

Easy-to-Write & Scalable Shared-Memory Applications with Transactions

Christos Kozyrakis

Computer Systems Laboratory
Stanford University
<http://csl.stanford.edu/~christos>

Parallelism is Finally Mainstream...

- All computer vendors are now building multiprocessors
 - Diminishing returns from uniprocessor architectures
 - 4, 8, 16, ..., 1024, ... processors on a chip
- Scientific computing no longer a niche
 - All programs must now become parallel programs
 - Need: practical & efficient parallel model
 - Parallel programs must be scalable and portable
 - Need: write code once, run efficiently at any scale
- This talk
 - Look at a promising solution from mainstream computing
 - Transaction-based shared-memory
 - Can it help with large-scale parallel computing?

The State of the Art

- **Shared-memory multiprocessors**
 - Implicit communication hidden from programmer
 - Easy to write first version; difficult to optimize
 - Difficult to write SM programs with >16 CPUs (NUMA)
- **Message-passing multiprocessors**
 - Explicit communication orchestrated by programmer
 - Difficult to write first version; simpler to tune afterwards
 - MPI: the defacto standard for large-scale machines
 - But too painful to use in most commercial environments
- **Common pains**
 - Programmer productivity suffers
 - Architectural knowledge required to tune performance
 - Each 10x increase in scale requires revisiting everything

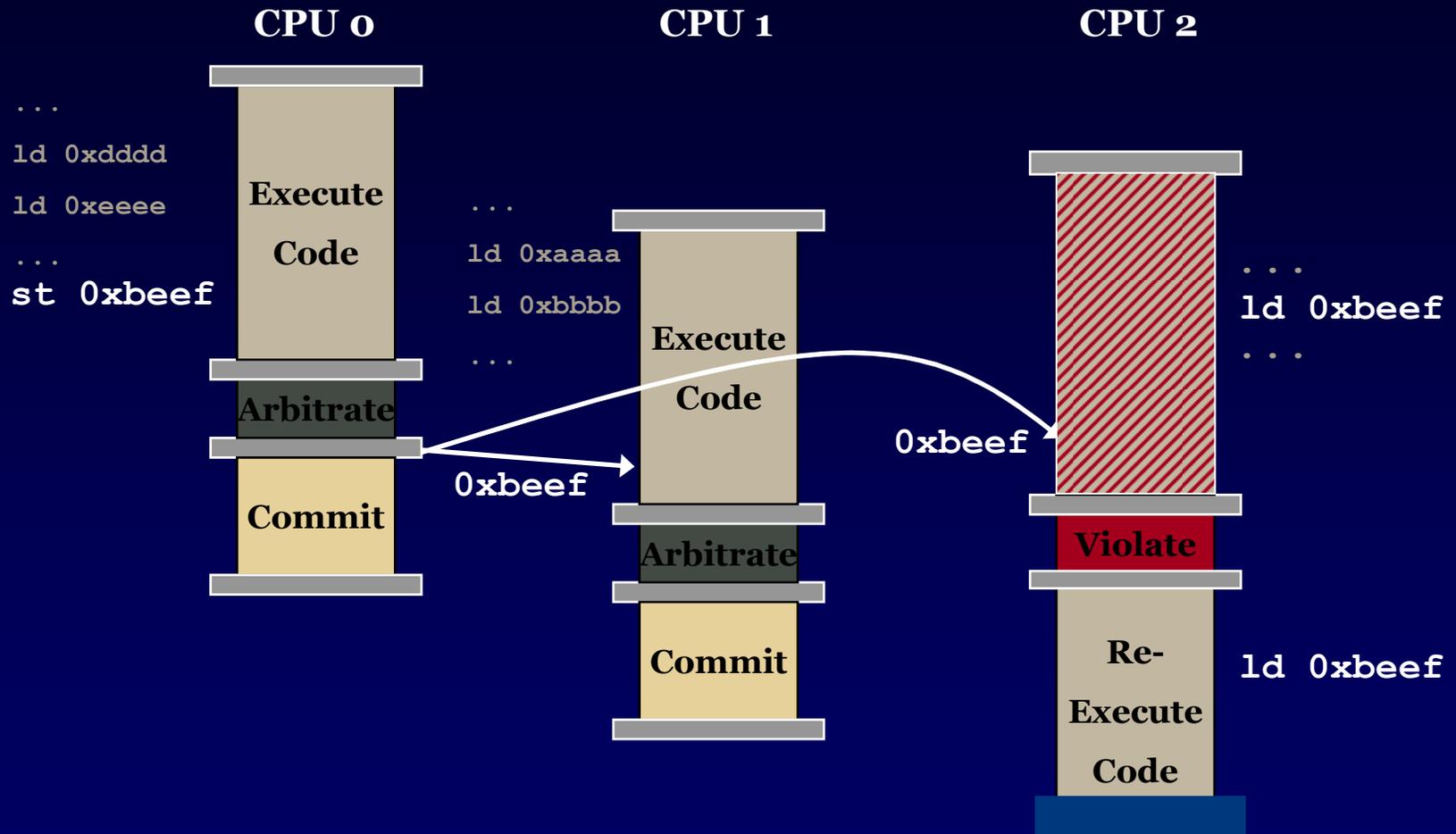
The Quest for Scalable Parallelism

- Summary: we want SM ease with MP performance
- Application characteristics
 - Large data-sets that stress caches and interconnects
 - Irregular & adaptive patterns that complicate programming
- Detailed requirements
 - Scalable execution resources (FLOPS)
 - This is the easy part of the problem
 - Efficient mechanisms to manage locality & communication
 - Methods to instrument & analyze system behavior
 - Methods to dynamically tune program to system behavior
 - Reliable operation in the presence of faults

Transactional Memory (TM) 101

- **Shared-memory with transactional semantics**
 - Program access shared data using atomic tasks
 - System provides atomicity, isolation, and consistency
- **Parallel performance through optimistic concurrency**
 - Assume independence and execute without any locks
 - If not true, abort and re-execute
- **TM simplifies parallel programming**
 - Coarse-grain, non-blocking synchronization for parallel algorithms
 - Speculative parallelization for sequential algorithms

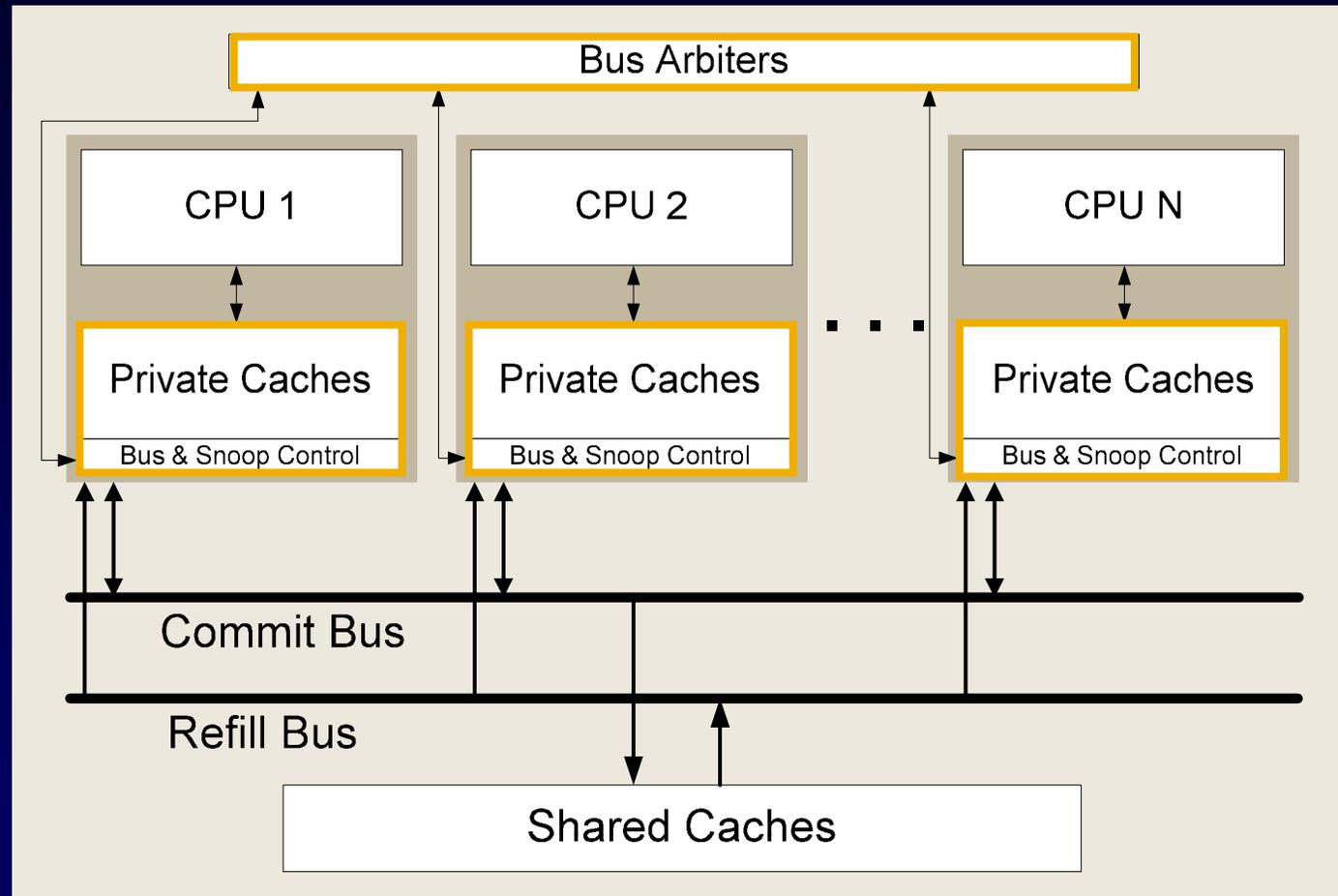
TCC: All Transactions, All The Time



Transactional coherence with deadlock-freedom guarantees
Intuitive consistency model that allows aggressive re-ordering

See [ISCA'04] for details

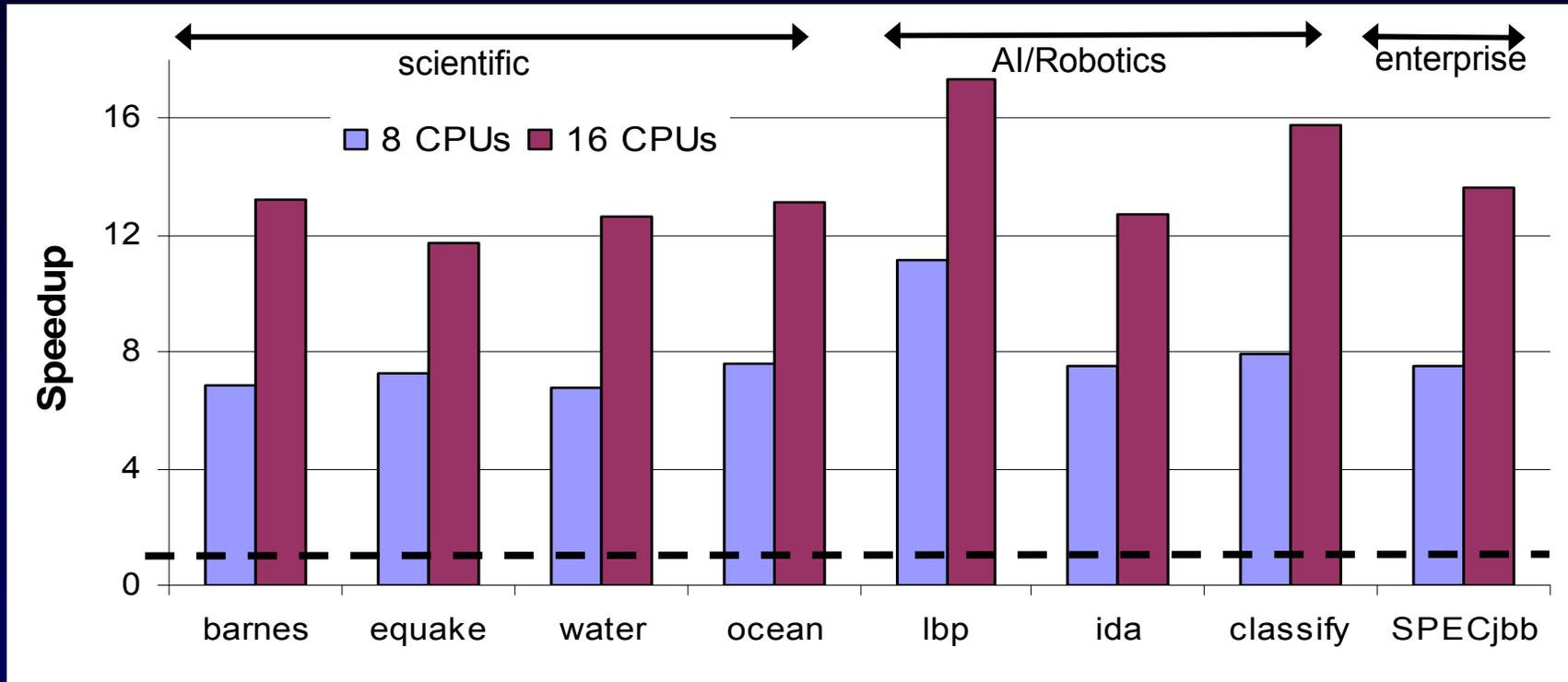
Small-scale Implementation (CMP)



Changes for TCC support

- Similar implementations for other CMP systems

Small-scale Performance

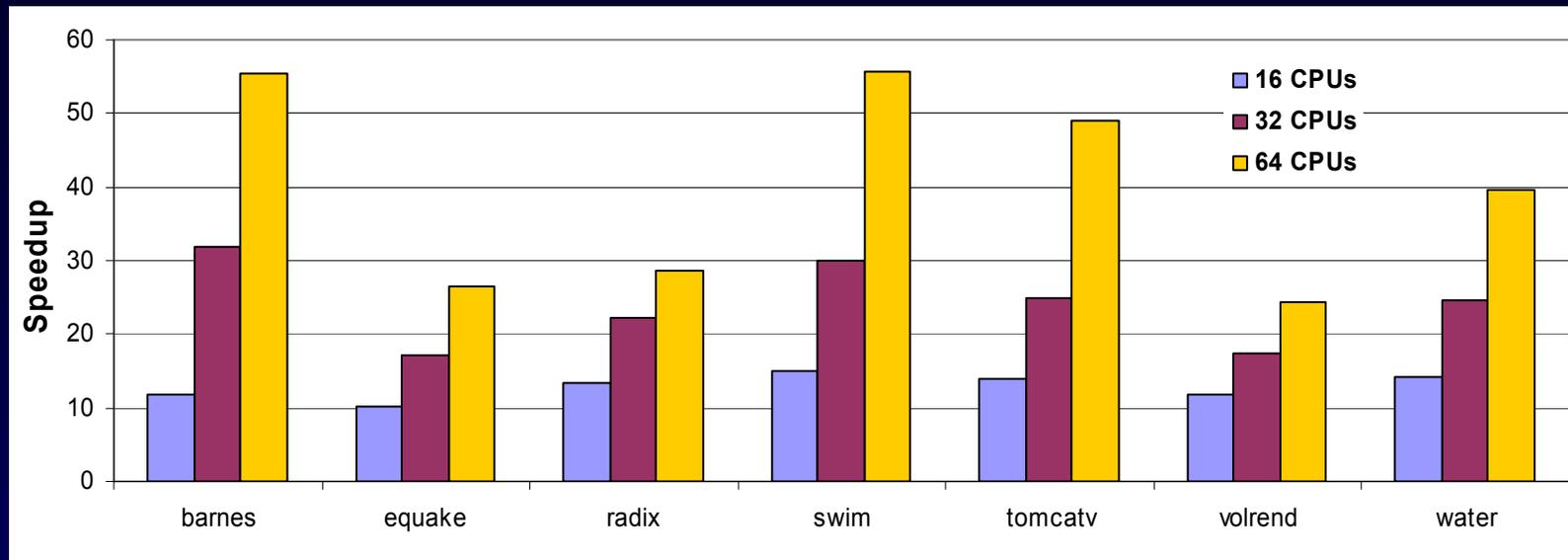


- Good performance across applications domains
- Easy to program and tune using feedback from hardware
 - TCC hardware continuously monitors memory accesses
 - Can identify most important performance bottlenecks for programmer

Large-scale Transactional Memory?

- Can TM scale beyond CMPs?
 - How do you implement TM in a NUMA environment?
- Can communication be optimized automatically?
 - How do you reach efficiency of message-passing model?
- Can TM assist with system reliability?
 - How do we exploit the atomicity in the case of faults?
- What is the prototype system for this research?
 - How do we provide a machine fast enough for scientists and flexible enough for architects?

Large-scale TCC Performance

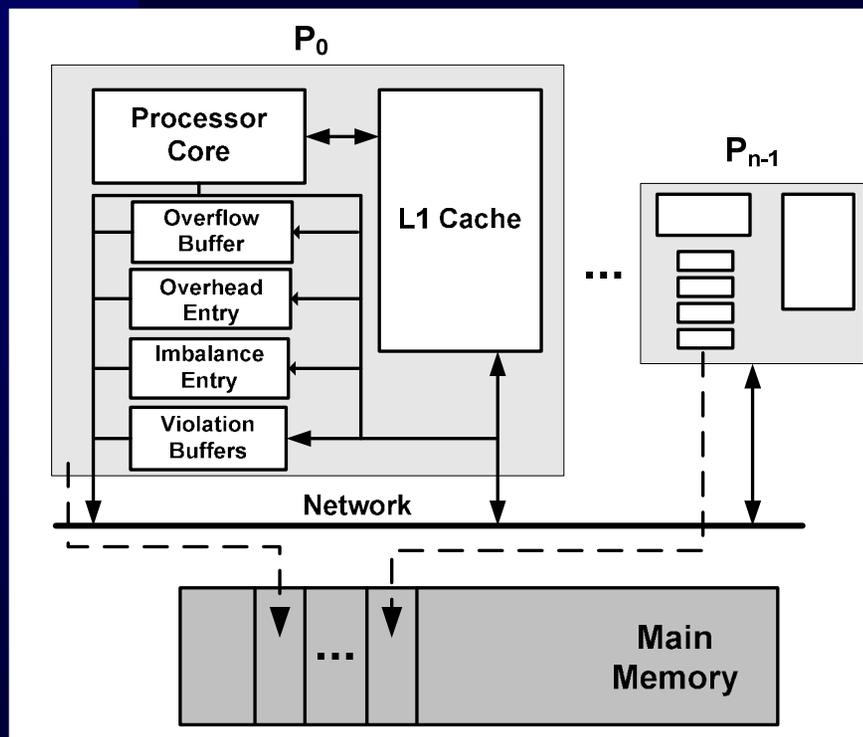


- **Directory-based implementation for NUMA systems**
 - Using parallel commit with two-phase protocol
 - Same execution model from programmer's perspective
- **Scalable performance for large processor counts**
 - Limited by dataset sizes for most of our experiments

Automatic Locality Optimizations

- **Build upon continuous memory monitoring**
 - Learn which data accessed within same transaction
 - Learn associations between code and data groups
 - Learn common producer/consumer patterns
- **Optimizations enabled**
 - Aggressive prefetching without sequential patterns
 - Schedule transactions close to their data
 - Proactively turn transaction commits to message sends
- **The overall opportunity**
 - Get message-passing behavior from a shared-memory system

TCC Profiling Environment



See [ICS'05] for details

- HW continuously track performance
 - Log events, cause, cost
 - Aggregates over multiple occurrences
 - Periodically flush to main memory
- Profiling accuracy
 - Pinpoints top performance problems
 - Type of performance problem
 - Related PCs, object addresses
 - Optimize applications in 2-3 steps
- Profiling cost
 - <1% performance loss
 - <1% area overhead

Reliability Optimizations

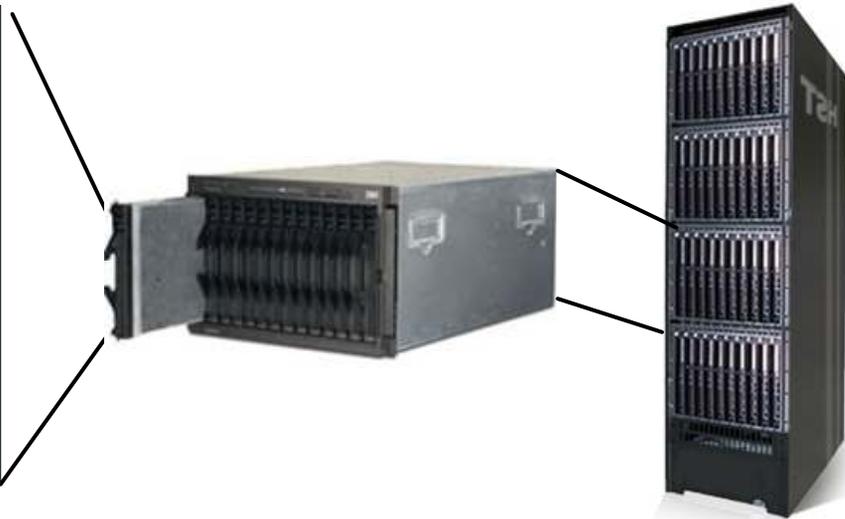
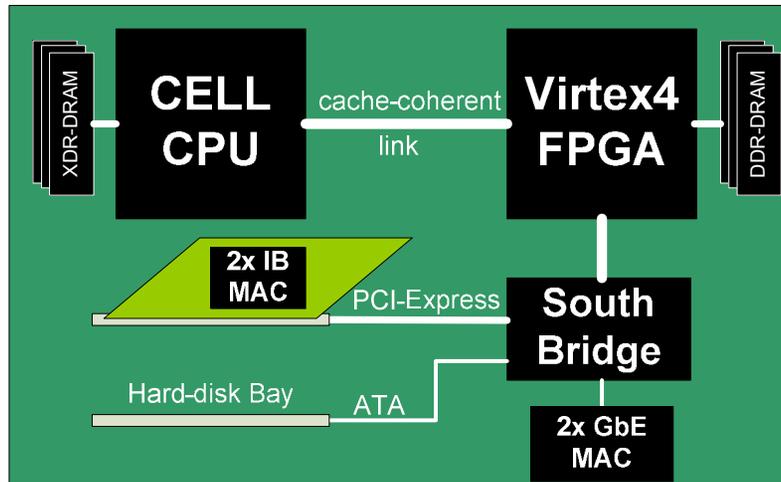
- **Micro-level: TM provides failure atomicity**
 - Easy “undo” of computation after fault is detected
 - Works with for both hardware and software faults
 - Allows software to decide how to best handle error recovery
 - Retry, migrate & retry, fail safely...
- **Macro-level: TM simplifies app-level checkpointing**
 - Transactions define clean boundaries for checkpoints
 - Can take checkpoint without stopping application
- **The overall opportunity**
 - A reliability framework that requires minimal programmer or operator involvement

The Research Infrastructure: FARM

CELL Blade Board

IBM BladeCenter Chassis

Blade Server Rack



- An industrial strength, scalable prototype
 - Full chassis: 19 TFLOPS and 280GB DRAM
- Advantages
 - Programmers: research features at industrial speeds
 - Architects: customizable memory & communication system

Conclusions

- **Large-scale parallelism is now a general need**
 - All programs must now become parallel programs
 - Parallel programs must be scalable and portable
- **Transactional memory**
 - Shared-memory with atomicity guarantees
 - Good performance with easy coding/tuning for CMPs
- **TM can also help with scalable parallelism**
 - Automatic optimizations for locality
 - Support for system reliability
- **Promising direction for parallel systems research**
 - Hardware, runtime, programming models, ...

Questions?

More info at <http://tcc.stanford.edu>