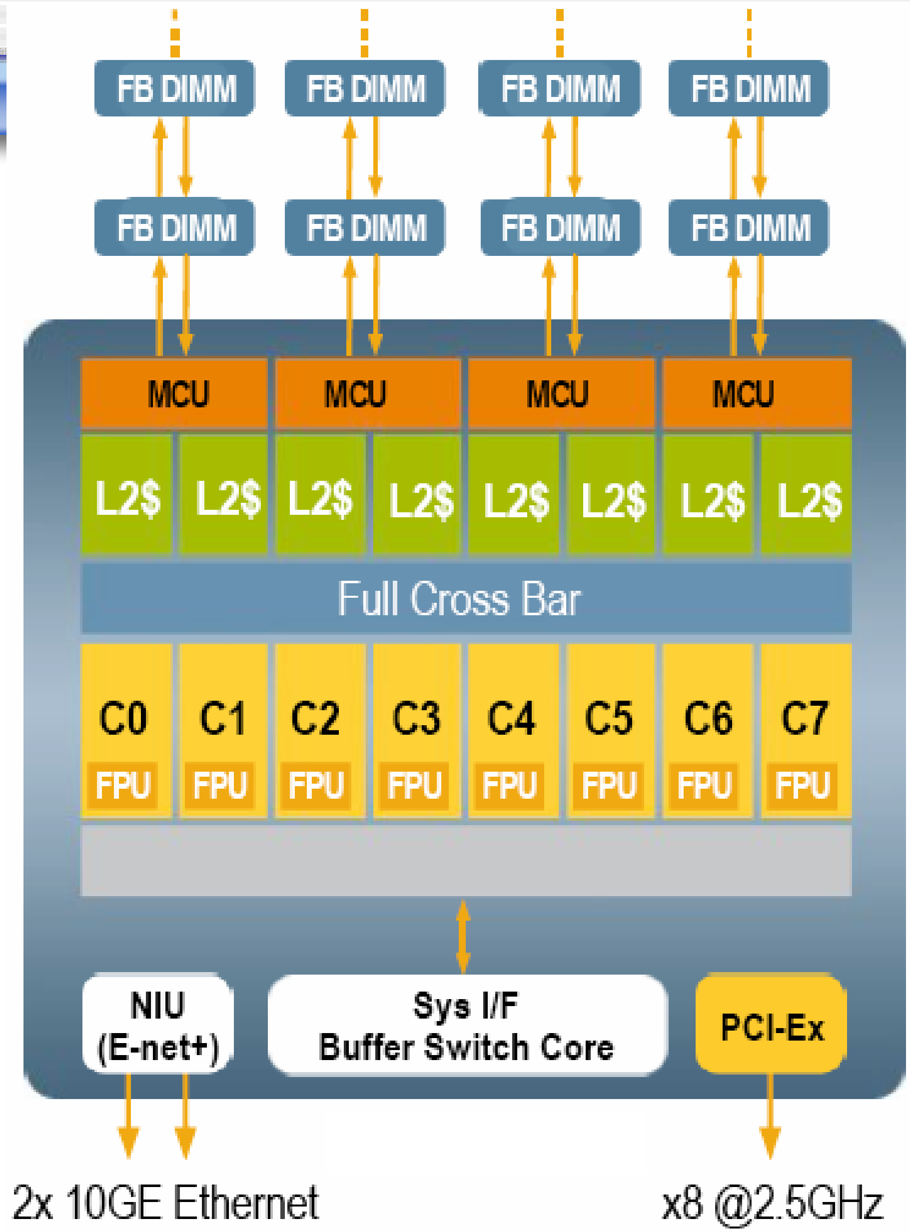
Delivering on the Multi-Core Promise

Kunle Olukotun

Stanford University

March 2007

Good News: High Throughput



- Sun Niagara 2
  - 8 cores x 8 threads = 64 threads
  - Low latency data sharing
  - High throughput/Watt
- Commercial servers
  - Abundant request level parallelism
  - Performance/Watt important
- Scientific and AI
  - Abundant data-level parallelism
  - Matrix, ML, SVM
  - Performance/Watt important

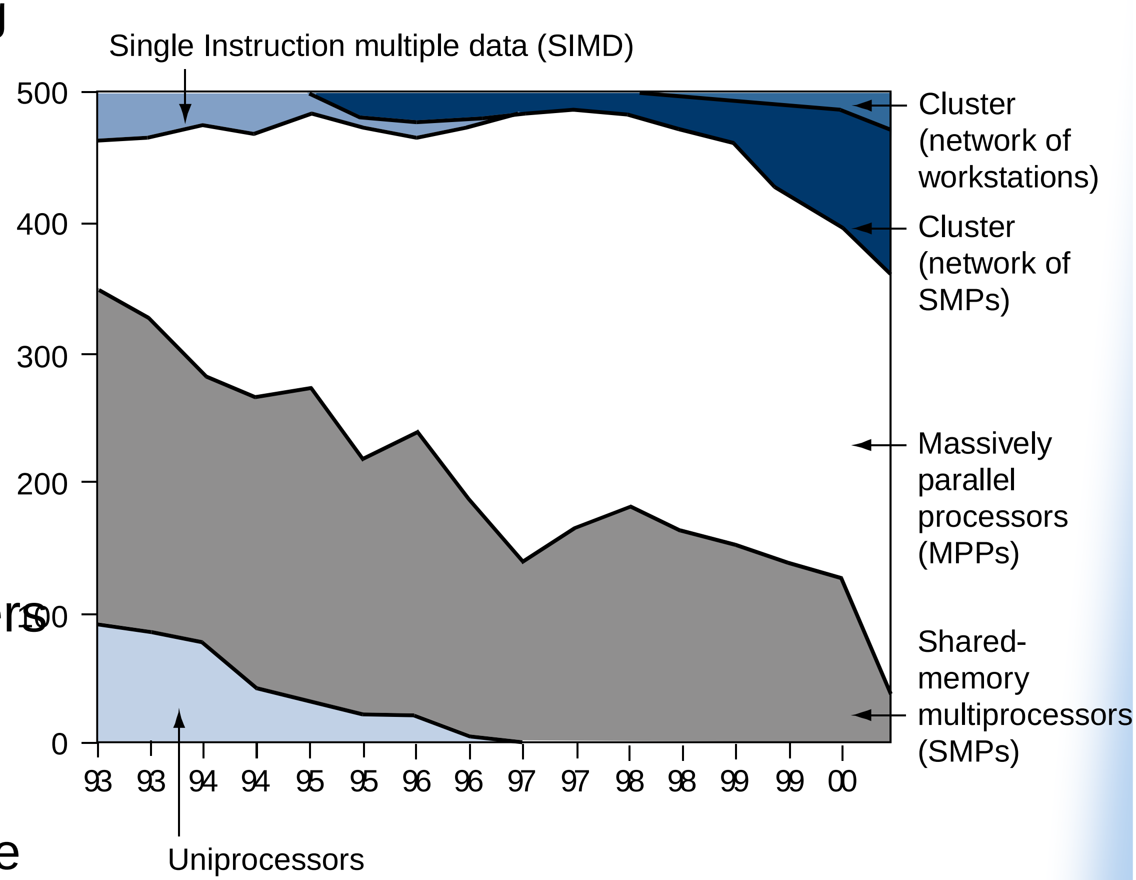
Bad News: Parallel Programming Gap

- By 2010, software developers will face…
- CPU's with:
  - 20+ cores
  - 100+ hardware threads
  - Deep memory hierarchies
- GPU's with general computing capabilities
- **Parallel programming gap**: Yawning divide between the capabilities of today's programmers, programming languages, models, and tools and the challenges of future parallel architectures and applications
- Clearly, we need to do more work
  - Automatic compilation will not scale to hundreds of threads

Shared Memory vs. Message Passing

- Lots of discussion in 90's with MPPs
  - SM much easier programming model
  - Performance similar, but MP much better for some apps
  - MP hardware is simpler
- Message passing won
  - Most machines > 100 processors use message passing
  - MPI the defacto standard
- Programmer productivity suffers
  - It takes too long to do "computational science"
  - Architectural knowledge required to tune performance

Plot of top 500 supercomputer sites over a decade



Very High-Level Programming Paradigms

- Need new parallel programming paradigms
  - Raise level of abstraction
  - Domain specific programming languages
  - Ease programming and extraction of parallelism
- Map-Reduce
  - Data parallelism (data mining, machine learning)
  - CMPs or clusters
- SQL
  - Information data management
- Synchronous Data Flow
  - Streaming computation
  - Telecom, DSP and Networking
- Matlab
  - Matrix based computation
  - Scientific computing
- Stitch together with scripting (Python, Ruby)

Parallelism Under the Covers

- Java and C++
- Streams
  - Beyond message passing
  - Data parallelism
  - Explicitly managed data transfers
  - Maximize use of memory and network bandwidth
- Transactions
  - Beyond shared memory
  - Thread-level parallelism
  - Eliminate locking problems and manual synchronization
  - Structured parallel programming

Transactional Memory

- Locks are broken
  - Performance – correctness tradeoff
    - Coarse-grain locks: serialization
    - Fine-grain locks: deadlocks, livelocks, races, …
  - Cannot easily compose lock-based code
- Programmer specifies large, atomic tasks
  - atomic { some_work; }
  - Multiple objects, unstructured control-flow, …
  - Declarative: user simply specifies, system implements details
- TM simplifies parallel programming
  - Parallel algorithms: non-blocking sync with coarse-grain code
    - Performance = fine grain locks
  - Sequential algorithms: speculative parallelization

Beyond Concurrency Control

- Atomicity & isolation are generally useful
  - For debugging, checkpointing, exception handling, garbage collection, security, speculation …
- These may be TM’s initial “killer apps”
- But they also change the requirements
- Cheap transactions for pervasive use
- “All transactions, all the time”
  - Stanford Transactional Coherence & Consistency (TCC)