L23: Parallel Programming Retrospective

December 3, 2009

Administrative

- Schedule for the rest of the semester
  - "Midterm Quiz" = long homework
    - Return by Dec. 15
  - Projects
    - 1 page status report due TODAY
      - handin cs4961 pstatus <file, ascii or PDF ok>
    - Poster session dry run (to see material) Dec. 8
    - Poster details (next slide)
- Mailing list: cs4961@list.eng.utah.edu

12/03/09

Poster Details

- I am providing:
- Foam core, tape, push pins, easels
- Plan on 2ft by 3ft or so of material (9-12 slides)
- Content:
  - Problem description and why it is important
  - Parallelization challenges
  - Parallel Algorithm
  - How are two programming models combined?
  - Performance results (speedup over sequential)
- Example

12/03/09

Outline

- Last New Topic: Transactional Memory
- General:
  - Where parallel hardware is headed
  - Where parallel software is headed
  - Parallel programming languages
- Sources for today's lecture
  - Transactional Coherence and Consistency, ASPLOS 2004, Stanford University
  - Vivek Sarkar, Rice University

12/03/09

Transactional Memory: Motivation

- Multithreaded programming requires:
  - Synchronization through barriers, condition variables, etc.
  - Shared variable access control through locks . . .
- Locks are inherently difficult to use
  - Locking design must balance performance *and correctness* *Coarse-grain locking: Lock contention Fine-grain locking: Extra overhead, more error-prone*
  - *Must be careful to avoid deadlocks or races in locking*
  - *Must not leave anything shared unprotected, or program may fail*
- *Parallel performance tuning is unintuitive*
  - *Performance bottlenecks appear through low level events Such as: false sharing, coherence misses, …*
- *Is there a simpler model with good performance?*

12/03/09

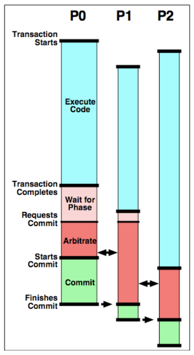
Using Transactions (Specifically TCC)

- Concept: Execute *transactions all of the time*
- *Programmer-defined groups of instructions within a program*
  End/Begin Transaction   Start Buffering Results
  Instruction #1
  Instruction #2 . . .
  End/Begin Transaction   Commit Results Now (+ Start New Transaction)
- *Can only "commit" machine state at the end of each transaction*
  - ***To Hardware: Processors update state atomically only at a coarse granularity***
  - ***To Programmer: Transactions encapsulate and replace locked "critical regions"***
- ***Transactions run in a continuous cycle . . .***

12/03/09

Transaction (TCC) Cycle

- Speculatively execute code and buffer
- Wait for commit permission
  - "Phase" provides commit ordering, if necessary
  - Imposes programmer-requested order on commits
  - Arbitrate with other CPUs
- Commit stores together, as a block
  - Provides a well-defined write ordering
  - To other processors, *all instructions within a transaction "appear" to execute atomically at transaction commit time*
  - *Provides "sequential" illusion to programmers Often eases parallelization of code*
  - *Latency-tolerant, but requires high bandwidth*
- *And repeat!*



12/03/09

A Parallelization Example

- Simple histogram example
  - Counts frequency of 0–100% scores in a data array
  - Unmodified, runs as a single large transaction (1 sequential code region)

```
int* data = load_data();
int i, buckets[101];
for (i = 0; i < 1000; i++) {
   buckets[data[i]]++;
}
print_buckets(buckets);
```

12/03/09

A Parallelization Example

- t_for transactional loop
  - Runs as 1002 transactions (1 sequential + 1000 parallel, ordered + 1 sequential)
  - Maintains sequential semantics of the original loop

```
int* data = load_data();
int i, buckets[101];
t_for (i = 0; i < 1000; i++) {
    buckets[data[i]]++;
}
print_buckets(buckets);
```

12/03/09

Conventional Parallelization of example

- Conventional parallelization requires explicit locking
  - Programmer must manually define the required locks
  - Programmer must manually mark critical regions Even more complex if multiple locks must be acquired at once
  - Completely *eliminated with TCC!*

```
int* data = load_data();  int i, buckets[101];
LOCK_TYPE bucketLock[101];
for (i = 0; i < 101; i++)    LOCK_INIT(bucketLock[i]);
for (i = 0; i < 1000; i++) {
   LOCK(bucketLock[data[i]]);
   buckets[data[i]]++;
  UNLOCK(bucketLock[data[i]]);
 }
print_buckets(buckets);
```

12/03/09

Other Concepts: Coherence and Fault Tolerance

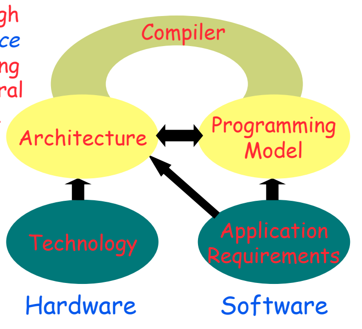
- Main idea:
  - Convenience of coarse-grain reasoning about parallelism and data sharing
  - Hardware/software takes care of synchronization details
  - Well-suited to code with heavy use of locking
- If two transactions try to commit the same data?
- If a transaction fails to complete?

12/03/09

My Research in this Space

Compiler-based optimization and auto-tuning

Achieve high *performance* by exploiting architectural features ...



... while freeing programmers from managing low-level details (***productivity***).

A Looming Software Crisis?

- Architectures are getting increasingly complex
  - Multiple cores, deep memory hierarchies, software-controlled storage, shared resources, SIMD compute engines, heterogeneity, ...
- Performance optimization is getting more important
  - Today's sequential and parallel applications ***may not*** be faster on tomorrow's architectures.
  - Especially if you want to add new capability!
  - Managing ***data locality*** even more important than parallelism.

Complexity!

Exascale Software Challenges

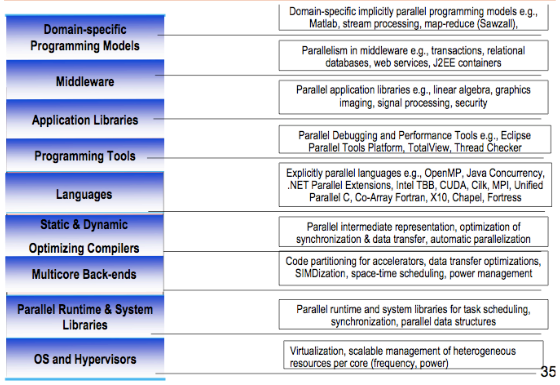
- Exascale architectures will be fundamentally different
  - Power management THE issue
  - Memory reduction to .01 bytes/flop
  - Hierarchical, heterogeneous
- Basic rethinking of the software "stack"
  - Ability to express and manage locality and parallelism for ~billion threads will require fundamental change
  - Support applications that are forward scalable and portable
  - Managing power (although locality helps there) and resilience requirements

Sarkar, Harrod and Snavely, "Software Challenges in Extreme Scale Systems," SciDAC 2009, June, 2009. Summary of results from a DARPA study entitled, "Exascale Software Study," (see http://users.ece.gatech.edu/%7Emrichard/ExascaleComputingStudyReports/ECS_reports.htm)

Motivation: Lessons at the Extreme End

- HPC programmers are more willing than most to suffer to get good performance
  - But pain is growing with each new architecture
  - And application base is expanding (*e.g.,* dynamic, graph-based applications)
- Government funding inadequate to make these systems useable
- Therefore, best hope is to leverage commodity solutions
  - Also, an interesting and fertile area of research lies in this intersection

Parallel Software Infrastructure Challenges



Domain-specific Programming Models	Domain-specific implicitly parallel programming models e.g., Matlab, stream processing, map-reduce (Sawzall),
Middleware	Parallelism in middleware e.g., transactions, relational databases, web services, J2EE containers
Application Libraries	Parallel application libraries e.g., linear algebra, graphics imaging, signal processing, security
Programming Tools	Parallel Debugging and Performance Tools e.g., Eclipse Parallel Tools Platform, TotalView, Thread Checker
Languages	Explicitly parallel languages e.g., OpenMP, Java Concurrency, .NET Parallel Extensions, Intel TBB, CUDA, Cilk, MPI, Unified Parallel C, Co-Array Fortran, X10, Chapel, Fortress
Static & Dynamic Optimizing Compilers	Parallel intermediate representation, optimization of synchronization & data transfer, automatic parallelization
Multicore Back-ends	Code partitioning for accelerators, data transfer optimizations, SIMDization, space-time scheduling, power management
Parallel Runtime & System Libraries	Parallel runtime and system libraries for task scheduling, synchronization, parallel data structures
OS and Hypervisors	Virtualization, scalable management of heterogeneous resources per core (frequency, power)

35

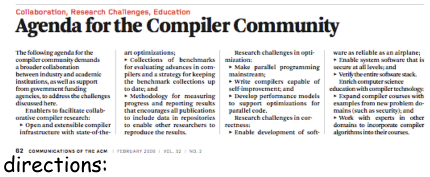
12/03/09

Slide source: Vivek Sarkar

Motivation: A Few Observations

- Overlap of requirements for petascale scientific computing and mainstream multi-core embedded and desktop computing.
- Many new and "commodity" application domains are similar to scientific computing.
  - Communication, speech, graphics and games, some cognitive algorithms, biomedical informatics **(& other "RMS" applications)**
- Importance of work with real applications **(who is your client?).**
  - Biomedical imaging, Molecular dynamics simulation, Nuclear fusion, Computational chemistry, speech recognition, knowledge discovery …

Where is compiler research going?



Collaboration, Research Challenges, Education

Agenda for the Compiler Community

The following agenda for the compiler community demands a broader collaboration between industry and academic institutions, as well as support from government funding agencies, to address the challenges discussed here.

Enablers to facilitate collaborative compiler research:

- Open and extensible compiler infrastructure with state-of-the-art optimizations;
- Collections of benchmarks for evaluating advances in compilers and a strategy for keeping the benchmark collections up to date; and
- Methodology for measuring progress and reporting results that encourages all publications to include data in repositories to enable other researchers to reproduce the results.

Research challenges in optimization:

- Make parallel programming mainstream;
- Write compilers capable of self-improvement; and
- Develop performance models to support optimizations for parallel code.

Research challenges in correctness:

- Enable development of software as reliable as an airplane;
- Enable system software that is secure at all levels; and
- Verify the entire software stack.

Enrich computer science education with compiler technology:

- Expand compiler courses with examples from new problem domains (such as security); and
- Work with experts in other domains to incorporate compiler algorithms into their courses.

62 COMMUNICATIONS OF THE ACM | FEBRUARY 2009 | VOL. 52 | NO. 2

- Main research directions:
  - Make parallel programming mainstream
  - Write compilers capable of self-improvement [autotuners]
  - Performance models to support optimizations for parallel code
  - Enable development of software as reliable as an airplane
  - Enable system software that is secure at all levels
  - Verify the entire software stack

Hall, Padua and Pingali, "Compiler Research: The Next Fifty Years," CACM, Feb. 2009. Results of an NSF Workshop entitled, "The Future of Compiler Research and Education," held at USC/ISI in Feb. 2007.

Autotuning Research Themes

search

- Compiler-based performance tuning
  - Use vast compute & storage resources to improve performance
  - Enumerate options, generate code, try, measure, record (conceptually)

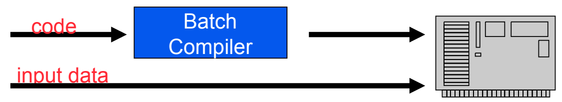
structure

- Optimization and performance tuning built from modular, understandable chunks
  - Easier to bring up on new platforms
  - Facilitates collaboration, moving the community forward

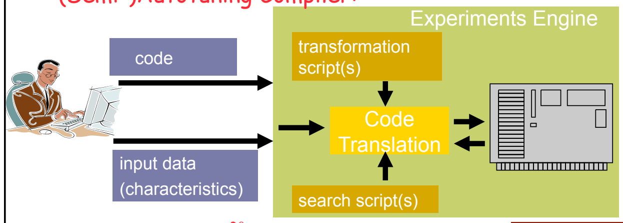
A Systematic, Principled Approach!

Recent Research: Auto-Tuning "Compiler"

Traditional view:

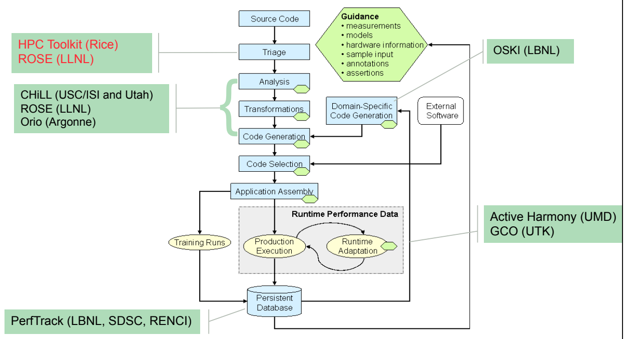


(Semi-)Autotuning Compiler:

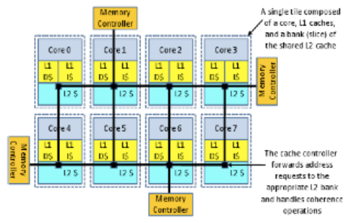


20

Collaborative Autotuning in PERI



Future Directions: New Architectures


Memory Controller
Core 0
Core 1
Core 2
Core 3
Core 4
Core 5
Core 6
Core 7
L1 D$
L1 I$
L2 $
A single tile composed of a core, L1 caches, and a bank (slice) of the shared L2 cache
The cache controller forwards address requests to the appropriate L2 bank and handles coherence operations

Non-uniform cache architectures (NUCA) and memory controllers:

•Multi-core architectures with distributed L2 cache

•Co-locate data & computation

•Manage data movement

New NSF Project: SHF:Small: Hardware/Software Management of Large Multi-Core Memory Hierarchies, Rajeev Balasubramonian (PI) and Mary Hall (co-PI), Sept. 2009-August 2012.

Future Directions: New Architectures

Nvidia Tesla system:
240 cores per chip,960 cores per unit, 32 units!

• CS6963: Parallel programming for GPUs
• Paper and poster at SAAHPC and other work from class under submission
• Automatic CUDA code generation from CHiLL

NVIDIA Recognizes University Of Utah As A Cuda Center Of Excellence *University of Utah is the Latest in a Growing List of Exceptional Schools Demonstrating Pioneering Work in Parallel* *(JULY 31, 2008—NVIDIA Corporation)*