Workloads, Scalability and QoS Considerations in CMP Platforms

Presenter – Don Newell
Sr. Principal Engineer
Intel Corporation

© 2007 Intel Corporation
Agenda

• Trends and research context
• Evolving Workload Scenarios
• Platform Scalability
• Platform QoS
Trends
- Cores, Threads, and Virtual Machines

Lots of hardware threads and greater software diversity will challenge cache, memory and I/O
Trends - Discussion Points

**Goal**: Scaling up to 100’s of logical processors on a single CPU die

- Scaling hardware threads means provisioning and re-architecting other platform resources

- Scaling hardware threads means learning to share – QoS revisited
**Trends** - Pool of Virtual Resources

**E.g. Front-End**
- Application Container
  - Workload
  - Workload
  - Workload
  - Guest OS
  - Virtual Resource Container
  - CPU
  - CPU
  - Memory
  - IO
  - IO
  - Memory

**E.g. Mid-Tier**
- Application Container
  - Workload
  - Workload
  - Guest OS
  - Virtual Resource Container
  - CPU
  - CPU
  - Memory
  - IO
  - IO
  - Memory

**E.g. Back-end**
- Application Container
  - Workload
  - Workload
  - Workload
  - Guest OS
  - Virtual Resource Container
  - CPU
  - CPU
  - Memory
  - IO
  - IO
  - Memory

**Virtual Machine Monitor**
- Creates, Deletes and Manages Dynamic Application Containers

**Processor Virtualization support**
- Processor Resources (Many cores)
  - CPU
  - CPU
  - CPU

**Processor supported Memory Virtualization**
- Memory Resources
  - Memory
  - Memory

**Processor supported IO Virtualization**
- IO Resources
  - IO
  - IO
Trends - Decomposed OS
Platform Scaling
Tera-Scale Workload Scenarios

Highly Multi-threaded Server Workloads
OLTP, J2EE App Servers, E-commerce, ERP, Search, etc
(TPC-C, SPECjbb2005, SAP, SPECjappserver2004, etc)

Virtualized Server Environments
(Workload Consolidation, Datacenter-on-chip Scenarios)

Cache/Memory Considerations:
- Performance => Overall Throughput
- Scalability => Headroom
- Quality of Service => Performance Isolation

Let’s begin with single server app........
And follow up with the consolidated scenario.......
Tera-Scale Cache Hierarchy Design

Hierarchical Sharing Benefits
- Reduces Replication
- Increases Effective Cache Size
- Better localized communication
- Reduced interconnect pressure

Hierarchical sharing increases caching effectiveness significantly

Shared MLC is equivalent to 2X of private MLCs
# OLTP Tera-Scale Case Study

<table>
<thead>
<tr>
<th>Core / Node</th>
<th>Single socket</th>
<th>Multiple Socket</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Perfect Last-Level Cache</strong></td>
<td>32 cores/4 threads</td>
<td>System Interconnect bottleneck</td>
</tr>
<tr>
<td>1 Large Core</td>
<td>Memory bottleneck</td>
<td>Need 40+ GB of B/W</td>
</tr>
<tr>
<td>~4 Small Cores</td>
<td>Memory bottleneck</td>
<td>Need 100+ GB of memory B/W</td>
</tr>
</tbody>
</table>

- Small core ~50% large core performance on server workloads
- 2X Performance Potential
- Need 100+ GB of memory B/W

**Significant Performance Potential => Memory Scalability Challenges**
Scalability Issues

Tera-scale Headroom requirements
- Start with 32 cores in 1st gen; Maybe grow to 48 cores in next gen?
- How much memory & interconnect bandwidth will we need?

32C / skt: 90-150GB/s
48C / skt: 150-240GB/s

32C / skt: ~40GB
48C / skt: ~50-70GB

Scaling Issues Just Get Worse
On-Socket DRAM Caches
(For Memory Scalability)

Enable Large Capacity L4s
• Low Latency
• High Bandwidth

Technologies
• 3D Stacking
• Multi-chip Packages (MCP)

Benefits
• Significant reduction in miss rate
• Avoids bandwidth wall

![Impact of Large DRAM$ (Major Server Workloads)](chart.png)
QoS and Performance Management
Background for QoS Discussion

**Trends**
- Increasing Core Count for Performance
- Increasing Workload Diversity
  - *Multi-Workload Scenarios in the Client*
  - *Virtualization and Consolidation in the Server*
  - *Heterogeneous Architectures for Graphics*

**Observations**
- Multi-core enables simultaneous execution of multiple workloads
- But not all Workloads are equal -- users do have preferences

*How well does the user-preferred application run? Should platforms optimize for the user-preferred application?*
## Resource Management

<table>
<thead>
<tr>
<th><strong>Capitalist</strong></th>
<th><strong>Commmunist/Fair</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>• No management of resources</td>
<td>• Fair distribution of resources</td>
</tr>
<tr>
<td>• If you can generate more requests, you will use more resources</td>
<td>• Give equal share of resources to all executing threads</td>
</tr>
<tr>
<td>• Grab as you will</td>
<td>• Does not necessarily guarantee equal performance</td>
</tr>
<tr>
<td>• <em>E.g. All of today’s policies</em></td>
<td>• <em>E.g. Partitioning resources for fairness and isolation</em></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Elitist</strong></th>
<th><strong>Utilitarian</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>• Focus on individual efficiency</td>
<td>• Focus on overall efficiency</td>
</tr>
<tr>
<td>• Provide more performance and resources to the VIP</td>
<td>• Give more resource to those that need it the most, less to others</td>
</tr>
<tr>
<td>• Limited resources to non-VIP</td>
<td>• <em>E.g. Cache-friendly vs. Unfriendly, resource-aware scheduling</em></td>
</tr>
<tr>
<td>• <em>E.g. Service Level Agreements, Foreground/Background</em></td>
<td></td>
</tr>
</tbody>
</table>
The Multi-Workload Problem

Platform does not distinguish in resource allocation

Preferred (foreground) application can suffer significant slowdown

Platform QoS can improve user-preferred application performance
Contention - Client side examples

- 30% exhibit 20%-5X slowdown
- 20% pairs exhibit 10-20% slowdown
- Rest exhibit <10% slowdown

Measured on a Dual core Conroe (4M cache, DDR2 667)

Similar data collected for server applications

Resource contention will impair the performance of important apps

*Performance Differentiation*
Application Behavior & Overall Performance

**Potential to improve overall performance**

Monitor resource usage and group/partition accordingly

Managing resource contention can improve overall throughput too

*(Performance Management)*
Service Level Agreements in the Enterprise

Disparate resource usage and contention hurts SLAs

(need for performance isolation and SLA enforcement)
Shared Platform Resources
Problem Summary

- CMP $\rightarrow$ Many heterogeneous threads, apps, VMs

Not all applications are equal
- Users have preferences
  - End-users (client) want to treat foreground preferentially
  - End-users (server) want service level differentiation (SLA) or isolation
- Applications use resources differently
  - Destructive vs. Constructive vs. Neutral Threads
  - No performance management to protect from bad behavior

Priority-based OS scheduling no longer sufficient
- With more cores, OS will allow high and low priority applications to run simultaneously and contend for resources
- Low priority applications will steal platform resources from high priority apps $\rightarrow$ loss in performance & user experience

Platform has no support for application differentiation
- Platform has no knowledge of preferences or resource usage
- Platform has no support for fine-grained tracking of many shared resources that have significant performance value
Platform QoS

QoS Exposure

QoS Hints via Architectural Interface

Feedback

Resource Monitoring

HW Policies

Resource Enforcement

QoS Enabled Resources

CPU Core

Cache

Memory

IO

Power

uArch resource usage

Cache Space

Bandwidth and Latency

I/O Response Time

Voltage/Frequency

Goals – Preferential Treatment of VIP, Better Overall Throughput
Visible QoS Spectrum (Cache/Memory)

**No QoS**
- No complexity
- \(\sim\) Good resource usage due to greedy approach
- \(\sim\) Okay throughput

**Possible Approach**
- Incremental Complexity
- Bridges the gap
- Extensible Architecture
- Satisfies Requirements for perf management, perf. differentiation

**Per-App Partitions**
- Significant Complexity
- \(\sim\) Lower resource usage due to per VM guarantee
- \(\sim\) Poorer throughput

**Classes of Service**
- Monitoring Per App
- Enforcement Per Class
- Full monitoring Per App
- Full enforcement Per App

**Possible Approach**
- Monitoring Per App*
- Enforcement Per Class
- E.g. 2MB, 6GB/s

**No determinism**
- No monitoring
- No enforcement

**Cache**
- No monitoring

**Memory**
- No enforcement

**Cache**
- Monitoring Per App*

**Memory**
- Enforcement Per Class

**Cache**
- Full monitoring Per App

**Memory**
- Full enforcement Per App

**Cache**
- E.g. 2MB, 6GB/s

**Memory**
- E.g. 2MB, 6GB/s
**QoS Aware Architecture - Cache**

**QoS Exposure:**
QoS Aware OS/VMM Platform QoS Register

**Requests tagged with Priority**

**Resource Monitoring:**
Monitor cache utilization per application

**Resource Enforcement:**
Enforce cache utilization for priority levels

**Platform Priority**
Sent through PQR

**QoS Aware OS/VMM:**
Platform Priority added to App state

**QoS Exposure:**
QoS Aware OS/VMM Platform QoS Register

**Set Application’s Platform Priority**

**Platform QoS Register**

**Core 0**

**Core 1**

**Cache**

**Memory**

**IO**

**Exposure QoS Interface**

**Requests tagged with Priority**

**Resource Monitoring:**
Monitor cache utilization per application

**Resource Enforcement:**
Enforce cache utilization for priority levels
Cache/Memory QoS Benefits

Client SPEC Case Study

QoS Potential
(Cache QoS + Memory QoS)

Response Time (Normalized)

Unmanaged Sharing
Memory QoS (BW reservation for Hi)
Cache QoS (Lo restricted to 20%)
Cache + Mem QoS
Dedicated (Best Case)

Response Time -- Lower is better

Based on Measurements, Simulations and Analytical Projections

Significant Benefits of Cache/Memory QoS
**QoS Aware Architecture: Power**

- **Power management knobs**
  - Voltage/Frequency scaling
  - Issue restriction
  - Gating

- **Resource Enforcement#1**
  - Victimize low priority for Power QoS

- **Resource Enforcement#2**
  - Use power throttling to enforce Performance QoS

**QoS Monitoring**
Cache, Memory, IO, Power
Power QoS Benefits

Improves performance of the user-preferred application

Performance impact with CPU clock throttling

- Max frequency (100%) to low priority → Max slowdown to High Priority Application
- Reduced frequency (89%) for low priority → Reduced Slowdown for Hi Priority
- Low frequency (39%) to low priority → Minimal slowdown for Hi Priority

Swim as low priority application
Virtualization: From VMs to VPAs (Managing Transparent Resources)

Virtual Machine Monitor manages CPU, IO and memory capacity
Virtual Platform Architectures manages performance with PQoS

Processor Virtualization support
Processor Resources
Cache/ Memory Allocation
Power Management
V/f. Instruction

Virtual Machine
Virtual Platform Architecture
Virtual Machine

Virtual Resource Container
CPU
IO
Memory Capacity

Virtual Resource Container
Workload
Guest OS

Virtual Resource Container
Workload
Guest OS

Virtual Resource Container
CPU
IO
Memory Capacity

Performance Isolation / SLA Differentiation motivates
Summary

Large-scale CMP is going to happen

- Lots of work to be done to identify and remove platform and architectural limitations preventing applications and execution environments from scaling up to 100’s of logical processors on a single CPU die

- Scalability concerns can be addressed
  - Hierarchy of Shared Caches
  - Large DRAM caches

- QoS concerns can be addressed
  - Dynamic Cache Allocation (Cache QoS)
  - Dynamic Power Management (Power QoS)

- Smart performance management requires more visibility be available to the execution environment
  - Resource utilization counters for schedulers, etc.