Graduate Students ...

look for other sources of information

make decisions, because all research problems are under-specified

evaluate their own work

write, write, write

question everything

http://www.cs.unc.edu/~azuma/hitch4.html

CS/ECE 560: Foundations of Fine-Grained Parallelism

Instructor:	**Michelle Strout** mstrout@cs.colostate.edu Computer Science Building 342 Office hours: Wednesday 2-3 by appointment
TA:	**Manaf Gharaibe** gharaibe@cs.colostate.edu
Questions:	send email to Michelle and Manaf cs560@cs.colostate.edu
URL:	http://www.cs.colostate.edu/~cs560/Spring2012

Plan for Today

Introductions
- Name, year in school, masters or phd student
- Parallel programming experience

Administrivia
- Course details
- How to succeed in this course

Motivation
- Why study automatic performance improving transformations including parallelization?
- Look at some sample program optimizations and assorted issues

Course Outline

Concepts

Expectations

Next Time

Administrivia

Course website (http://www.cs.colostate.edu/~cs560/Spring2012)
- Progress page will have everything (schedule, notes, videos, assignments)
- Assignments are also listed on assignments page

RamCT
- Grade book
- Submitting assignments (also send to cs560@cs.colostate.edu)
- Discussions
- Chat for online students during office hours
- Mail will be sent through this interface, check your email address

Succeeding in this course
- Check the progress and assignments pages on the website every day.
- Spend at least 1-2 hours per day outside of class doing reading assignments and homeworks.
- Embrace the RTFM (read the fine manual) concept.

Class Objectives

Short Term: Learn how to write highly tuned code for emerging target architectures such as GPU and multicore.

Medium Term: Learn the general principles involved as well as the skills so you can handle the next new architectural paradigm.

Long Term:

- Learn how program parallelization and optimization is done automatically.
- Evaluate existing automation tools and identify gaps and future research questions.

Class Approach (Syllabus)

11 homework assignments (35%)

- Each assignment will be written up using latex (a latex template will be provided).
- The lowest homework score will be dropped.
- The homeworks will include small programming assignments, evaluation of performance, and other questions.

Online Quizzes and Discussions (5%)

- Will ask questions about the reading material.
- Will discuss important concepts with the whole class.

Project (35%)

- Includes a proposal, intermediate report, final paper, and poster.
- Will be working with a new code and/or new automation tool.

Take home midterm (25%)

Motivation

Why parallelism?

- The free ride of increasing clock rates is over. Parallelism will be the main way to improve performance.
- The power wall has pushed computer architecture to multiple cores.

Example parallel machines

- **Multicore,** where there are more than one processor, or cores, per chip
  - vege machines in the department, two sockets with 4 cores each, shared memory between all 8
  - Cray XT6m with 1248 cores, each node supports shared memory for 24 cores
- **GPU (Graphics Processing Unit),** SW managed caching, fine-grained parallelism
  - NVIDIA cards in room 325 machines
  - Tesla GPU card in bananas, coconuts, apples, and oranges

Yes, but can it help me get a job?

Summer internships in past 5 years

- LLNL with ROSE compiler (2)
- Cray with Chapel group
- NCAR (5)
- Intel working on hand-parallelization based on compiler feedback

Intel, Microsoft, etc. are all very interested in parallel computing because parallelism will need to be managed in software.

Government labs often looking for research programmers who know about parallelism and automating parallelism.

Many new parallel programming models are in development and use. For example, my group has open graduate research positions.

Ultimate Goal is to Automate Parallelization

Automate parallelization and performance optimization in a compiler

- What is the programming model for the programmer?
- What is the intermediate representation (IR) to enable parallelism and data reuse?
- How can we generate efficient code from the IR?

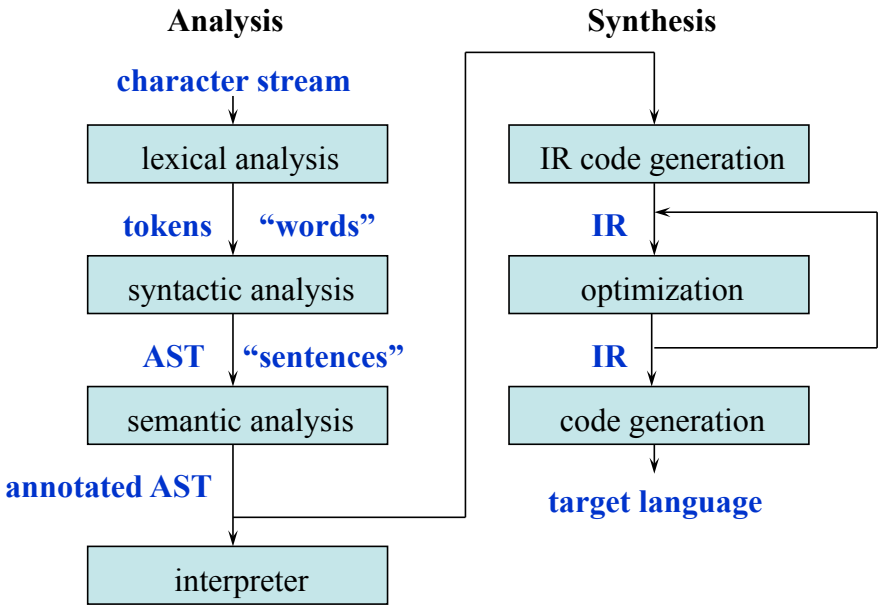
Run-time Library Support

- The compiler cannot do it all.
- How can the compiler provide information to the run-time system?
- What algorithms should be used to efficiently schedule at runtime?

Course Focus

- The intermediate representation for the compiler.
- Will look at the interaction between the IR and runtime for sparse computations such as molecular dynamics simulations.

Structure of a Typical Compiler



Isn't Compilation/Automated Parallelism A Solved Problem?

"Optimization for scalar machines is a problem that was solved ten years ago"
-- David Kuck, 1990

Machines keep changing
- New features present new problems (*e.g.,* MMX, SSE, GPU)
- Changing costs lead to different concerns (*e.g.,* loads)

Languages keep changing
- Wacky ideas (*e.g.,* OOP and GC) have gone mainstrean
- Parallelism is now mainstream

Applications keep changing
- Interactive, real-time, mobile, secure

Some apps always want more
- More precision
- Simulate larger systems

Goals keep changing
- Correctness
- **Run-time performance**
- Code size
- Compile-time performance
- Power
- Security

Modern View of Compilers

Analysis and performance transformations are useful everywhere
- Analysis and transformations can be performed at run time and link time, not just at "compile time"
- Optimization can be applied to OS as well as applications
- Analysis can be used to improve security by finding bugs
- Analysis can be used in software engineering
  - Program understanding, reverse engineering, refactoring
  - Debugging and testing
- New parallel programming abstractions can enable the automation of parallelization and performance transformations
- Increased interaction between hardware, programmer, and compilers can improve performance
- Bottom line
  - Analysis and transformation play essential roles in computer systems
  - Computation important ⇒ *understanding* computation important

Some Exciting Current Research in PLDI and PPOPP

Premier conferences for dissemination of compiler, programming languages, and parallelism research

–PLDI (Programming Language Design and Implementation)

–PPOPP (Principles and Practice of Parallel Programming)

Parallel Programming Languages

- Most common: C/C++ or Fortran 90+ combined with MPI and/or OpenMP
- Up and coming languages and programming models
  - DARPA HPCS languages: Cray's Chapel, IBM's X10, Sun's Fortress
  - PGAS languages like UPC and CoArray FORTRAN
  - CUDA and OpenCL for programming GPUs
  - TBB from Intel
  - Concurrent Collections: Intel and Rice University collaboration
  - AlphaZ for expressing programs as equations, CSU project

What we are going to do in 560

Learn how to write highly tuned code for emerging target architectures such as GPU and multicore.

- Parallelize and apply performance transformations to some scientific computing benchmarks (HW0 uses protein string matching).
- Evaluate and present performance results.

Learn the general principles involved as well as the skills.

- What is the fastest a computation can execute on a given architecture?
- Where is the parallelism and data reuse in a computation?

Study the Automation of Parallelization and Performance Optimization

- Learn frameworks for representing computations and their implementation details separately (Polyhedral model, Sparse Polyhedral Framework, etc.).
- Evaluate existing automation tools (Omega, AlphaZ, and Pluto) by comparing their results with hand-optimized code.

Types of Performance Optimizations

Definition

- A *performance optimization* is a transformation that is expected to improve the program in some way; often consists of *analysis* and *transformation*
  e.g., decreasing the execution time or decreasing memory requirements

Machine-independent optimizations

- Eliminate redundant computation
- Move computation to less frequently executed place
- Specialize some general purpose code
- Remove useless code
- Identify parallelism and data reuse

Types of Optimizations (cont)

Machine-dependent optimizations

- Replace costly operation with cheaper one
- Replace sequence of operations with cheaper one
- Hide latency
- Improve data locality
- Exploit machine parallelism
- Reduce power consumption

Enabling transformations

- Expose opportunities for other optimizations
- Help structure optimizations

Sample Optimizations

Loop-invariant code motion

– *e.g.,*

```
for i = 1 to 10 do
    x = 3;
    ...
```

→

```
x = 3;
for i = 1 to 10 do
    ...
```

Loop unrolling

– *e.g.,*

```
for i = 1 to 10 do
    a[i] = a[i] + 1;
```

→

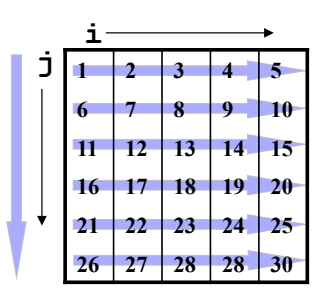
```
for i = 1 to 10 by 2 do
    a[i] = a[i] + 1;
    a[i+1] = a[i+1] + 1;
```


More examples: Loop Permutation for Improved Locality

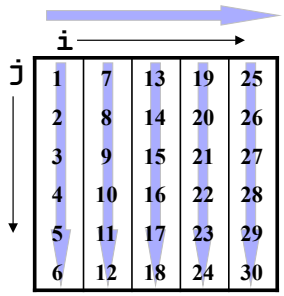
Sample code: Assume Fortran's Column Major Order array layout

```
do j = 1,6
 do i = 1,5
   A(j,i) = A(j,i)+1
 enddo
enddo
```

```
do i = 1,5
  do j = 1,6
   A(j,i) = A(j,i)+1
  enddo
enddo
```



poor cache locality

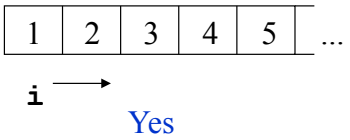


good cache locality

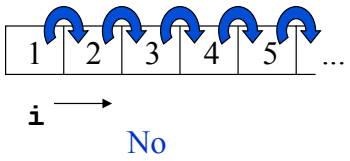
More examples: Parallelization

Can we parallelize the following loops?

```
do i = 1,100
 A(i) = A(i)+1
enddo
```




```
do i = 1,100
 A(i) = A(i-1)+1
enddo
```



Is an Parallelization or Optimization Worthwhile?

Criteria for evaluating optimizations

- Safety: does it preserve behavior?
- Profitability: does it actually improve the code?
- Opportunity: is it widely applicable?
- Cost (compilation time): can it be practically performed?
- Cost (complexity): can it be practically implemented?

How do we automate the process?

Specifying loop transformations and parallelization in frameworks

- Represent the computation within an integer tuple space

Representing Computations with Iteration Spaces

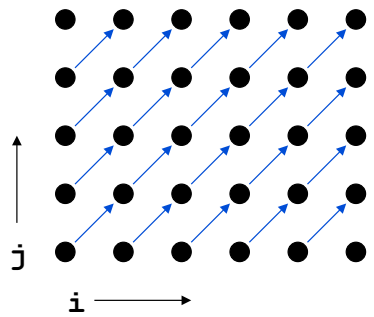
Idea

- Explicitly represent the iterations of a loop nest

Example

```
do i = 1,6
  do j = 1,5
     A(i,j) = A(i-1,j-1)+1
  enddo
enddo
```

Iteration Space



Iteration Space

- A set of integer tuples that represents the iterations of a loop
- Can visualize the dependences in an iteration space

How do we automate the process?

Specifying loop transformations and parallelization in frameworks

- Represent the computation within an integer tuple space
- Analyze and represent dependences
- Specify or determine schedules and storage mappings that improve performance
- Generate code given the computation, schedule, and storage mapping specifications

Tradeoffs

- The more restrictive the model, the easier it is to automate analysis, schedule selection, and storage mapping.
- The model becomes more complex as its capabilities grow: imperfectly nested loops, more transformations, etc.

Course Outline I

Review

- Timing benchmarks and graphing results
- Parallelization with OpenMP
- Big-O, functions, set theoretic notation
- Linear algebra concepts such as matrix multiply

Working with Applications

- Touchstone problems for this class
- Categories of applications based on The Berkeley View

Computer Architectures and Parallelism

- Multiple levels of parallelism and memory
- What is the best performance possible for a particular application?

Performance Analysis, Metrics, and Models

Loop transformations and parallelization

- Applying them by hand
- When are they useful

Course Outline II

Frameworks for Automating Loop Transformations

- Frameworks: unimodular, polyhedral, presburger, and sparse polyhedral
- Represent computations, represent loop transformations, test the legality of transformations, and compose transformations
- Generate efficient code

Tiling: Aggregate computation to reduce overhead

Run-time Reordering Transformations: Inspector/Executor strategies

Full Automation

- Set and relation manipulation
- Scheduling and storage mapping algorithms
- Code generation algorithms

Current State of the Tools

- Tools: omega, AlphaZ, and Pluto
- What are the implementation gaps?
- What are the limitations and future research questions?

Concepts

Parallelism is critical to improving performance in current and future machines.

Compilation

- Stages of compilation
- Automating parallelization and performance optimization in compiler is the ideal.

Example Program Optimizations

- Loop-invariant code motion
- Induction variable elimination
- Loop unrolling, permutation, and parallelization

Approach for Automating Program Parallelization and Optimization

- Represent computations with iteration spaces
- Schedule the iteration space

Expectations

DO

- Expect to spend more time on this course than on a challenging undergraduate course.
- Write more than one draft for your project and homework reports. Spelling mistakes will be penalized. Correct grammar is also expected (for help use Word or even better the Writing Center).
- Make decisions when the homework is underspecified. Describe the reasoning for your decisions in the homework report.
- Read assigned reading. Much of it will take more than two readings and anything in the readings might be on the midterm or quizzes.
- Break your homeworks and projects into small pieces, thus making things easier to make progress and debug.
- Ask questions and come to (or call into) office hours sooner rather than later.

Thinking is important and should be done frequently.

Next Time

Reading

- So long and thanks for the PhD
- OpenMP tutorial through C/C++ **for** directive example
- Twelve Ways to Fool the Masses … by David Bailey

Homework

- Send a request for a Cray account
- Start working on HW0

Lecture

- Review of Parallelism and OpenMP