ME964
High Performance Computing for Engineering Applications

Spring 2012

Dan Negrut
Associate Professor
Department of Mechanical Engineering
University of Wisconsin, Madison

January 24, 2012

"I think there is a world market for maybe five computers."
T. J. Watson, chairman of IBM, 1943.

- Purpose of today's lecture
  - Get a 30,000 perspective on this class and understand whether this is a class worth taking

- What we will cover today
  - Course logistics
  - Brief overview of syllabus
  - Motivation and central themes of this class
  - Start quick overview of C programming language

Instructor: Dan Negrut

- Polytechnic Institute of Bucharest, Romania
  - B.S. – Aerospace Engineering (1992)
- The University of Iowa
  - Ph.D. – Mechanical Engineering (1998)
- MSC.Software
  - Product Development Engineer 1998-2005
- The University of Michigan
  - Adjunct Assistant Professor, Dept. of Mathematics (2004)
- Division of Mathematics and Computer Science, Argonne National Laboratory
  - Visiting Scientist 2004-2005, 2006, 2010
- The University of Wisconsin-Madison, Joined in Nov. 2005
  - Research Focus: Computational Dynamics (Dynamics of Multi-body Systems)
  - Established the Simulation-Based Engineering Lab (http://sbel.wisc.edu)

Acknowledgements

- Students helping with this class
  - Toby Heyn [your TA]
  - Andrew Seidl [takes care of hardware]
  - Hammad Mazhar [help with CUDA & thrust]

- College of Engineering - financial support for video recording

- NVIDIA & US Army ARO:
  - Financial support to build Euler, CPU/GPU cluster used in this class

Good to know…

Time	9:30 Tu & Th
Location	1163ME
Office	2035ME
Phone	608 890-0914
E-Mail	negrut@engr.wisc.edu
Course Webpage	http://sbel.wisc.edu/Courses/ME964/2012/index.htm
Grades reported at:	learnuw.wisc.edu
ME964 Forum:	http://sbel.wisc.edu/Forum/index.php?board=3.0

ME 964 Spring 2012

- Office Hours:
  - Monday 2 – 4 PM
  - Wednesday 2 – 4 PM

- Call or email to arrange for meetings outside office hours

- Walk-ins are fine as long as they are in the afternoon

- TA: Toby Heyn
  - Desk in room 2042ME
  - heyn@wisc.edu

References

- No textbook is required, but there are some recommended ones:

- **<u>Highly recommended</u>**
- NVIDIA CUDA C Programming Guide V4.0, 2012 :
  - http://developer.download.nvidia.com/compute/cuda/4_0/toolkit/docs/CUDA_C_Programming_Guide.pdf

- Jason Sanders and Edward Kandrot: CUDA by Example: An Introduction to General-Purpose GPU Programming, Addison-Wesley Professional, 2010 (on reserve, Wendt Lib.)
- David B. Kirk and Wen-mei W. Hwu: Programming Massively Parallel Processors: A Hands-on Approach, Morgan Kaufmann, 2010 (on reserve, Wendt Lib.)
- Peter Pacheco: An Introduction to Parallel Programming, Morgan Kaufmann, 2011
- B. Kernighan and D. Ritchie, The C Programming Language
- B. Stroustrup, The C++ Programming Language, Third Edition

References [Cntd.]

- **Further reading**

- Wen-mei W. Hwu (editor), GPU Gems 4, 2011, Addison Wesley
- Rob Farber: CUDA Application Design and Development, Morgan Kaufmann 2011
- H. Nguyen (editor), GPU Gems 3, Addison Wesley, 2007 (on reserve, Wendt Lib.)
- Peter Pacheco: Parallel Programming with MPI, Morgan Kaufmann, 1996
- T. Mattson, et al.: Patterns for Parallel Programming, Addison Wesley, 2005
- Michael J. Quinn: Parallel Programming in C with MPI and OpenMP, McGraw Hill, 2003
- A. Grama, A. Gupta, G. Karypis, V. Kumar: Introduction to Parallel Computing, Addison Wesley, 2003

Course Related Information

- Handouts will be printed out and provided before each lecture
- Lecture slides (PPT and PDF) will be made available online at class website
- Video streaming of class anticipated to be available on the same day at
  - http://mediasite.engr.wisc.edu/Mediasite/Catalog/pages/catalog.aspx?catalogId=31c0b7c4-3a0f-410b-bacf-0c238380112f&folderId=96ee9eab-32a4-4321-8b45-6eae85c267ef&rootDynamicFolderId=e5b4a945-c68f-45b2-9eb7-b2512f5122cd
- Grades will be maintained online at Learn@UW
- Syllabus will be updated as we go
  - It will contain info about
    - Topics we cover
    - Homework assignments
  - Available at the course website
    - http://sbel.wisc.edu/Courses/ME964/2012/index.htm

Grading

Homework	40%
Midterm Exam	10%
Midterm Project	20%
Final Project	25%
Course Participation	5%
Total	100%

NOTE:

- Score related questions (homework/exam) must be raised prior to next class after the homework/exam is returned.

Homework Policies

- About 12 HWs assigned
  - No late HW accepted
    - HW due at 11:59 PM on the due day

- Homework with lowest score will be dropped when computing final score

- Homework and projects should be handed in using Learn@UW dropbox
  - To get credit for your work the time-stamp should be prior to the assignment due time/date

Midterm Exam

- One midterm exam
- Scheduled during regular class hours
- Tentatively scheduled on **April 17**
- Doesn’t require use of a computer (it’s a pen and paper exam)
- It’s a “closed books” exam

Midterm Project

- Has to do with implementation of a parallel solution for solving a large *dense* system of equations
  - Size: as high as you can go
  - Implemented in CUDA
  - Focus on banded and full matrices

- Due on **April 12** at 11:59 PM

- Accounts for 20% of final grade

- Project is individual or produced by two-student teams

- Should contain a comparison of your parallel code with solvers that are available already in the Scientific Computing community
  - Intel MKL, LAPACK, etc,

- Should include profiling results and a weak scaling analysis

Final ~~Exam~~ Project

- Scheduled for May 17, 2:45 – 4:45 PM
  - There will be no final exam, rather a Final Project
  - The Final Project is due on May 16, at 11:59 PM
  - Two hour time slot used to have Final Project presentations
  - Additional presentation time slots will very likely be needed during finals’ week
    - I will come up with a way for you to select your time slot based on your availability during the finals’ week

Final ~~Exam~~ Project

- Final Project (accounts for 25% of final grade):
  - It is an individual project or produced by two-student teams
  - You choose a problem that suites your research or interests
  - You are encouraged to tackle a meaningful problem
    - Attempt to solve a useful problem rather than a problem that you are confident that you can solve
    - Projects that are not successful are ok, provided you aim high enough and demonstrate good work
    - Continuing the Midterm Project topic is ok (shifting focus on sparse systems)
  - Tentatively, work on Final Project will start on April 11 once you submit a project proposal

Class Participation

- Accounts for 5% of final grade. To earn the 5%, you must:
  - Contribute at least five meaningful posts on the class Forum
    - Forum is live at: http://sbel.wisc.edu/Forum/index.php?board=3.0
    - Forum meant to serve as a quick way to answer some of your questions by instructor and other ME964 colleagues
    - You should get an email with login info shortly (today or tomorrow)

Scores and Grades

Score	Grade
92-100	A
86-91	AB
78-85	B
70-77	BC
60-69	C
50-59	D

- Grading will <u>not</u> be done on a curve
- Final score will be rounded to the nearest integer prior to having a letter assigned
  - Example:
    - 85.59 becomes AB
    - 85.27 becomes B

Prerequisites

- This is a high-level graduate class in a very fluid topic
- Familiarity with C is needed
  - You can probably be fine if you are a friend of Java
- Good programming skills are necessary
  - Understanding pointers
  - Being able to wrestle with a compile error on your own
  - Having used a debugger
  - Having used a profiler

Rules of Engagement

- You are encouraged to discuss assignments with other class students
  - Post and read posts on Forum
- Getting **verbal** advice and suggestions from anybody is fine
- copy/paste of non-trivial code is not acceptable
  - Non-trivial = more than a line or so
  - Includes reading someone else's code and then going off to write your own
- Use of third party libraries that directly implement the solution of a HW/Project is not acceptable unless explicitly asked to do so

A Word on Hardware…

- The course designed to leverage a dedicated CPU/GPU cluster
  - CPUs: Intel Xeon 5520, a quadcore chip (about 200 cores)
  - GPUs: NVIDIA TESLA C2050/C2070 and some GTX480 cards
    - 56 GPU cards available
    - Most of the GPU cards have 448 Scalar Processors
    - GPUs have between 1.5 to 6 GB global memory on the device

- Each student receives an individual account that will be used for
  - GPU computing
  - MPI-enabled parallel computing
  - OpenMP multi-core computing

- Advice: if possible, do all the programming on a local machine. Move to the cluster for "production" runs

A Word on Software…

- We will use Linux as our operating system of choice

- We'll use the following versions of libraries/releases:
  - CUDA: 4.1
  - MPI: 2.0
  - OpenMP: 3.0

- Reliance on makefiles generated with CMake, a build utility tool
  - Scripts will be available to you in order to facilitate compile/link/debug/profile process

- Integrated Development Environment: Eclipse, Indigo version

- We will use a suite of debugging and profiling tools
  - gdb: debugger under Linux
  - cuda-gdb: debugger for CUDA applications running on the GPU
  - NVIDIA Profiler

- Most of these tools are embedded in Eclipse

Staying in Touch…

- Please do not email me unless you have a personal problem
  - Examples:
    - Good: Schedule a one-on-one meeting outside office hours
    - Bad: Asking me clarifications on Problem 2 of the current assignment (this needs to be on the Forum)
    - Bad: telling me that you can't compile your code (this should also go to the Forum)

- Any course-related question should be posted on the Forum
  - I continuously monitor the Forum
  - If you can answer a Forum post, please do so (counts towards your 5% class participation and helps me as well)
  - Keeps all of us on the same page

- The forum is *very* useful

Course Emphasis

- There are multiple choices when it comes to implementing parallelism
  - PThreads, Intel's TBB, OpenMP, MPI, Ct, Cilk, CUDA, Etc.

- Course focuses on parallelism enabled by
  - The Graphics Processing Unit (GPU), mostly aimed at fine grain level parallelism
  - Message Passing Interface (MPI) standard, aimed at coarse grain parallelism
  - OpenMP standard, aimed both at fine and coarse level parallelism

- This is not going to be a hard course but it'll be a busy course
  - You'll easily understand all the material that we'll cover (no rocket science)
  - Probably the benefits come from
    - Bringing into focus within 15 weeks a collection of topics that are central to the idea of HPC
    - Getting to actually do what it takes to implement a HPC solution (write code, build, link, run, debug, profile)

Course Objectives

- Get familiar with today's High-Performance Computing (HPC) software and hardware
  - Usually "high-performance" implies execution on parallel architectures; i.e., architectures that have the potential to finish a run much faster than when the same application is executed sequentially
- Help you recognize applications/problems that can draw on HPC
- Help you gain basic skills that will help you map these applications onto a parallel computing hardware/software stack
- Introduce basic software design patterns for parallel computing

Course Objectives
[Cntd.]

- What I'll try to accomplish
  - Provide enough information for you to start writing software that can leverage parallel computing to hopefully reduce the amount of time required by your simulations to complete
- What I will not attempt to do
  - Investigate how to design new parallel computing languages or language features, compilers, how new hardware should be designed, etc.
- To summarize,
  - I'm a Mechanical Engineer, a consumer of parallel computing
  - I'm not interested in how to design parallel computing hardware or instruction architecture sets for parallel computing

High Performance Computing for Engineering Applications

Why This Title?

- Computer Science: ISA, Limits to Instruction Level Parallelism and Multithreading, Speculative Execution, Pipelining, Memory Hierarchy, Memory Transactions, Cache Coherence, etc.
  - Long story short: how should a processor be built?

- Electrical Engineering: how will we build the processor that the CS colleagues have in mind?

- This class: how to use the system built by electrical engineers who implemented the architecture devised by the CS colleagues
  - At the end of the day, in our research in Science/Engineering we'll be dealing with one of the seven dwarfs…

Phillip Colella's "Seven Dwarfs"

High-end simulation in the physical sciences = 7 numerical methods:

1. Structured Grids (including locally structured grids, e.g. Adaptive Mesh Refinement)
2. Unstructured Grids
3. Fast Fourier Transform
4. Dense Linear Algebra
5. Sparse Linear Algebra
6. Particles
7. Monte Carlo

- If add four more for embedded, covers all 41 EEMBC benchmarks
  8. Search/Sort
  9. Filter
  10. Combinational logic
  11. Finite State Machine
- Note: Data sizes (8 bit to 32 bit) and types (integer, character) differ, but algorithms the same

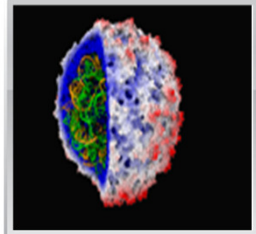
Slide from "Defining Software Requirements for Scientific Computing", Phillip Colella, 2004 .
Credit: D. Patterson

GPU Speed Improvements

[relative to sequential solution]

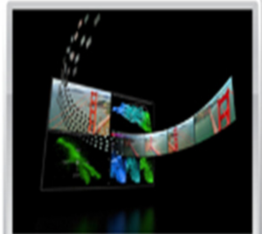
146X

Medical Imaging
U of Utah



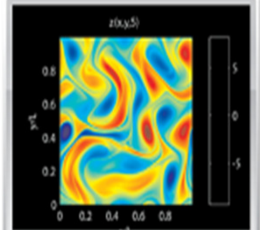
36X

Molecular Dynamics
U of Illinois, Urbana



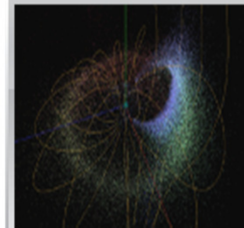
18X

Video Transcoding
Elemental Tech



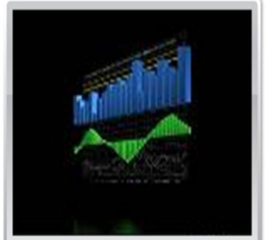
50X

MATLAB Computing
AccelerEyes



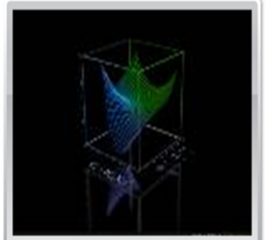
100X

Astrophysics
RIKEN



149X

Financial simulation
Oxford



47X

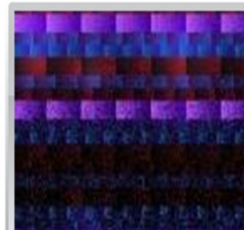
Linear Algebra
Universidad Jaime

20X

3D Ultrasound
Techniscan

130X

Quantum Chemistry
U of Illinois, Urbana



30X

Gene Sequencing
U of Maryland

Who Will Be the ME964 Student?

- 32 students enrolled coming from 13 UW departments
  - Biomedical Engineering, Chemical Engineering, Chemistry, Civil & Environmental, Computer Science, Electrical Engineering, Engineering Mechanics, Geography, Environment and Resources, Material Science, Mechanical Engineering, Medical Physics, Nuclear Engineering and Engineering Physics

- "High Performance Computing for Engineering Applications"
  - There is no need to have a prior Engineering degree
  - The course assumes a level of programming experience of a typical Engineer
  - For people in CS might not be challenging enough…

Auditing the Course

- Why auditing?
  - Augments your experience with this class
    - You get an account on the CPU/GPU cluster
    - You will be added to the email list
    - Can post questions on the forum

- How to register for auditing:
  - In order to audit a course, a student must first enroll in the course as usual. Then the student must request to audit the course online. (There is a tutorial available through the Office of the Registrar.) Finally, the student must save & print the form. Once they have obtained the necessary signatures, the form should be turned in to the Academic Dean in the Grad School at 217 Bascom. The Grad School offers more information on Auditing Courses in their Academic Policies and Procedures.

Tutorial website: http://www.registrar.wisc.edu/isis_helpdocs/enrollment_demos/V90CourseChangeRequest/V90CourseChangeRequest.htm

Auditing Courses: http://www.grad.wisc.edu/education/acadpolicy/guidelines.html#13

Overview of Material Covered
[Spring 2012]

- Quick C Intro
- General considerations in relation to trends in the chip industry
- Overview of parallel computation paradigms and supporting hardware/software
- GPU computing and the CUDA programming model
- GPU parallel computing using the Thrust template library
- MPI programming
- OpenMP programming
- Two lectures where each of you shares with the rest of the class how you use parallel computing (or plan to use it)
  - Your best practices, favorite approach to parallel programming, do's and don'ts, etc.

At the beginning of the road…

- Teaching the class for the third time
  - There are still some rough edges
  - There might be questions that I don't have an answer for
    - I promise I'll follow up on these and get back with you (on the Forum)
  - We'll cover a lot of topics and there'll be challenging assignments
- Please ask questions (be curious)

My Advice to You [is simple]

- If you can, innovate, do something remarkable, amaze the rest of us…

End ME964 Overview

Beginning: Quick Review of C

- Essential reading: Chapter 5 of “The C Programming Language” (Kernighan and Ritchie)
- Acknowledgement: Slides on this C Intro include material due to Donghui Zhang and Lewis Girod

C Syntax and Hello World

What do the < > mean?

#include inserts another file. “.h” files are called “header” files. They contain declarations/definitions needed to interface to libraries and code in other “.c” files.

A comment, ignored by the compiler

```
#include <stdio.h>
/* The simplest C Program */
int main(int argc, char **argv)
{
  printf(“Hello World\n”);
  return 0;
}
```

The main() function is always where your program starts running.

Blocks of code (“lexical scopes”) are marked by { … }

Return ‘0’ from this function

Lexical Scoping

Every Variable is Defined within some scope. A Variable cannot be referenced by name (a.k.a. Symbol) from outside of that scope.

Lexical scopes are defined with curly braces { }.

- The scope of Function Arguments is the complete body of that function.
- The scope of Variables defined inside a function starts at the definition and ends at the closing brace of the containing block
- The scope of Variables defined outside a function starts at the definition and ends at the end of the file. Called "Global" Vars.

```
void p(char x)
{
            /* p,x */
  char y;
            /* p,x,y */
  char z;
            /* p,x,y,z */
}
            /* p */
char z;
            /* p,z */

void q(char a)
{
  char b;
            /* p,z,q,a,b */

  {
    char c;
            /* p,z,q,a,b,c */
  }

  char d;
  /* p,z,q,a,b,d (not c) */
}

/* p,z,q */
```

char b?

legal?

Comparison and Mathematical Operators

```
== equal to
<  less than
<= less than or equal
>  greater than
>= greater than or equal
!= not equal
&& logical and
|| logical or
!  logical not
```

```
+  plus
-  minus
*  mult
/  divide
%  modulo
```

```
&  bitwise and
|  bitwise or
^  bitwise xor
~  bitwise not
<< shift left
>> shift right
```

Beware division:

- 5 / 10 → 0 *whereas* 5 / 10.0 → 0.5
- Division by 0 will cause a FPE

Don't confuse & and &&..
1 & 2 → 0 *whereas* 1 && 2 → <true>

The rules of precedence are clearly defined but often difficult to remember or non-intuitive. When in doubt, add parentheses to make it explicit.

Assignment Operators

```
x = y    assign y to x
x++      post-increment x
++x      pre-increment x
x--      post-decrement x
--x      pre-decrement x
```

```
x += y  assign (x+y) to x
x -= y  assign (x-y) to x
x *= y  assign (x*y) to x
x /= y  assign (x/y) to x
x %= y  assign (x%y) to x
```

Note the difference between ++x and x++ (high vs low priority (precedence)):

```
int x=5;
int y;
y = ++x;
/* x == 6, y == 6 */
```

```
int x=5;
int y;
y = x++;
/* x == 6, y == 5 */
```

Don’t confuse “=“ and “==“!

```
int x=5;
if (x==6)   /* false */
{
  /* ... */
}
/* x is still 5 */
```

```
int x=5;
if (x=6)   /* always true */
{
  /* x is now 6 */
}
/* ... */
```

A Quick Digression About the Compiler

```
#include <stdio.h>
/* The simplest C Program */
int main(int argc, char **argv)
{
  printf("Hello World\n");
  return 0;
}
```

Preprocess

Compilation occurs in two steps: "Preprocessing" and "Compiling"

```
__extension__ typedef  unsigned long long int
__dev_t;
__extension__ typedef  unsigned int    __uid_t;
__extension__ typedef  unsigned int    __gid_t;
__extension__ typedef  unsigned long int
__ino_t;
__extension__ typedef  unsigned long long int
__ino64_t;
__extension__ typedef  unsigned int
__nlink_t;
__extension__ typedef  long int    __off_t;
__extension__ typedef  long long int
__off64_t;
extern void flockfile (FILE *__stream)  ;
extern int ftrylockfile (FILE *__stream)  ;
extern void funlockfile (FILE *__stream)  ;
int main(int argc, char **argv)
{
  printf("Hello World\n");
  return 0;
}
```

In Preprocessing, source code is "expanded" into a larger form that is simpler for the compiler to understand. Any line that starts with '#' is a line that is interpreted by the Preprocessor.

- Include files are "pasted in" (#include)
- Macros are "expanded" (#define)
- Comments are stripped out (/* */ , //)
- Continued lines are joined (\ )

The compiler then converts the resulting text (called translation unit) into binary code the CPU can execute.

Compile

my_program

C Memory Pointers

- To discuss memory pointers, we need to talk first about the concept of memory
- We'll conclude by touching on a couple of other C elements:
  - Arrays, typedef, and structs

The “memory”

Memory: similar to a big table of numbered slots where bytes of data are stored.

The number of a slot is its Address.
One byte Value can be stored in each slot.

Some data values span more than one slot, like the character string “Hello\n”

A Type provides a logical meaning to a span of memory. Some simple types are:

`char`	a single character (1 slot)
`char [10]`	an array of 10 characters
`int`	signed 4 byte integer
`float`	4 byte floating point
`int64_t`	signed 8 byte integer

Addr	Value
0	
1	
2	
3	
4	‘H’ (72)
5	‘e’ (101)
6	‘l’ (108)
7	‘l’ (108)
8	‘o’ (111)
9	‘\n’ (10)
10	‘\0’ (0)
11	
12	

What is a Variable?

symbol table?

A Variable names a place in memory where you store a Value of a certain Type.

You first Declare a variable by giving it a name and specifying its type and optionally an initial value

declare vs. define

```
char x;
char y='e';
```

Variable x declared but undefined

Initial value

Name

What names are legal?

Type is single character (char)

extern? static? const?

The compiler puts x and y somewhere in memory.

Symbol	Addr	Value
	0	
	1	
	2	
	3	
x	4	Some garbage
y	5	'e' (101)
	6	
	7	
	8	
	9	
	10	
	11	
	12	

Multi-byte Variables

Different types require different amounts of memory. Most architectures store data on "word boundaries", or even multiples of the size of a primitive data type (int, char)

```
char x;
char y='e';
int z = 0x01020304;
```

0x means the constant is written in hex

padding

An int requires 4 bytes

Symbol	Addr	Value
	0	
	1	
	2	
	3	
x	4	Some garbage
y	5	'e' (101)
	6	
	7	
z	8	4
	9	3
	10	2
	11	1
	12	

Architecture uses little-endian convention, since it stores the most significant byte first