

Architecture, Algorithms, and Programming Models

William Gropp



Logistics

- Classes Wednesday and Friday 2-3:15, SC 1131
 - ◆ Except when I'm on travel
 - Sept 3,5,10; 24
 - Oct 8
 - Nov 19,21
- Office hours: TBD, but stop in any time my door is open
- Format: Lectures; Presentations of papers by Students, and open discussion
 - ◆ Discussion of Papers
 - You should be reading papers in the area
 - Sign up and present a paper
- Evaluation: Homework + Presentation + Project (more on this later)



3

Goals of this course

- Algorithms don't perform the way many expect
 - ◆ Learn why - difference between idealized and real architecture
- Learn techniques to restructure algorithms and data structures to better
 - ◆ Some could be performed by compilers but aren't
 - ◆ More interesting are changes to the *data structures*
- Learn ways to create algorithms that are more suited to modern hardware
- Learn how programming models help or hinder the expression of efficient algorithms and what changes are on the horizon



2

What Will You Learn?

- Computer architecture relevant to understanding performance
 - ◆ Or, why(maybe) your codes do not perform as you expect
- Interaction of architecture with algorithms
 - ◆ Some performance loss is due to the complexity of the system
 - The algorithm does not need to change, but does need some help to be efficiently realized
 - ◆ Some performance loss is due to the mismatch between the hardware and the algorithm
 - The algorithm must be revised or replaced



4

What Will You Learn (con't)

- Strategies for developing both kinds of algorithmic enhancements
- And most important, how to think about performance
- Once you understand the interplay between algorithms and architecture, we'll ask how do programming languages influence/affect/guide the algorithms that are developed, evaluated, and used.
 - ◆ The programming language may add additional constraints to what is considered possible
 - ◆ What languages, tools, and language extensions are available to aid in algorithm and program development



5

Parallel Computing

- Some issues will be relevant to parallel computing
- Most of the initial discussion will be single thread performance or single node performance
 - ◆ Parallel computing architectures will be covered as needed
 - ◆ For reasons that will become clear, much of the programming language discussion will involve parallel computing



6

Project

- A term project is required for this course. Students will select an algorithm and analyze it with respect to one or more architectures and programming models. Students will make a proposal for a project, due October 17th. The final project report is due December 10th.
 - ◆ How does the algorithm fit the selected architecture(s)?
 - ◆ What is an appropriate performance model?
 - ◆ How well do the selected programming models support the implementation of the algorithm with performance?



7

Some Examples of Projects

- For one of the following algorithms, analyze how well the natural implementation in several programming models works on one or more architectures:
 - ◆ Dense matrix-matrix multiply
 - ◆ Sparse matrix-vector multiply
 - ◆ Sparse matrix-matrix multiply
 - ◆ Stencil computations
 - ◆ Distributed hash
 - ◆ Map Reduce



8

More Examples of Projects

- For one of the following algorithms, consider alternatives that can solve the same problem to a similar degree of accuracy but with a better fit to one or more selected architectures
 - ◆ Global FFT (e.g., replaced with a different basis with better locality)
 - ◆ Conjugate gradient for solving linear systems on parallel machines (e.g., allow overlap of reduction operations with other operations)
 - ◆ For one algorithm, focus on programming model support for achieving performance with that algorithm while retaining the ability to maintain and modify the implementation.



9

More on Projects

- Feel free to propose something on which you are working as the project
 - ◆ The best projects are the ones in which you have the most invested
 - ◆ The purpose of the project proposal is to ensure that the project is not too large and not too small



10

Paper Presentations

- Research papers provide a great way to explore a topic in depth
- Each student will present one paper, covering
 - ◆ Problem that the paper address
 - ◆ Solution approach
 - ◆ Major Lessons
 - ◆ Followup (weaknesses, changes since paper written, extensions)
 - ◆ Presentation length approximately 15 minutes, followed by questions and discussion
- Feel free to nominate a paper or choose from the following list. Links on the class web site
<http://www.cs.uiuc.edu/homes/wgropp/cs598/> :



11

Architecture

- There are several good books on both serial and parallel computer architecture, along with numerous book chapters and online. The papers here were written for algorithm developers and talk about the realities of architecture in that context.
 - ◆ Bit reversal on uniprocessors (Alan Karp, SIAM Review, 1996)
 - ◆ Experimental Analysis of Algorithms (Catherine McGeoch, Notices of the American Mathematical Society, March 2001)
 - ◆ Reflections on the Memory Wall (Sally McKee, ACM Conference on Computing Frontiers, 2004)



12

Algorithms

- These papers consider some aspects of algorithms that must perform well on real hardware, as opposed to idealized hardware.
 - ◆ The Landscape of Parallel Computing Research: A View from Berkeley, particularly Section 3. (Tech report EECS-2006-183, 2006)
 - ◆ Cache-Oblivious Algorithms (Algorithms for Memory Hierarchies, LNCS 2625, pages 193-212, Springer Verlag)
 - ◆ An analysis of sparse matrix-vector multiply is presented in this paper: Achieving high sustained performance in an unstructured mesh CFD application (SC1999)



13

Programming Models

- Many programming models have been proposed to address various goals. These papers look at some of the issues in programming for high performance while retaining productivity.
 - ◆ A comparative study of the NAS MG benchmark across parallel languages and architectures SC 2000
 - ◆ Programmability of the HPCS Languages: A Case Study with a Quantum Chemistry Kernel
 - ◆ Automated Transformation for Performance-Critical Kernels ACM SIGPLAN LCSD'07, Yi and Whaley



14

Example

- Lets look at a simple example
- Matrix-matrix multiply
 - ◆ Classic example, often used in discussion of compiler optimizations
 - ◆ Core of the "HPLinpack" benchmark
 - ◆ Simple to express: In Fortran,
do i=1, n
 do j=1, n
 c(i,j) = 0
 do k=1, n
 c(i,j) = c(i,j) + a(i,k) * b(k,j)



15

Performance Estimate

- How fast should this run?
 - ◆ Standard complexity analysis in numerical analysis counts floating point operations
 - ◆ Our matrix-matrix multiply algorithm has $2n^3$ floating point operations
 - 3 nested loops, each with n iterations
 - 1 multiply, 1 add in each inner iteration
 - ◆ For $n=100$, 2×10^6 operations, or about 1 msec on a 2GHz processor :)
 - ◆ For $n=1000$, 2×10^9 operations, or about 1 sec



16

The Reality

- N=100
 - ◆ 1818 MF (1.1ms)
- N=1000
 - ◆ 335 MF (6s)
- What this tells us:
 - ◆ Obvious expression of algorithms are not transformed into leading performance.
- Try it yourself! Code on web site



17

Questions and Comments

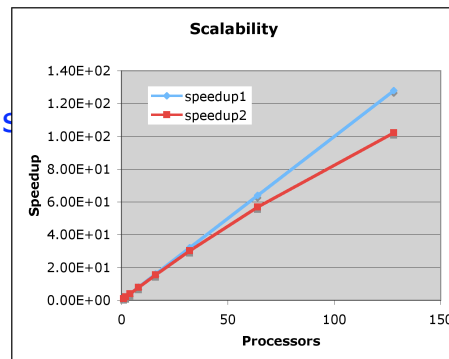
- Why does the algorithm run slowly at large sizes?
- Why doesn't the compiler do a better job?
- What about other algorithms such as Strassen's algorithm?



18

Another Example: Scalability

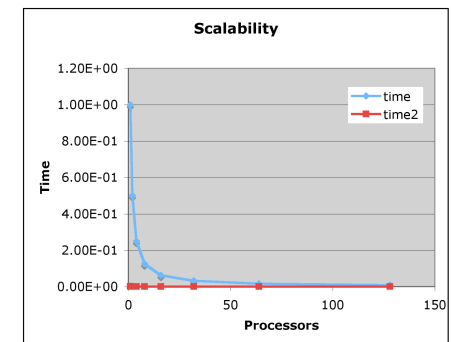
- Two algorithms run in parallel
- Perfect speedup is # of processors
- Algorithm 1 is (nearly) perfect
- Algorithm 2 is fading
- Which is best?



19

Scalability and Time

- Algorithm 1 has very poor uniprocessor performance
- Algorithm 2 has very good uniprocessor performance
- Not a (too) contrived case - such examples have appeared in papers and proposals



20

Course Outline

1. Introduction
 1. Course logistics/schedule/projects
 2. Overview of course goals and topics covered
2. Example of the Issues
 1. Matrix-vector multiply
 2. Matrix-matrix multiply
3. Review of Computer Architecture for Performance Understanding
 1. Memory Hierarchy
 2. Instruction Execution
 3. Parallel Processing
4. Realistic Performance Models
 1. Applied to examples
 2. Use of upper and lower bounds on performance
5. Parallel Computer Architecture for Performance Understanding
 1. Change to memory hierarchy and model; memory consistency
 2. Latency
 3. Concurrency and Little's Law
 4. Interconnect
 5. Complexity Models



21

Course Outline (2)

6. Adapting Algorithms to Architecture
 1. Changing order of evaluation (and hence access)
7. Adapting Data Structures to Architecture
 1. Padding
 2. Alignment
 3. Recursive mappings
8. Interlude: Performance Measurement
 1. Measurement techniques and hazards
 2. Statistical techniques
 3. Reproducibility
9. Algorithmic Strategies
 1. The Cache Oblivious approach
 2. Autotuning and families of algorithms
10. Latency Tolerant Algorithms
 1. Limits on scalability
 2. Solution 1: Adding hardware
 3. Solution 2: Removing synchronization
 4. Solution 3: Changing the problem



22

Course Outline (3)

11. Programming Language Support for Algorithms
 1. Historical overview
 2. Partitioned Global Address Space languages
 3. The DARPA High Productivity Computing Systems languages
12. Programming Models for SMPs
 1. OpenMP
 2. Threads
13. Evaluations of Some Programming Models
14. Special Purpose Languages
 1. Domain Specific languages
 2. Performance Transformation languages
15. Multicore issues for Algorithms
 1. High local bandwidth
 2. Low (relative) off-chip bandwidth
 3. Threads for latency hiding
16. Challenges at Extreme Scale
 1. Concurrency
 2. Power



23

Algorithms

- What is an algorithm?
 - ◆ A set of instructions to perform a task
- How do we evaluate an algorithm?
 - ◆ Correctness
 - ◆ Accuracy
 - Not an absolute
 - ◆ Efficiency
 - Relative to current and future machines
- How do we measure efficiency?
 - ◆ Often by counting floating point operations
 - ◆ Compare to "peak performance"



24

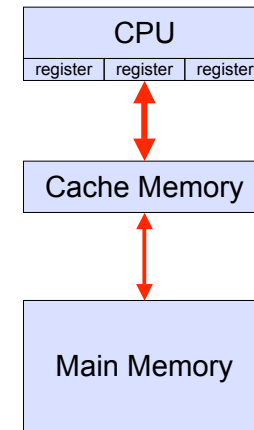
Real and Idealized Computer Architectures

- Any algorithm assumes an idealized architecture
 - Common choice:
 - Floating point work costs time
 - Data movement is free
 - Real systems:
 - Floating point is free (fully overlapped with other operations)
 - Data movement costs time...a lot of time
- Classical complexity analysis for numerical algorithms is *no longer correct* (more precisely, no longer *relevant*)
 - Known since at least BLAS2 and BLAS3



Simplified Computer Architecture

- Main memory contains the program data
- Cache memory contains a copy of the main memory data
 - Cache is faster but consumes more space and power
 - Cache items accessed by their address in main memory
- Registers contain working data only
 - Modern CPUs perform most or all operations only on data in register

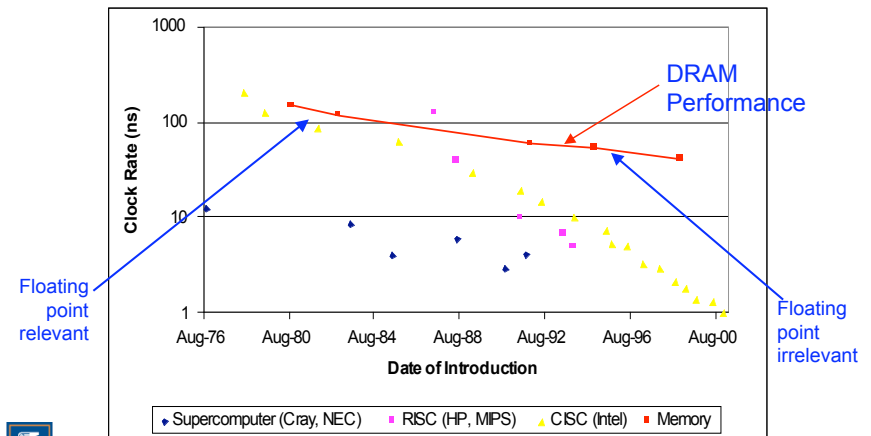


Simplified Performance Models

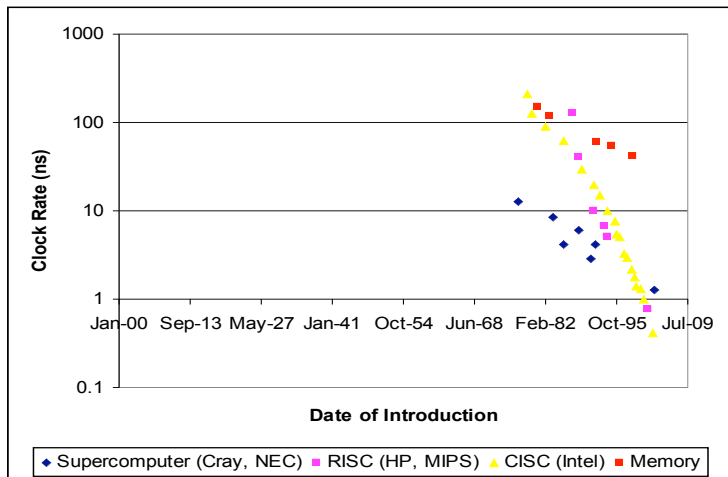
- The processor runs in discrete steps, controlled by a *clock*
 - Typical clock rates exceed 2 GHz
- Few operations complete in a single clock period
- One way to describe the operations is with two parameters:
 - Latency: the time it takes to complete a single operation (number of clock ticks or cycles)
 - Bandwidth: the steady-state rate at which operations can be completed
- Example: Memory access
 - A read of a single word from main memory may take 450+ cycles (IBM POWER6)
 - Bandwidth to main memory is GB/s (>4 on POWER6)



CPU and Memory Performance



More Recent Results (or meet Y2K, or *correctness first*)



29

Trends in Computer Architecture II

- Clock speeds will continue to increase
 - ◆ The rate of clock rate increase has increased recently ☺
 - ◆ Light travels 3 cm (in a vacuum) in one cycle of a 10 GHz clock
 - CPU chips won't be causally connected within a single clock cycle, i.e., a signal will not cross the chip in a single clock cycle
 - Processors will be parallel!

31

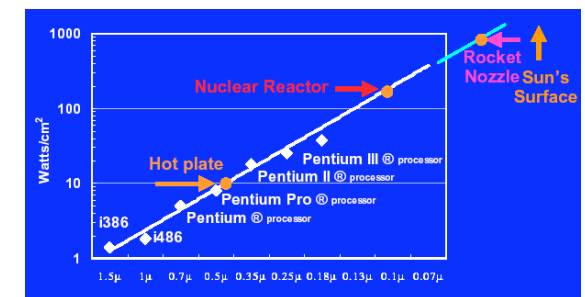
Trends in Computer Architecture I

- Latency to memory will continue to grow relative to CPU speed
 - ◆ Latency hiding techniques require finding increasing amounts of independent work: Little's law implies
 - Number of concurrent memory references = Latency * rate
 - For 1 reference per cycle, this is already 100-1000 concurrent references

30

Trends in Computer Architecture III

- Power dissipation problems will force more changes
 - ◆ Current trends imply chips with energy densities greater than a nuclear reactor
 - ◆ Already a problem: The current issue of consumer reports looks at the likelihood of getting a serious burn from your laptop!
 - ◆ Will force new ways to get performance, such as extensive parallelism



32