The Era of Single Chip Multiprocessors

Kunle Olukotun

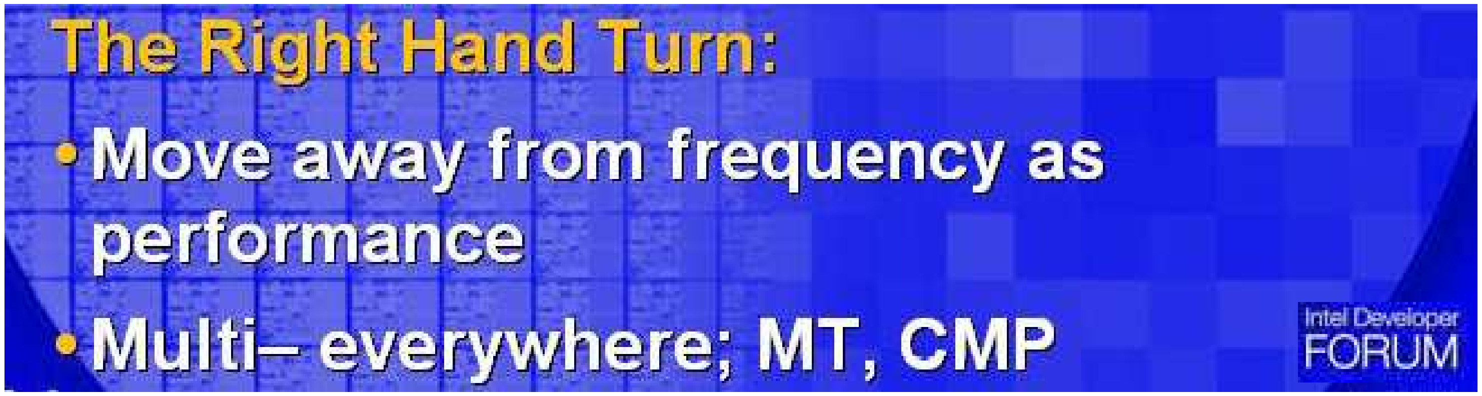
kunle@stanford.edu

Computer Systems Laboratory

Stanford University

The End of the Word As We Know It

- **Process Technology Stops Improving**
  - Moore's law but …
  - Transistors don't get faster (65nm vs. 45nm)
  - Wires are much worse
- **Single Thread Performance Plateau**
  - Design and verification complexity is overwhelming
  - Power consumption increasing dramatically
  - Instruction-level parallelism (ILP) is limited


The Right Hand Turn:

- Move away from frequency as performance
- Multi– everywhere; MT, CMP

Intel Developer FORUM

From Intel Developer Forum, September 2004

The Era of Single-Chip Multiprocessors

- **Single-chip multiprocessors provide a scalable alternative**
  - Relies on scalable forms of parallelism
    - Request level parallelism
    - Data level parallelism
  - Obvious scaling path that matches VLSI technology
  - Modular design with inherent fault-tolerance
- **Single-chip multiprocessors systems are here**
  - All processor vendors are following this approach
  - In embedded, server, and even desktop systems
- **CMPs need thread-level parallelism**

The Parallel Programming Problem

- **It’s the Software Stupid!**
  - Parallel programming is too difficult for average programmer
- **The reality**
  - Millions of people can write decent sequential programs
  - Few people can write correct parallel programs
    - Races, deadlock, memory consistency
  - Even fewer can write efficient parallel programs
    - Lock contention, coherence misses, false sharing
- **We must solve this problem: there’s no safety net**
  - Single thread performance is over
  - The future of computer architecture rests on the solution

Transactional Coherence & Consistency (TCC)

- **Threads and locks are the wrong programming model**

- **Parallel programming with transactions**
  - No threads, no locks, just transactions…

- **Transactions are the *only abstraction* for**
  - Parallel work
  - Communication
  - Memory coherence
  - Memory consistency
  - Failure atomicity and recovery
  - Performance optimization

- **Transactions run continuously**

Transactions Solve the Parallel Programming Problem

- **Identifying irregular parallelism**
  - Transactions allow for unproven, speculative parallelism
- **Difficulty of using locks**
  - Transactions provide coarse-grain atomicity for multi-object tasks
- **Reasoning about consistency models**
  - Consistency at transactions boundaries
- **Difficulty of tuning parallel programs**
  - Transactional execution identifies bottlenecks & optimizations
- **Handling faults and recovery**
  - Transactions support fault isolation and atomic recovery

- **Transactions have a problem**
  - Transactional memory is slow
  - Now there is an architecture problem worth solving