With large number of cores per chip future CMP's dependence on the on chip communication would be very critical. With growing number of cores the critical factors to its performance changes from L1 miss to network delay. We simulated 4, 8 and 16 core CMPs to investigate the performance with widely used directory and token based MOESI coherency protocol combined with various network topologies.

Without a symbiotic relation being established between cache coherency and NoC parameters, future CMPs would not be able to handle the network congestion efficiently and hence network delay in future CMPs would be very crucial on the overall performance. We would like to investigate more on the network delay with more than 16 cores and improve the coherency protocols so that it considers the network congestion. Enhancing MOESI protocol with network congestion handling capability will be the future goal in the future.

REFERENCES

[1] S. Bell, *et al*., "TILE64 - Processor: A 64-Core SoC with Mesh Interco nnect", Proc. ISSCC 2008.

[2] Intel, Co., "Single-chip Cloud Computer", available http://techresearch.intel.com/articles/Tera-Scale/1826.htm.

[3] J. Bautista, "Tera-scale Computing and Interconnect Challenges– 3D Stacking Considerations", Tutorial, ISCA, 2008.

[4] W. Kwon, *et al*., "A Practical Approach of Memory Access Parallelization to Exploit Multiple Off-chip DDR Memories",Proc. DAC, 2008.

[5] E. Aho, *et al*., "A Case for Multi-channel Memories in Video Recording", Proc. DATE, 2009.

[6] J. Kim, *et al*., "A Low Latency Router Supporting Adaptivity for On-Chip Interconnects", Proc. DAC, 2005.

[7] A. Singh, *et al*., "GOAL: A Load-Balanced Adaptive Routing Algorithm for Torus Networks", Proc. ISCA, 2003.

[8] A. Singh, *et al*., "Globally Adaptive Load-Balanced Routing on Tori", IEEE Computer Architecture Letters, 3(2):2, 2004.

[9] D. Kim, et al "A Network Congestion-Aware Memory Controller" 2010 Fourth ACM/IEEE International Symposium on Networks-on-Chip.

[10] P. Sweazey and A. J. Smith. A class of compatible cache consistency protocols and their support by the IEEE futurebus. In ISCA '86: Proceedings of the 13th annual international symposium on Computer architecture, pages 414{423, Los Alamitos, CA, USA, 1986. IEEE Computer Society Press.

[11] Z. Chishti, M. Powell, and T. Vijaykumar, "Optimizing replication, communication, and capacity allocation in cmps," Proceedings of the 32nd annual international symposium on Computer Architecture, pp. 357–368, 2005.

[12] Pierfrancesco Foglia, Francesco Panicucci, Cosimo Antonio Prete, Marco Solinas ."An evaluation of behaviors of S-NUCA CMPs running scientific workload". Procedings of 12th euromicro conference on digital system design / architectures, methods and tools.

[13] Haixia Wang Dongsheng Wang Peng Li ."SRC-based Cache Coherence Protocol in Chip Multiprocessor," Frontier of Computer Science and Technology, 2006. FCST '06.

[14] Milo M. K. Martin, Mark D. Hill, and David A. Wood."Token Coherence: Decoupling Performance and Correctness". Appears in the proceedings of the 30th Annual International Symposium on Computer Architecture (ISCA-30) San Diego, CA, June 9-11, 2003

[15] On-ChipNetworks. Natalie Enright Jerger and Li-Shiuan Peh.

[16] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, and G. Hallberg. Simics: A full system simulation platform. IEEE Computer, 35(2):50–58, Feb. 2002.

[17] Winsconsin Multifacet GEMS Simulator http://www.cs.wisc.edu/gems/.

[18] Niket Agarwal, Li-Shiuan Peh and Niraj Jha, "GARNET: A Detailed Interconnection Network Model inside a Full-system Simulation Framework," CE-P08-001, Dept. of Electrical Enginering, Princeton University, Feb., 2008.

[19] Hangsheng Wang, Xinping Zhu, Li-Shiuan Peh and Sharad Malik, "Orion: A Power-Performance Simulator for Interconnection Networks," In Proceedings of MICRO 35, Istanbul, Turkey, November 2002.

[20] http://parsec.cs.princeton.edu.

[21] Kedzierski,K.MoretoM.,Cazorla F. and J.Valero," Adapting cache partitioning algorithms to pseudo-LRU replacement policies," 2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS), 1-12,April 2010.

Potential of Dynamic Binary Parallelization

Jing Yang, Kevin Skadron, Mary Lou Soffa, and Kamin Whitehouse
Department of Computer Science
University of Virginia
{jy8y, skadron, soffa, whitehouse}@cs.virginia.edu

Abstract

As core counts continue to grow in modern microarchitectures, automatic parallelization technologies are becoming increasingly important to fill the gap between hardware that has increased parallelism and software that is still designed for sequential execution. In previous research, we have proposed a novel dynamic binary parallelization scheme called T-DBP, which leverages hot traces to provide a large instruction window without introducing spurious control and data dependencies. In this paper, we conduct a limit study to estimate the maximum possible performance of T-DBP on the SPEC CPU2000 benchmark suite. Our results indicate an average speedup of 9.18x and 22.34x over sequential execution for the integer and floating point benchmarks, respectively. We also explain this high speed increase by quantitatively demonstrating that T-DBP uses runtime information to overcome two key handicaps of compile-time parallelization techniques. By artificially emulating the effects of these handicaps in T-DBP, the average speedup shrinks to 4.51x (integer) and 9.36x (floating point), respectively.

1 Introduction

With the number of cores increasing rapidly but the performance per core increasing slowly at best, software must be parallelized to maintain performance improvement. Manual parallelization typically yields the best speedups, because the programmer can choose new algorithms and data structures that are more amenable to parallelism. However, manual parallelization is often prohibitively time-consuming and error-prone, especially due to data races and memory-consistency complexities. Furthermore, some portions of code may simply be too difficult to understand or refactor for parallelization. Code is only parallelized when the return on investment is sufficient.

There has also been considerable research on automatic parallelization. However, most existing automatic techniques are performed statically at compile time and require source code to be analyzed, suffering three serious problems. First, in many cases, some or all of the source code and development tool chain has been lost or, in the case of third-party software, was never available. During the Y2K crisis, it was estimated that some companies were missing as much as 60 percent of their source code [6]. Second, modern applications are assembled and defined at run time, making use of shared libraries, virtual functions, plug-ins, dynamically-generated code, and other dynamic mechanisms. Furthermore, some software includes components that are written in different languages. All these aspects of separate development and compilation prevent the compiler from obtaining a holistic view of the program, leading to the risk of incompatible parallelization techniques, subtle data races, and resource over-subscription. Third, compile-time analysis has to conservatively respect all control and data dependencies that appear on the control flow graph (CFG). This deters parallelization, since many of these dependencies may not involve the execution path that is actually taken. All the above considerations motivate binary code parallelization at run time, which we call *dynamic binary parallelization (DBP)*. Without effective techniques that can operate on binary code, a large fraction of software will be left behind. And without the ability to parallelize at run time, opportunities for parallelism are curtailed.

In previous research [23], we have proposed a novel DBP scheme (*T-DBP*) based on the insight that programs tend to frequently repeat sequences of instructions called *hot traces*. T-DBP monitors a program at run time and dynamically identifies these hot traces, parallelizes them, and caches them for later use so that the program can execute in parallel every time a hot trace repeats. The parallelization purpose, however, imposes two significant challenges in trace construction, which have never been simultaneously overcomed by state-of-the-art technologies [2, 3, 7, 14, 15, 19, 25, 16]. First, traces have to be *as long as possible* to expose more distant parallelism. Second, traces have to be *logically atomic*. They should have a single entry point and exit point, and encapsulate only a single flow of control.

Thus, analysis can ignore all control dependencies (as well as derived data dependencies) among instructions within a trace, enabling more aggressive parallelization. This atomicity property necessitates speculative execution to recover program state when a trace deviates from the execution path that is actually taken. The dilemma, however, is that the longer a trace is, the more difficult it can be speculated accurately. T-DBP exploits multi-path execution [1, 21] and invents a holistic algorithm to balance trace length and speculation accuracy, which constructs the longest traces that can be accurately speculated on the available number of cores. Although our preliminary results have indicated an average speedup of 1.96x (8-way parallelization) over sequential execution, we believe that T-DBP is able to achieve much larger speed increases by improving on the initial prototype implementation.

Thus, we conduct a limit study in this paper to estimate the *maximum possible* performance of T-DBP, which can act as the guidelines for future improvements. We perform the experiments by appling T-DBP to the SPEC CPU2000 benchmark suite and compare the execution speed to sequential execution. Our results indicate an average speedup of 9.18x and 22.34x for the integer and floating point benchmarks, respectively. For all benchmarks, T-DBP is able to achieve 1) long traces, 2) large trace coverage, and 3) high speculation accuracy at the same time, indicating a large room for further improvements from [23].

We also use this limit study to explain *why* T-DBP has that performance. As described above, we hypothesize that T-DBP has high performance because it overcomes two key handicaps of compile-time parallelization techniques by: 1) parallelizing across boundaries between application and library code, and 2) only respecting control and data dependencies that appear in the actual execution path. We quantitatively test the hypothesis by repeating the experiments while artificially applying each of the handicaps to T-DBP. When both handicaps are applied, the average speed increase shrinks to 4.51x (integer) and 9.36x (floating point), respectively. These results support the hypothesis that T-DBP is able to use runtime information to exploit parallelism which compile-time techniques ignore.

2 Related Work

Existing DBP schemes are generally divided into two main categories: parallelizing the raw dynamic instruction stream (DIS) [9, 18, 20] and parallelizing the dynamically-generated CFG [5, 8, 24]. DIS-based techniques use extra hardware to combine multiple cores to work cooperatively as a wider core. Focusing on exploiting instruction level parallelism (ILP), this scheme has wide applicability, since ILP typically exists throughout the entire program (with different amounts). Limited by branch prediction accuracy and instruction window size, however, this scheme generally fails to exploit distant or coarse-grained parallelism, resulting in relatively mediocre speedups. On the other hand, CFG-based techniques expose a global view of the program and allow discovery of loop and thread level parallelism (LLP and TLP), which has the potential to produce much larger speedups. However, analysis on the CFG has to conservativly consider the large number of possible paths of program execution, many of which are rarely executed in a particular run. This requires the compiler to respect control and data dependencies that do not appear in the actual execution path, inhibiting parallelism opportunities. When source code is not available, this problem is exaggerated due to the lack of high-level information (e.g., types, variables, data structures), which is essential to achieve accurate alias analysis. Thus, it is not surprising that existing CFG-based techniques [5, 8, 24] have failed to parallelize at least half of the selected benchmarks.

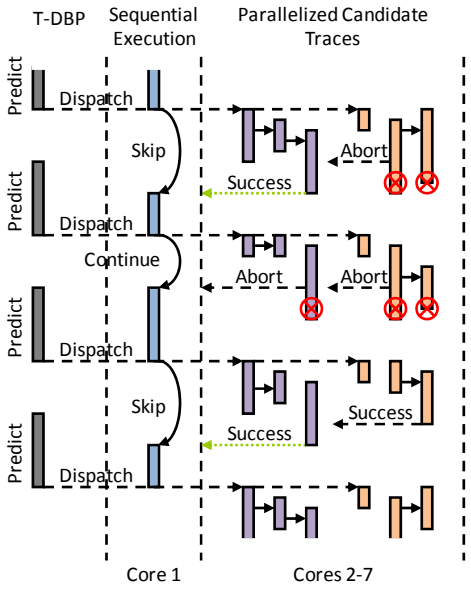


Figure 1. T-DBP uses one core for trace management plus sequential execution, and the remaining cores for speculative execution of parallelized candidate traces.

3 Overview of T-DBP

A conceptual overview of T-DBP is illustrated in Figure 1. Core 1 is instrumented with trace management functionality and starts to execute the unmodified, sequential binary. Simultaneously, the *trace constructor* monitors the instruction stream and identifies traces from frequently repeating instruction sequences. The traces are then pro-

Benchmark		Exec. on Traces	Avg. Trace Leng.	Mispred. Rate	Speedup
INT	gzip	86.97 %	108	1.95 %	1.54
	vpr	80.90 %	125	3.92 %	1.34
	mcf	31.28 %	93	11.77 %	1.08
	crafty	61.48 %	78	2.53 %	1.38
	parser	32.99 %	101	4.30 %	1.15
	eon	74.53 %	122	3.03 %	1.59
	bzip2	61.31 %	112	7.61 %	1.45
MEDIA	adpcm-dec	95.69 %	73	1.53 %	1.20
	adpcm-enc	97.10 %	83	0.42 %	1.08
	epic-dec	89.08 %	136	7.63 %	1.50
	epic-enc	96.52 %	862	8.11 %	4.63
	g721-dec	86.64 %	103	5.37 %	1.70
	g721-enc	70.55 %	105	5.27 %	1.57
	gsm-dec	97.93 %	1,098	8.06 %	1.43
	gsm-enc	97.23 %	756	2.54 %	2.53
	jpeg-dec	87.97 %	240	4.21 %	1.87
	jpeg-enc	59.23 %	138	2.55 %	1.28
	mpeg2-dec	91.31 %	175	2.17 %	2.01
	mpeg2-enc	68.81 %	394	10.56 %	1.85
FP	wupwise	99.12 %	2,179	2.24 %	2.50
	swim	96.74 %	836	4.38 %	1.53
	mgrid	99.85 %	7,890	0.19 %	3.73
	applu	97.58 %	4,583.	0.96 %	3.94
	mesa	98.06 %	567	0.52 %	2.41
	art	99.07 %	3,986	0.76 %	3.01
	equake	95.55 %	638	2.60 %	2.28
	ammp	79.64 %	182	1.71 %	1.31
	sixtrack	89.88 %	119	0.70 %	1.66
	apsi	98.62 %	3,362	1.23 %	2.45

Table 1. This table shows 1) the percentage of instructions (executed by the unmodified program) that are covered by correctly predicted traces, 2) the average length of the trace (in number of instructions) that commits in each correct prediction, 3) the trace misprediction rate, and 4) the speedup over sequential execution by performing 8-way parallelization and dispatching at most 32 candidate traces.

cessed by the *trace parallelizer* and stored in the *trace cache*. This parallelization process is offloaded to spare cores in order not to affect the sequential execution.

Thus, at every point during execution, the *trace predictor* checks for *candidate traces*: parallelized traces in the trace cache that 1) begin with the instruction that is about to be executed by the sequential binary, and 2) have a high probability of running to completion. If any exist, it suspends the sequential execution and launches them in the remaining available cores (Cores 2 to 7). The speculated traces operate on *copies* of the actual program state. If a trace deviates from the execution path that is actually taken, it aborts and its copy of program state is discarded. If any traces run to completion, one of them is selected and its copy of program state is committed to the suspended sequential execution, which "skips forward" in time to the end of the selected trace. Figure 1 illustrates three example scenarios. First, the right trace aborts and the left trace succeeds, causing the sequential execution to skip forward. Second, both traces abort and so the sequential binary continues running from the last dispatch point. Third, both traces succeed and the copy of program state from the left trace is selected to commit. In the last case, one trace is the prefix of the other trace, which happens infrequently in practice.

Table 1 summarizes the performance of T-DBP as implemented in [23], which uses a combination of novel and existing algorithms. One clear takeaway from this table is that T-DBP performs best on floating point benchmarks, followed by media benchmarks, and could only achieve mediocre speedups on integer benchmarks. This is not surprising since integer programs normally have more complicated control flows that are hard to predict and pointer-based memory accesses that are hard to disambiguate. The current algorithms have encountered significant difficulties to further increase trace coverage and trace length while maintaining high speculation accuracy. Thus, it is important and necessary to conduct a limit study to estimate the maximum possible performance of T-DBP, which can act as the guidelines for future improvements.

4 Limit Study Setup

We analyze the limits of T-DBP by making two idealizations about the hardware and algorithms: 1) the program runs on a many-core processor with an unbounded number of cores, and 2) the trace construction algorithm can always identify the most frequently repeating patterns of instructions. The first idealization not only assumes unlimited computing resources, but also guarantees perfect speculation accuracy since all candidate traces can be executed simultaneously. The second idealization maximizes trace length to expose more distant parallelism.

To conduct this limit study, we performed a five step process for each program in the SPEC CPU2000 benchmark suite[1]: 1) record the complete execution sequence of the program, 2) analyze the recording offline to identify the frequently repeating traces, 3) create a new execution sequence by replacing each trace in the original execution sequence with a parallelized version, 4) analyze the parallel execution time of the new execution sequence using a model of a shared-memory many-core processor, and 5) replay a linearization of the new execution sequence on a real machine and check correctness of the result: a successful replay implies correct synchronizations within the parallelized traces. All programs are executed using the test data sets as input to maintain a reasonable amount of recorded data. In the next five subsections, we describe in detail how we implemented each of these steps.

[1]The `perlbmk` benchmark was omitted because it recursively calls itself, starting multiple instances of our capture framework and exhausting memory of the machine.

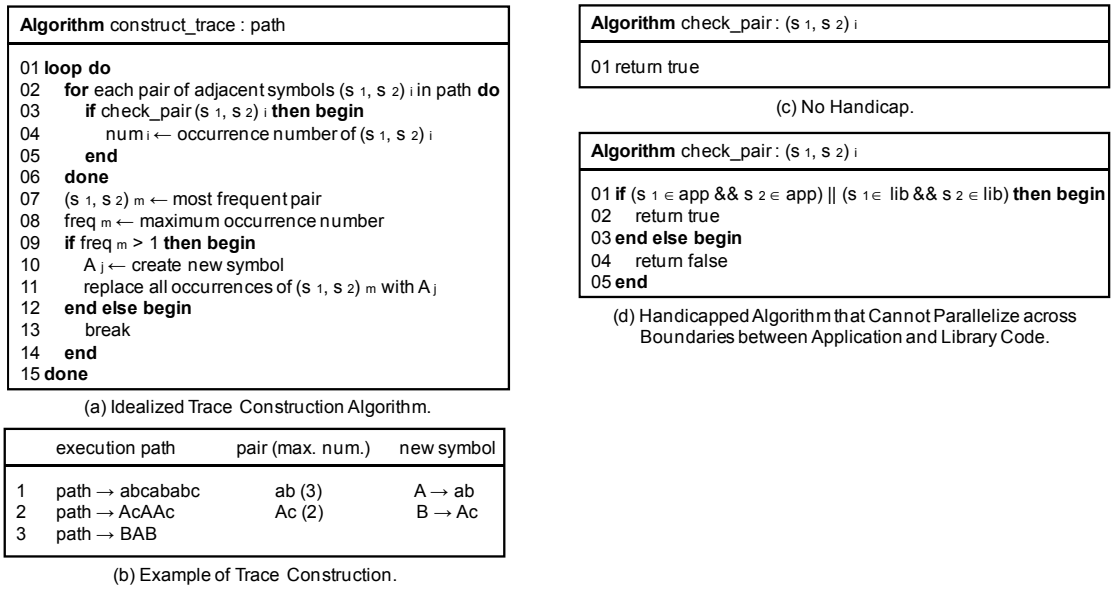


Figure 2. The idealized trace construction algorithm finds the most frequently repeating patterns of instructions in the entire execution sequence, as shown in the example. The handicapped version does not construct traces across boundaries between application and library code.

Other studies have built systems to efficiently record program execution [13, 26], and some can also replay the recording by capturing non-deterministic events such as interrupts, preemption, and user input [10, 22]. Our limit study framework is unique in that we can modify the execution sequence and replay the modified version to verify that it is equivalent to the original execution.

4.1 Recording Execution Sequences

We record the original execution sequence of the program by adding instrumentation code to the binary executable. This is performed by employing translation-based dynamic instrumentation to the program during its execution: whenever a new basic block is translated, instrumentation code is added to record the program counter whenever the basic block executes. Instrumentation code is also added to record the effective address of every load and store instruction, as well as the memory value of each load instruction. We record the actual memory values so that the program can be deterministically replayed. Otherwise, background operating system processes could change the state of certain system libraries, creating non-deterministic effects during program playback. We record the effective addresses of memory accesses to perform memory disambiguation, as described below in Section 4.3.

Recording the complete execution sequence of the program produces a large amount of information and so we use double buffering to reduce the runtime overhead [27] and apply the VPC3 algorithm to compress the collected information [4]. This greatly increases execution speed and reduces disk space requirements, although a typical one second program still required three minutes and fifty megabytes of disk space to record.

4.2 Analyzing Execution Sequences to Construct Repeating Traces

Once the execution sequence has been recorded, we construct traces by finding all frequently repeating patterns of instructions. We do this using an offline dictionary-based algorithm that is typically used for compression [12], illustrated in Figure 2(a). Initially every basic block is defined to be a unique symbol. We then identify the two symbols s_i and s_j that are the most frequent pair of adjacent symbols in the entire execution sequence (lines 2 to 8). If no pair appears more than once, the algorithm stops (line 13). Otherwise, we replace all occurrences of $s_i s_j$ with a new symbol A_j (lines 10 to 11). The execution sequence now has fewer symbols and the algorithm repeats to again find the most frequent pair of adjacent symbols. When the algorithm completes, all symbols remaining on the execution sequence become the selected traces. Figure 2(b) shows an example of how traces are constructed on an execution sequence of eight basic blocks (a, b, c, a, b, a, b, c). In the first iteration, ab is found to occur most frequently (three times)

and is replaced by a new symbol A. In the second iteration, Ac occurs two times and is replaced by a new symbol B. After that, no pair of adjacent symbols occurs more than once and the algorithm completes, constructing two different traces A (basic block sequence a, b) and B (basic block sequence a, b, c).

We can modify the trace construction algorithm to handicap T-DBP so that it cannot parallelize across boundaries between application and library code. More specifically, we replace the original `check_pair` function (Figure 2(c)) invoked on line 3 of the trace construction algorithm with an alternative version that only allows two adjacent basic blocks to be combined into a single symbol if both of them belong to application code or both of them belong to library code. The pseudo code for this handicapped algorithm is illustrated in Figures 2(d) and its effect on execution speed will be analyzed in Section 5.

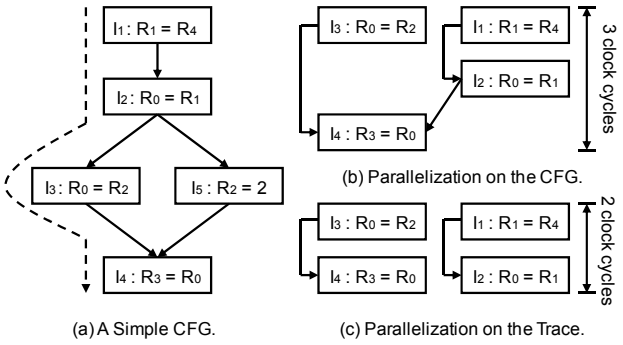


Figure 3. Analysis of the trace (dashed arrow) produces fewer true dependencies than analysis of the CFG, leading to improved parallel performance.

4.3 Parallelizing Execution Sequences

Once the repeating traces in the execution sequence are identified, they are parallelized using a modified version of the dynamic critical path scheduling algorithm [11], which is derived from previous research on allocating task graphs to fully-connected multiprocessors. This algorithm is selected since it was experimentally demonstrated to produce the minimum execution time among all comparable algorithms. For the purpose of trace parallelization, we define each instruction to be a separate task. The algorithm mainly includes the following four steps:

1. Eliminate all anti and output register dependencies in the trace through renaming. Identify all true dependencies and build the dependency graph. Initialize the current schedule to be an empty schedule. The effective addresses of memory accesses recorded in Section 4.1 are used to perform memory disambiguation.

2. Calculate the absolute earliest start time (AEST) and absolute latest start time (ALST) of each instruction based on the current schedule. Let L be the group of instructions with the smallest value of $ALST - AEST$, and pick instruction i from L that does not have predecessors in L.

3. Schedule instruction i on core j where 1) after insertion, it does not delay the ALST of all instructions already scheduled on that core, including itself, and 2) there are no violations of any true dependencies.

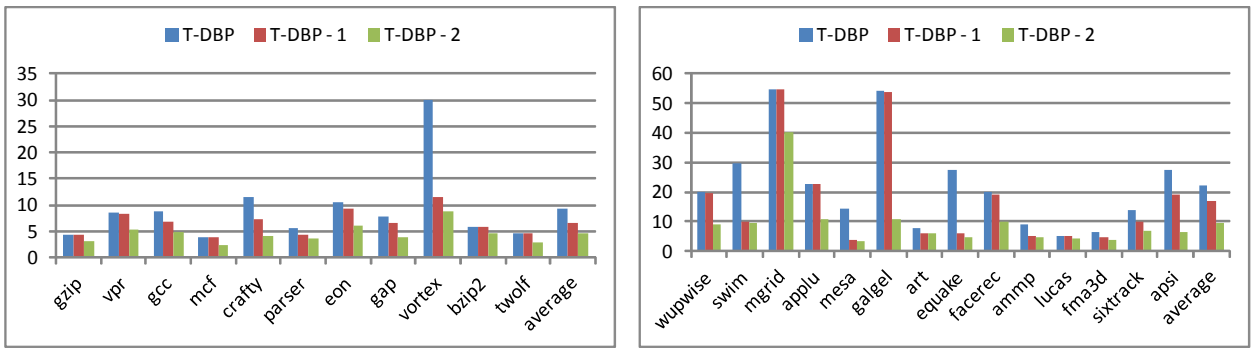
4. Go back to Step 2 if not all instructions are scheduled.

After all traces are parallelized, we replace their occurrences in the original execution sequence with the parallelized versions. This new execution sequence represents the ideal execution sequence that T-DBP might produce in the real world. Correctly replacing every single trace in the original execution sequence with the parallelized version corresponds to the idealized assumption of perfect speculation accuracy in our limit study.

We can also modify the trace parallelization algorithm to handicap T-DBP so that it has to respect all control and data dependencies that appear on the CFG. For example, Figure 3(a) illustrates the CFG of a small program snippet containing five instructions: I_1, I_2, I_3, I_4, and I_5. Analysis of this CFG reveals three true dependencies: $I_1 \rightarrow I_2$, $I_3 \rightarrow I_4$, and $I_2 \rightarrow I_4$. The last dependency exists because of the possible execution path through I_5. The best possible parallelization of the left branch (dashed arrow) in this CFG that respects all of the three true dependencies is depicted in Figure 3(b), with a parallel execution time of three clock cycles. In contrast, if the path along the left branch is converted into a trace at run time, an analysis of the trace would not find a true dependency $I_2 \rightarrow I_4$ because I_3 produces the freshest value of R_0. A parallelization of this trace would thus run the same instructions with a parallel execution time of only two clock cycles, as depicted in Figure 3(c). The effect of this handicapped algorithm on execution speed will be analyzed in Section 5.

4.4 Modeling Parallel Execution Time

To calculate the execution time of a sequence of instructions, we define each instruction to have an execution time of one clock cycle because of pipelining. We define the execution time of a parallelized trace to be the maximum AEST of all instructions in the trace plus one, to account for the execution of the last instruction. We require at least one clock cycle to separate any two instructions with true dependencies that execute on different cores, for inter-core com-



(a) CINT2000.

(b) CFP2000.

Figure 4. The standard T-DBP achieves an average speedup of 9.18x and 22.34x over sequential execution for the integer and floating point benchmarks, respectively. When all handicaps are artificially emulated, the average increase in execution speed shrinks to 4.51x (integer) and 9.36x (floating point), respectively.

munication. Software-based synchronization mechanisms such as locks, barriers, and monitors can cause more than one clock cycle of runtime overhead due to the interactions with the operating system, but special hardware such as the synchronization array [17] provides efficient, non-memory-based communication between different cores on the same chip. This technique enables the production and consumption of a single register value on different cores to be performed in back-to-back cycles, as long as the communicating buffer between the two cores is not full.

4.5 Verifying Parallel Execution Sequences

In addition to calculating the execution time, we can also execute the parallelized execution sequence to ensure correct synchronizations within the parallelized traces. To do this, all basic blocks and traces in the final execution sequence are linked together into a single executable, loaded into to its original address space, and replayed on a real machine. For parallelized traces, a linearization is created based on the final AEST of each instruction. This process does not test all possible linearizations and thus does not guarantee that the synchronization is one hundred percent correct, but it does create a linearization that is substantially different from the original execution sequence and has allowed us to verify many programs with reasonable confidence. During the replay, load instructions are not actually executed; the corresponding memory value that was recorded in the original execution is provided to the target register. This prevents background processes in the operating from producing non-deterministic values which can cause segmentation faults.

5 Experimental Results

We used the limit study framework described in Section 4 to analyze the performance of T-DBP on the SPEC CPU2000 benchmark suite. We tested and compared three different versions of the T-DBP implementation. The first implementation is standard T-DBP with no handicaps applied. The other two implementations use handicapped versions of the trace construction (Section 4.2) and parallelization (Section 4.3) algorithms, respectively. These three implementations are named as follows:

- T-DBP: both trace construction and trace parallelization are unconstrained.
- T-DBP−1: trace construction cannot cross boundaries between application and library code.
- T-DBP−2: trace construction cannot cross boundaries between application and library code; trace parallelization has to respect all true dependencies that appear in the CFG.

The performance of all three versions of the T-DBP implementation is illustrated in Figure 4. For the standard T-DBP that is unconstrained, the average speedup over sequential execution is 9.18x and 22.34x for the integer and floating point benchmarks, respectively. The higher speed increase for the floating point programs is likely due to the fact that they contain a larger fraction of numerical code, which introduces fewer true dependencies. When all handicaps are artificially emulated, the average increase in execution speed shrinks to 4.51x (integer) and 9.36x (floating point), respectively. These results support the hypothesis that the ability of T-DBP to overcome these two handicaps

Benchmark		Exec. on Traces	Avg. Trace. Leng.	Exec. in Libraries
INT	gzip	100.00 %	90	4.83 %
	vpr	99.99 %	50	4.06 %
	gcc	99.99 %	102	15.91 %
	mcf	100.00 %	106	41.82 %
	crafty	99.99 %	80	7.80 %
	parser	99.99 %	94	8.51 %
	eon	99.97 %	126	6.60 %
	gap	99.99 %	174	18.12 %
	vortex	100.00 %	1,879	12.96 %
	bzip2	100.00 %	118	0.18 %
	twolf	99.98 %	100	6.59 %
FP	wupwise	100.00 %	2,584	0.68 %
	swim	100.00 %	10,800	52.28 %
	mgrid	99.99 %	26,931	0.05 %
	applu	99.93 %	2,003	2.22 %
	mesa	100.00 %	4,503	70.62 %
	galgel	99.99 %	967	0.13 %
	art	99.99 %	1,465	6.67 %
	equake	99.95 %	748	68.43 %
	facerec	99.97 %	3,203	1.10 %
	ammp	99.99 %	953	15.56 %
	lucus	99.99 %	368	0.01 %
	fma3d	100.00 %	58	14.45 %
	sixtrack	100.00 %	2,081	8.31 %
	apsi	99.98 %	7,238	10.21 %

Table 2. This table shows 1) the percentage of basic blocks that are formed into traces, 2) the average number of basic blocks per trace, and 3) the percentage of basic blocks that belong to libraries.

accounts for its ability to explore a higher degree of parallelism than compile-time techniques do.

5.1 Analysis of Trace Parallelization

The only difference between T-DBP−2 and T-DBP−1 is that T-DBP−2 performs dependency analysis on the CFG during the parallelization process while T-DBP−1 performs dependency analysis directly on traces. Thus, the results of these two versions of the T-DBP implementation indicate the degree to which parallelism increases when using runtime information to eliminate spurious dependencies. The average speedup of T-DBP−1 over sequential execution is 12.47x, outperforming that of T-DBP−2 by a factor of 1.72x. This result validates the hypothesis that dependency analysis on the CFG is a significant handicap for compile-time parallelization techniques.

5.2 Analysis of Trace Construction

The relative results of T-DBP−1 and T-DBP indicate that parallelizing across boundaries between application and library code can improve the average speedup over sequential execution from 12.47x to 16.55x. Note that the speed increase does not necessarily correspond to the percentage of executed basic blocks that belong to libraries, which is listed in the fifth column of Table 2. In fact, `mcf` executes more library instructions than all other integer benchmarks but almost shows the minimum improvement between T-DBP−1 and T-DBP. Also note that the library instructions are being parallelized in both versions of the T-DBP implementation; the handicapped version only eliminates parallelization *between* application and library instructions. The degree to which this handicap affects the speed increase is related to the degree to which application instructions are interleaved with library instructions. These results validate the hypothesis that the inability to parallelize across boundaries between application and library code is a significant handicap for compile-time parallelization techniques.

When all handicaps are removed, T-DBP constructs very long traces. The fourth column of Table 2 lists the average number of basic blocks within each constructed trace, which can be as large as 1,879 for integer benchmarks (`vortex`) and $26,931$ for floating point benchmarks (`mgrid`). For all the programs, the average trace length is at least an order of maganitude larger than that was achieved in [23], indicating a large room for further improvements. The third column of Table 2 lists the percentage of basic blocks in the entire execution sequence that are formed into traces. This result shows that nearly all basic blocks are combined to construct longer traces and can be parallelized for later reuse. The singleton basic blocks that do occur are primarily from the prologue and epilogue of the program. Thus, a small number of traces can cover nearly the entire execution sequence of a typical program, which suggests good trace predictability. In a nutshell, an idealized system based on T-DBP can achieve 1) long traces, 2) large trace coverage, and 3) high speculation accuracy at the same time.

6 Conclusions

In previous research [23], we have proposed a novel DBP scheme (T-DBP) that leverages hot traces to provide a large instruction window without introducing spurious control and data dependencies. Although our preliminary results have indicated an average speedup of 1.96x (8-way parallelization) over sequential execution, we believe T-DBP is able to achieve much larger speed increases by improving on the initial prototype implementation. Thus, we conduct a limit study in this paper to 1) estimate the *maximum possible* performance of T-DBP, and 2) explain *why* T-DBP has that performance. Our results indicate an average speedup of 9.18x and 22.34x over sequential execution for the integer and floating point benchmarks, respectively. We explain this high speed increase by quantitatively demonstrating that T-DBP uses runtime information to overcome two key handicaps of compile-time parallelization techniques: 1) not parallelizing across boundaries between application and library code, and 2) conservatively respecting all con-

trol and data dependencies that appear on the CFG. By artificially emulating the effects of these handicaps in T-DBP, the average speedup shrinks to 4.51x (integer) and 9.36x (floating point), respectively.

References

[1] P. Ahuja, K. Skadron, M. Martonosi, and D. Clark. Multipath Execution: Opportunities and Limits. In *Proceedings of the International Conference on Supercomputing*, 1998.

[2] V. Bala, E. Duesterwald, and S. Banerjia. Dynamo: A Transparent Dynamic Optimization System. In *Proceedings of the Conference on Programming Language Design and Implementation*, 2000.

[3] D. Bruening, T. Garnett, and S. Amarasinghe. An Infrastructure for Adaptive Dynamic Optimization. In *Proceedings of the International Symposium on Code Generation and Optimization*, 2003.

[4] M. Burtscher. VPC3: A Fast and Effective Trace-Compression Algorithm. In *Proceedings of the International Conference on Measurement and Modeling of Computer Systems*, 2004.

[5] M. DeVuyst, D. Tullsen, and S.-W. Kim. Runtime Parallelization of Legacy Code on a Transactional Memory System. In *Proceedings of the International Conference on High Performance and Embedded Architectures and Compilers*, 2011.

[6] L. Freeman. Recover Missing Source Code to Overcome "Leaky-Roof Syndrome". *Enterprise Systems Journal*, 1997.

[7] R. Hank, S. Mahlke, R. Bringmann, J. Gyllenhaal, and W.-M. Hwu. Superblock Formation Using Static Program Analysis. In *Proceedings of the International Symposium on Microarchitecture*, 1993.

[8] B. Hertzberg and K. Olukotun. Runtime Automatic Speculative Parallelization. In *Proceedings of the International Symposium on Code Generation and Optimization*, 2011.

[9] E. Ipek, M. Kirman, N. Kirman, and J. Martinez. Core Fusion: Accommodating Software Diversity in Chip Multiprocessors. In *Proceedings of the International Symposium on Computer Architecture*, 2007.

[10] S. King, G. Dunlap, and P. Chen. Debugging Operating Systems with Time-Traveling Virtual Machines. In *Proceedings of the USENIX Annual Technical Conference*, 2005.

[11] Y. Kwok and I. Ahmad. Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessors. *Transactions on Parallel and Distributed Systems*, 7(5), 1996.

[12] J. Larsson and A. Moffat. Offline Dictionary-Based Compression. In *Proceedings of the Conference on Data Compression*, 1999.

[13] J. Larus. Whole Program Paths. In *Proceedings of the Conference on Programming Language Design and Implementation*, 1999.

[14] S. Mahlke, D. Lin, W. Chen, R. Hank, and R. Bringmann. Effective Compiler Support for Predicated Execution using the Hyperblock. In *Proceedings of the International Symposium on Microarchitecture*, 1992.

[15] M. Merten, A. Trick, E. Nystrom, R. Barnes, and W.-M. Hwu. A Hardware Mechanism for Dynamic Extraction and Relayout of Program Hot Spots. In *Proceedings of the International Symposium on Computer Architecture*, 2000.

[16] S. Patel and S. Lumetta. rePLay: A Hardware Framework for Dynamic Optimization. *IEEE Transactions on Computers*, 50(6), 2001.

[17] R. Rangan, N. Vachharajani, M. Vachharajani, and D. August. Decoupled Software Pipelining with the Synchronization Array. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, 2004.

[18] R. Ranjan, F. Latorre, P. Marcuello, and A. Gonzalez. Fg-STP: Fine-Grain Single Thread Partitioning on Multicores. In *Proceedings of the International Symposium on High Performance Computer Architecture*, 2011.

[19] K. Sankaralingam, R. Nagarajan, H. Liu, C. Kim, J. Huh, D. Burger, S. Keckler, and C. Moore. Exploiting ILP, TLP, and DLP with the Polymorphous TRIPS Architecture. In *Proceedings of the International Symposium on Computer Architecture*, 2003.

[20] D. Tarjan, M. Boyer, and K. Skadron. Federation: Repurposing Scalar Cores for Out-of-Order Instruction Issue. In *Proceedings of the Design Automation Conference*, 2008.

[21] S. Wallace, B. Calder, and D. Tullsen. Threaded Multiple Path Execution. In *Proceedings of the International Symposium on Computer Architecture*, 1998.

[22] M. Xu, R. Bodik, and M. Hill. A "Flight Data Recorde" for Enabling Full-system Multiprocessor Deterministic Replay. In *Proceedings of the International Symposium on Computer Architecture*, 2003.

[23] J. Yang, K. Skadron, M. L. Soffa, and K. Whitehouse. Feasibility of Dynamic Binary Parallelization. In *Proceedings of the Workshop on Hot Topics in Parallelism*, 2011.

[24] E. Yardimci and M. Franz. Dynamic Parallelization and Mapping of Binary Executables on Hierarchical Platforms. In *Proceedings of the Conference On Computing Frontiers*, 2006.

[25] W. Zhang, B. Calder, and D. Tullsen. An Event-Driven Multithreaded Dynamic Optimization Framework. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, 2005.

[26] X. Zhang and R. Gupta. Whole Program Traces. In *Proceedings of the International Symposium on Microarchitecture*, 2004.

[27] Q. Zhao, I. Cutcutache, and W. Wong. PiPA: Pipelined Profiling and Analysis on Multi-Core Systems. In *Proceedings of the International Symposium on Code Generation and Optimization*, 2008.

CRQ-based Fair Scheduling on Composable Multicore Architectures

Tao Sun, Hong An, Tao Wang, Haibo Zhang, Gu Liu, Mengjie Mao
School of Computer Science and Technology
University of Science and Technology of China
Hefei, China,
suntaos@mail.ustc.edu.cn, han@ustc.edu.cn,{tao36, kopcarl, gliu, mjmao}@mail.ustc.edu.cn

***Abstract*— Emerging composable chip multiprocessors (CCMPs) allow system software to dynamically configure chip computing resources into different number and types of cores at runtime. However, such dynamic heterogeneity poses a significant challenge to making fair scheduling, since the operating system traditionally only assumes fixed number and types of cores. To address the fair scheduling problem on CCMP, firstly, this paper introduces centralized run queue (CRQ) to capture the changing number of cores, and proposes a pipeline-like scheduling mechanism to hide the large scheduling decision overhead caused by the CRQ. Secondly, an efficiency-based dynamic priority (EDP) algorithm is proposed to keep fair scheduling, which can not only provide same applications with performance proportional to their priorities, but also ensure equal-priority (different) applications to get equivalent performance slowdowns when running simultaneously. In our experiments, several multi-program workloads are used for fairness evaluation, and the CFS algorithm is also ported to CCMP for comparing with EDP. The simulation results demonstrate that, besides achieving the fairness targets on CCMP, EDP also outperforms CFS by as much as 10.6% in average turnaround time under heterogeneous workload.**

Keywords- composable multicore; scheduling; fairness; centralized run queue (CRQ)

I. INTRODUCTION

An emerging family of performance-asymmetric multicore architectures, named composable chip multiprocessor (CCMP), has been proposed in recent years [1, 5, 6]. Instead of placing fixed heterogeneous cores at chip design time, CCMP consists of small homogenous *physical cores*, but provides *dynamic heterogeneity* at runtime by aggregating different number of physical cores into different sized *logical cores*. This enables CCMP to adapt the computing resources as the workload mix or application parallelism requirements changed. Figure 1 shows a 16-physical core CCMP and its two possible dynamic configurations. For ease of description, we use the terminology *P-N* to refer to a logical core which is composed of *N* physical cores, and we say P-x and P-y at different ***type*** when x≠y. Typically, a logical core P-N will have N-times issue width, instruction window, and level-1 I/D cache capacity than P-1. The system software can dynamically configure different number and types of logical cores at runtime by calling the hardware primitives (supported by CCMP micro-architecture).

However, such dynamic heterogeneity poses a significant challenge to operating system in making fair scheduling on CCMP. Firstly, schedulers on both symmetric-CMPs and fixed asymmetric-CMPs conventionally assume fixed number of cores, and build distributed run queues (DRQs) on each core for process scheduling. But in CCMP system, the number of logical cores is often changed to gain the benefits of dynamic heterogeneity, which makes the scheduler too expensive to maintain DRQs upon logical cores. As Figure 2 illustrates, when a logical core is being *created* or *freed*, the scheduler needs to *gather* or *scatter* plenty of processes simultaneously to maintain DRQs. Both operations are highly expensive.

Secondly, maintaining priority-based fairness on CCMP is more challenging, as different applications (or different phases of an application) probably run on different types of logical cores, and the speedup-gains also vary widely across different applications even on same core type (shown in Figure 3). Here we define fairness in two features: it provides same applications with performance proportional to their priorities; while it ensures equal-priority (different) applications to get equivalent performance slowdowns when running simultaneously, i.e. multi-program affects per-application performance equally if they have equal priorities.

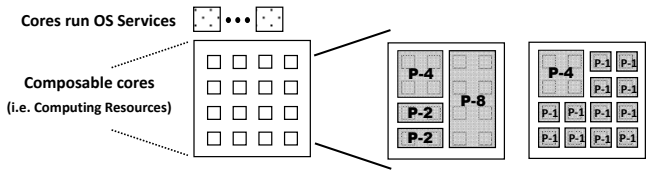


Figure 1: Example 16-physical core CCMP and its two dynamic configurations

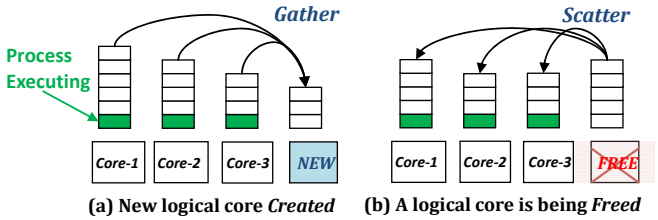


Figure 2: Gather/ Scatter operations caused by maintaining DRQs on CCMP

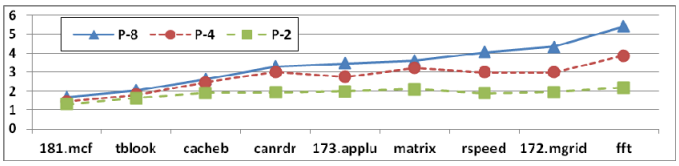


Figure 3: Speedups of different types of cores (P2/P4/P8) over P-1 running a subset of SPEC 2K and EEMBC applications.

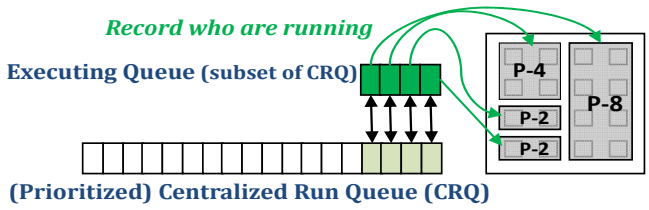


Figure 4: Example prioritized CRQ (can also be maintained as R-B tree in CFS).

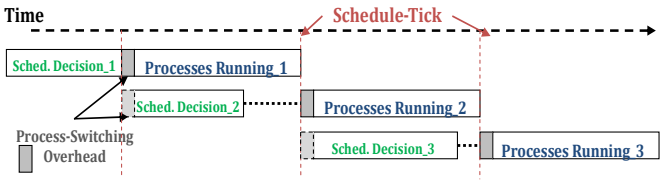


Figure 5: Pipeline-like scheduling mechanism. The scheduling decision overhead may vary but typically shorter than OS **schedule-tick** (0.5ms in this work).

The rest of paper is organized as follows. Section II describes the centralized run queue (CRQ) based scheduling mechanism on CCMP. Section III firstly describes our efficiency-based dynamic priority (EDP) algorithm, then discusses how EDP can achieve system-wide fairness. Section IV discusses the experimental results. Finally, Section V concludes.

II. CRQ-BASED SCHEDULING MECHANISM ON CCMP

A. Centralized Run Queue

As discussed in Section I, maintaining per-core distributed run queues (DRQs) upon CCMP is highly expensive. Instead, as illustrated in Figure 4, using centralized run queue (CRQ), the scheduler can easily capture the changing number of logical cores by maintaining an additional executing queue, which is simply a subset of the CRQ but records the current running processes. Furthermore, the CRQ can also be implemented as the prioritized queue for fair scheduling.

Besides the benefits, CRQ also brings larger scheduling decision overhead [7] and the scalability problem. In next subsection, we propose a pipeline-like scheduling mechanism to hide the decision overhead of scheduler. In this paper, we do not address the scalability in first place (we leave it for future work), but we would like to make a short discussion on it.

Scalability of using CRQ. It is assumed that the future many-core OS is probably to divide the cores into clusters and provide a set of OS services for each individual cluster in parallel [10]. Thus, the OS scheduling can be done at two levels: intra- and inter- clusters. In each cluster, the CRQ-based pipeline-like mechanism can be well used because of a limited number of cores (due to dividing). Between clusters, the CRQ of each cluster can be viewed as the per-cluster DRQ at cluster-level to support load balancing and cluster-level scalability.

B. Pipeline-like Scheduling Mechanism

Overview of the scheduling procedure on CCMP. Figure 6 presents an overview of the scheduling procedure on CCMP with CRQ. A new process will be assigned the smallest core type (i.e. P-1) when created. At the beginning of every schedule-tick (0.5ms in this work), the OS scheduler collects the runtime information of each running process (shown by ②), evaluates the *execution efficiencies* (defined in Eq.2) and

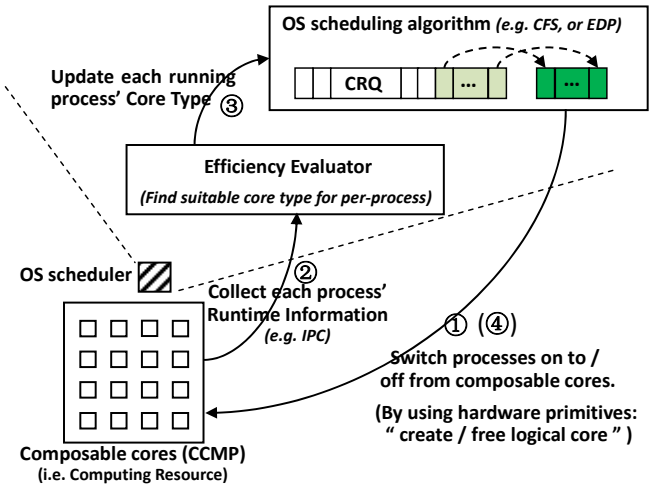


Figure 6: Overview of the proposed scheduling procedure on CCMP.

updates the core types (shown by ③); then, the scheduler switches processes based on scheduling decision and puts the selected ones to run on composable cores by calling the hardware primitives *create/free logical core* (shown by ①&④).

$$Speedup_{\text{P-N}} = \frac{IPC_{\text{P-N}}}{IPC_{\text{P-1}}}, \quad \boldsymbol{IPC_{\text{P-N}}} \text{ is task's IPC collected on P-N} \qquad \text{(Eq. 1)}$$

$$ExecutionEfficiency_{\text{P-N}} = \frac{Speedup_{\text{P-N}}}{N} \qquad \text{(Eq. 2)}$$

Pipeline-like scheduling mechanism. Also shown in Figure 6, OS services are running on separate cores. This provides an opportunity to "pipeline" OS scheduling decision with processes running on composable cores. Such pipeline-like mechanism is based on two facts. First, since the core types of processes are unchanged between two adjacent OS schedule-ticks, once the scheduler finishes process-switching, it in fact accurately knows the core type of each running process in this schedule period and the remaining time slice after this period. Second, once the scheduler finishes process-switching, the core running scheduler becomes idle.

Thus, as shown in Figure 5, while processes running on composable cores, the scheduler can correctly pre-calculate the dynamic priority (or key in CFS [8]) of each process, and pre-maintain the CRQ to be prioritized as the new-calculated priorities. When next schedule-tick arrives, since the CRQ has been properly pre-sorted, the scheduler only needs to update the core types of processes running in last period, and then simply put processes one by one to composable cores from the head of CRQ (if process-switching happens), until there is no enough resources to form the demanded core type. Therefore, the large scheduling decision overhead caused by the CRQ can be well hidden by such pipeline-like scheduling mechanism.

It is worth mentioning that most of existing scheduling algorithms, e.g. CFS [8], can be rebuilt on top of such pipeline-like scheduling mechanism with moderate modifications.

III. EDP FAIR SCHEDULING ON CCMP

This paper proposes an efficiency-based dynamic priority (EDP) algorithm to make fair scheduling on CCMP, whose target is to provide same applications with performance propor-

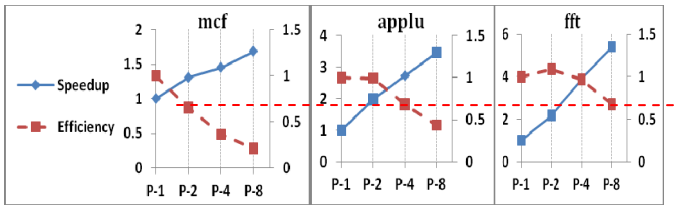


Figure 7: **Speedup (left Y-axis)** and execution **Efficiency (right Y-axis)** of different types of cores (P-1/ P-2/ P-4/ P-8) running three typical applications. Three figures are aligned in right Y-axis (efficiency).

tional to their priorities, as well as ensure equal-priority (different) applications to get equivalent performance slowdowns when running simultaneously.

EDP fair scheduling has two basic ideas: first, it keeps all applications running at efficiencies around a given threshold (typically given by OS scheduler); second, it consumes applications' time slices in different speeds depending on their core types. Following two subsections discuss the two ideas in detail.

A. Keep Applications Running at Similar Efficiencies

A powerful characteristic of CCMP is its capability to dynamically configure computing resources (i.e. physical cores) into different number and types of logical cores. This provides an opportunity to keep different applications running at similar efficiencies by assigning them different types of logical cores.

Figure 7 presents the speedups and efficiencies of three typically different applications running on different types of cores. We can see that, although each application commonly gets higher speedup at bigger core type, its execution efficiency continuously decreases. This suggests that if a system-wide efficiency threshold is given, different applications are able to achieve the threshold by running on different types of cores. E.g., *mcf* can reach the efficiency shown by the red line in Figure 7 when running on P-2, while *applu* on P-4 and *fft* on P-8.

As execution efficiency decreases when core type increases, in this paper we modified the PDPA algorithm [4] to find the biggest core type for each application in each phase (depending on execution efficiency) when a lowest_efficiency_threshold is given. Due to space limitation, the details of modified algorithm are not shown in paper.

B. Efficiency-based Dynamic Priority (EDP) algorithm

Figure 3 has shown that the speedup-gains vary widely across different applications. Thus, we argue that the fair scheduling on CCMP must consider per-application's *execution efficiency* (recall Eq.2). Based on this idea, this paper proposes an EDP algorithm, which takes the cores' heterogeneity (i.e. core types), the tasks' priorities and the execution efficiencies all into scheduling decision.

EDP keeps track of all runnable processes (in CRQ) by two arrays, an active array and an expired array, like O(1) scheduler [9] does. Let's say a new schedule round begins when the pointers of expired array and active array switched, i.e. the active array in last round becomes the expired array in current round and vice versa.

EDP schedules with following features:

① It uses the same algorithm as O(1) scheduler [9] to give each process a dynamic priority to avoid starvation. But it does not give any additional time slice when promoting the dynamic priority of one waiting process.

② It always selects to execute the processes in active array with higher dynamic priorities, and schdules the equal dynamic priority processes as first-in-first-out order in each round.

③ It assigns each process proportional amount of time slice according to its static priority (100~139).

④ It assigns each process different speed to consume its time slice, depending on its core type. For example, when a process runs on P-2, its time slice is reduced by 2 after one schedule-tick. But when it runs on P-4, its time slice is reduced by 4. Such a scheduling decision ensures equal-priority applications to get equal amounts of computing resources.

⑤ When one process uses up its time slice in this round, EDP will reset its time slice and move it from active array to expired array.

⑥ To avoid wasting of computing resources, when there are less than a pre-defined number (4 in this work) of processes in active array but the expired array is nonempty, EDP will add each remaining process in active array with its time slice in new round, promote its dynamic priority (10 in this work), and move it to expired array. Then the active and expired array are switched and new schedule round begins. This method may cause unfairness in each round, but fairness can be maintained among several rounds.

C. Summarize the Fair Scheduling on CCMP with EDP

EDP can ensure processes to get the proportional amounts of computing resources to their priorities. By employing the modified PDPA algorithm (discussed in subsection A), EDP can also keep different applications running at similar efficiencies by setting a lowest_efficiency_threshold (set to 0.8 in this paper). So EDP can provide the demanded fairness.

IV. EXPERIMENTAL RESULTS

A. Experimental Methodology

Platform. We model a 32-physical core CCMP in the experiments by using a cycle-accurate TFlex [1] simulator. The micro-architecture parameters are same as in [1]. Maximum 16 logical cores can be formed simultaneously and four types are allowed(P-1, P-2, P-4, P-8). We assume the future CCMP chips can run at 1GHz, so 0.5ms is equal as 0.5M cycles in simulator.

Overheads. The overheads of logical core reconfiguration and process-switching are all counted in final results. The *hard-cost* of per-core reconfiguration (include saving/restoring register and TLB states to/from memory, configuring a new logical core) is about 550 cycles; the *soft-cost* (include recreating branch predictor and L1 cache states on the new logical core) is counted into task's execution time. The scheduling decision overhead does not affect results, since the decision overhead can be well hidden by the pipeline-like mechanism (Section II.B).

Workloads. Nine of SPEC 2K and EEMBC applications are selected to build homogenous and heterogeneous workloads. They are shown in Fig. 3: *mcf, tblook, cacheb* (low speedups), *canrdr, applu, matrix* (medium speedups) and *rspeed, mgrid, fft* (high speedups). The simpoint tool is used for SEPC applications. Each task is chosen to be the length of 100M cycles of an application on P-1. Each workload consists of 30 tasks.

Evaluation. All the 30 tasks in each workload become runnable at the same beginning time. Whenever a process exits, a new one is started to keep the system load unchanged. The

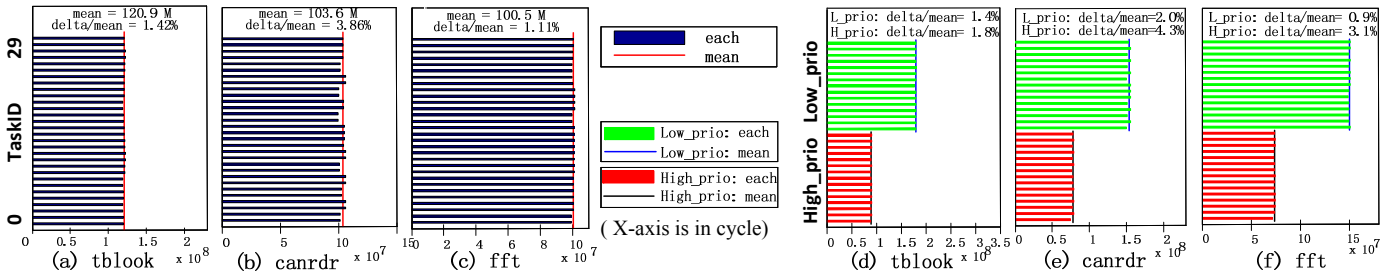


Figure 8: EDP scheduling on homogenous workloads. Each workload consists of same applications, and the applications are shown under figures. Tasks in (a) ~ (c) have same priority. But half of tasks in (d)~(f) have high-priority (100), while the others have low-priority (120), the ideal performance of high : low should be 2:1.

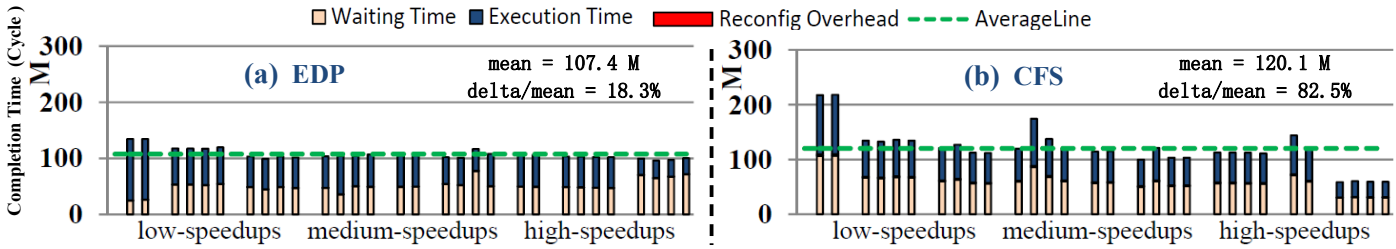


Figure 9: Fairness evaluation under heterogeneous workload, comparing between EDP and CFS scheduling on CCMP. Reconfig-overhead is too small to see in figures. The heterogeneous workload consists of 2 mcf, 4 tblook, 4 cacheb, 4 canrdr, 2 applu, 4 matrix, 2 mgrid, 4 rspeed, and 4 fft (total 30 tasks).

fairness is evaluated by the maximum deviation degree of the *completion time* (i.e. *turnaround time*) of each of 30 tasks. The average completion time is also used as the performance metric.

B. *Experimental Results*

Firstly, we evaluate if EDP can ensure priority-based fairness for homogenous applications. Figure 8 (a) ~ (c) show the completion time of each task in three workloads, which are consists of 30 copies of *tblook*, *canrdr*, and *fft*, respectively. The tasks in each workload (a) ~ (c) will get the same completion time if an ideal fair scheduler is used. We can see that EDP can provide good fairness. More precisely, the *delta* in the figure presents the degree of unfairness, which is calculated as:

$$delta = \text{maximum}\ (\ |\text{mean} - \text{best}|,\ |\text{mean} - \text{worst}|\ )$$

The workloads in Figure 8 (d)~ (f) are similar with (a)~ (c), except that the tasks have different priorities. Half of the tasks have high_priority (100), and the others have low_priority (120). The high_prio tasks will get as twice performance as low_prio ones under an ideal fair scheduling. We can see that EDP again provides good priority-based fairness.

Because CFS shares the similar scheduling decisions as EDP under the homogenous workloads, CFS results are not shown.

Secondly, we evaluate if EDP can ensure equal-priority applications to be affected equally under multi-program execution. The heterogeneous workload (includes 9 different applications) is used for evaluation. Figure 9 (a) shows the completion time of each task under EDP scheduling, and Figure 9 (b) shows the CFS scheduling results for comparison. Since CFS does not consider the heterogeneity between logical cores and the execution efficiencies of applications, it fails to ensure equal-priority applications getting equivalent performance slowdowns under multi-program execution. The results show that EDP is able to provide such fairness.

Finally, it is worth mentioning that the average completion time under EDP scheduling is 10.6% shorter than CFS when heterogeneous workload used (Figure 9), which means EDP provides better per-task performance than CFS in average.

V. Conclusion and Future Work

To make fair scheduling on CCMP, firstly, this paper introduced CRQ to capture the changing number of logical cores, then proposed a pipeline-like scheduling mechanism to hide the larger scheduling decision overhead caused by the CRQ. Secondly, this paper proposed an EDP algorithm, which can provide priority-based fairness under both homogenous and heterogeneous workloads. The experimental results also showed that EDP outperforms CFS by as much as 10.6% in average turnaround time under heterogeneous workload on CCMP.

We see two main areas of focus in next work. First, we will evaluate how much benefit can be brought by the composability through comparing several system metrics between symmetric-, asymmetric-CMPs (e.g. [3]) and CCMP. Second, we will address and evaluate the scalability of using CRQ on CCMP.

Acknowledgment

We thank Doug Burger, Dong Li, and Xiufeng Sui for suggestions. This work is financially supported by the National Research Program of China under contract 2011CB302501, 60633040, 2009AA01Z106.

References

[1] C. Kim, et al. Composable lightweight processors. *MICRO'* 2007
[2] H. Vandierendonck, and A. Seznec. Fairness metrics for multi-threaded processors. *Computer Architecture Letter, issue 1*. 2011
[3] T. Li, et al. Operating system support for overlapping-ISA heterogeneous multi-core architectures. *HPCA'* 2010.
[4] D. Gulati, et al. Multitasking workload scheduling on flexible-core chip multiprocessors. *PACT'* 2008
[5] E. Ipek, et al. Core fusion: accommodating software diversity in chip multiprocessors. *ISCA'* 2007
[6] Y. Watanabe, et al. WiDGET: Wisconsin Decoupled Grid Execution Tiles. *ISCA'* 2010
[7] S. P. Dandamudi. Reducing Run Queue Contention in Shared Memory Multiprocessors. *IEEE Computer'* 1997.
[8] I. Molnar, "Modular Scheduler Core and Completely Fair Scheduler [CFS]". http://lwn.net/Articles/230501. 2008.
[9] IBM DeveloperWorks, Inside the linux scheduler, http://www.ibm.com/developerworks/linux/library/l-scheduler. 2006
[10] D. Wentzlaff and A. Agarwal. Factored operating systems (fos): the case for a scalable operating system for multicore. *ACM Ope. Sys. Rev*. 2009.