COMS/CPRE 425
Spring 2005
Lecture 3

Ricky A. Kendall
rickyk@cs.iastate.edu
Logistics (1)

• Questions
  ➢ Any questions from material of the last Lecture?
  ➢ Have you confirmed your CS accounts?
  ➢ Any Add/Drop slips need to be signed?

• Computer Access Forms
  ➢ SCL and NERSC forms please read, sign and return to me today.

• Everyone know what Matrix Multiply is?
  ➢ Do the following terms make sense?
    ▪ Ddot or simple loops
    ▪ Daxpy
    ▪ Blocked

• Any questions about the homework?
  ➢ Make sure you check the “messages” board on the course website.
Logistics (2)

• I have read the NERSC Policies and Procedures and understand my responsibilities in the use of NERSC resources.

Signature: <data>
Print Name: <data>
Citizenship: <data>
Organization: Iowa State University
Email Address: <data>
Work Phone Number: <data>
Principal Investigator: Ricky A. Kendall
Date: <data>
Logistics (3)

• I understand the computer use policies above and agree to follow them when using Scalable Computing Laboratory Resources.
• Name:__________________________________
• Telephone:______________________________
• Email Address:__________________________
• Sponsor's Name: Ricky A. Kendall
• Organization: ComS/CprE 425 ISU
• Signature:________________________________
• Date:___________________________________
• Access Needed: 4pack, redwing, osage
Logistics (4)

• I have had several questions regarding the best Linux/Unix distribution.
  ➢ For novice System Administrators (get it up and go)
    ▪ RedHat flavor of the month
      ♦ ISU has a site license-like agreement with Red Hat (See AIT).
    ▪ I’ve heard good things about Mandrake
  ➢ If you want to learn more about how things work
    ▪ Debian (What we use in the SCL)
    ▪ Gentoo
    ▪ Slackware
    ▪ FreeBSD 4.4
  ➢ You can download and burn images from:
    ▪ http://www.linuxiso.org
    ▪ If you want Linux and you don’t have access to a burner see me.
Logistics (5)

• **Reading Assignments** are posted on the calendar page.
  ➢ Chapters 1 and 2 were assigned for this and next week.

• **Homework due dates**
  ➢ #1 1/20/2004
  ➢ #2 2/3/2004

• **Midterm is scheduled on Wednesday, March 2\textsuperscript{nd}, 2005**
  ➢ Closed notes/book/student’s test next to you!
Design Document

• Really a combination of the Specifications and Code Design from “The Programming Process” outlined in the first lecture.

• It should include:
  - Detailed description of what the code does
  - A restatement of the requirements and your assumptions about those requirements
  - Options for input and runtime (e.g., how to run the code)
  - What output to expect
    ▪ Does output go to standard out (😊) or to a file(균)?
    ▪ Error codes and conditions
Design Document (2)

- **It should include (continued):**
  - An overview of the code structure.
  - Scope of each function and computed macro
    - What does the function do?
    - What input the function expects
    - Pseudo code or description of the interfaces for each function.
  - Description of the data structures.
    - E.g., Arrays A, B, Canalytical, Ccomputed are all linear arrays addressed using a simple macro that computes the linear offset from the two dimensions (row,column)
  - Algorithms used in the code
  - Any other information requested by the assignment
    - E.g., How did you test your generation functions to know that they worked.
Results and Analysis Document

• Pointers to or the actual output to runs
• Summary of output timings (e.g., tables of info)
• Any issues when running the code
  ➢ It works for all but the third test case.
• Why the performance was or was not what was expected.
  ➢ Comparisons of different algorithms.
    ▪ E.g., are all three generations routines O(N^2) where N is the rank of the matrix??
• Graphs of performance (use excel or gnuplot)
• If any runs are done in parallel then graphs of speedup and efficiency are required.
Makefiles

- Makefiles provide a mechanism for building applications.
  - They define the way in which software components interact.
  - They facilitate rapid prototyping by only compiling and linking software components that have changed since the last build.
  - You will only need to provide simple makefiles for the homework assignments but they must work properly.
    - If they don’t then that is a problem!!
Makefiles (2)

make uses a description file usually “makefile”, “Makefile” or “MAKEFILE” (or “GNUmakefile” for GNU make)

make [options] [-f description_file] [targets] [macros]

It is best to construct your “makefile” so that all you have to do is use the command “make”

Interesting options:
- -n : show all the commands that would be used but do not actually execute the commands.
- -p : print out all macros, rules and target commands that are used (lots of information!)
Makefiles (3)

What does a makefile look like?
#
# makefile for two matrix multiply codes
LIBS = -lm
OBJ = cputime.o walltime.o
all: mxm mxm2 # cool codes
    @echo "both mxm and mxm2 are made"
mxm: mxm.o $(OBJ)
    $(CC) -o $@ mxm.o $(OBJ) $(LIBS)
mxm2: mxm2.o $(OBJ)
    $(CC) -o $@ mxm2.o $(OBJ) $(LIBS)
clean: ; -rm -f $(OBJ) mxm.o mxm2.o
reallclean:
    @$ (MAKE) clean
    -rm -f mxm mxm2
Makefiles (4)

How does a makefile work?
#
# makefile for two matrix multiply codes
LIBS = -lm
OBJ = cputime.o walltime.o

all: mxm mxm2
  @echo "both mxm and mxm2 are made"
mxm: mxm.o $(OBJ)
  $(CC) -o $@ mxm.o $(OBJ) $(LIBS)
mxm2: mxm2.o $(OBJ)
  $(CC) -o $@ mxm2.o $(OBJ) $(LIBS)

% make
cc  -c mxm.c  -o mxm.o
cc  -c cputime.c  -o cputime.o
cc  -c walltime.c  -o walltime.o
cc  -o mxm  mxm.o  cputime.o
     walltime.o  -lm
cc  -c mxm2.c  -o mxm2.o
cc  -o mxm2  mxm2.o  cputime.o
     walltime.o  -lm
both mxm and mxm2 are made
Makefiles (5)

Managing Projects with make
by Andrew Oram and Talbott
http://www.amazon.com
http://www.barnesandnoble.com
~ $15
A complex makefile

OBJ = matvec.o genX.o genY.o genZ.o matvec_ddot.o matvec_daxpy.o \
    matvec_blocked.o set_it.o CPU_Time.o Wall_Time.o Analyze_matrix.o \
    print_mat.o vec_check.o vec_compare.o vec_getminmaxdiff.o
LIBS = -lm
CFLAGS = -g
.c.o:  $*.c
   $(CC) -c $(CFLAGS) -o $*.o $*.c
matvec: $(OBJ)
   $(CC) -o $@ $(CFLAGS) $(OBJ) $(LIBS)
genX.o: genXYZ.h printit.h
genY.o: genXYZ.h printit.h
genZ.o: genXYZ.h printit.h
matvec.o: proto.h
clean:
   @-rm -f $(OBJ)
reallclean:
   @-rm -f $(OBJ) matvec
High Performance Computing (HPC)

- Overview
- RISC Processors
  - pipelines
- Vector Systems
- Distributed and Parallel Systems
The Scope of HPC

• What is a High Performance Computer or a supercomputer?
  - There is no static definition.
  - It implies bigger, better, faster, and more expensive.

• Basically covers from the palmtop to the teraflop systems available.
  - Most based on RISC processors
  - COTS or NOW clusters
    ▪ Commodity of the Shelf
    ▪ Network of Workstations
  - Vector Supercomputers
  - SMP systems, MPP systems
  - IBM Blue Gene System
  - Japan’s Earth Simulator
Parallel Vs. Distributed Applications

• Many similarities and distinctions.
  • Parallel applications are primarily those that divide a given problem space into tasks that are executed concurrently.
    ➢ Often called Task Parallelism
  • Distributed Applications are primarily those that utilize the divided tasks on separate resources sometimes in different locations.
    ➢ Often called Functional Parallelism.
• Modern scalable applications utilize both kinds of functionality at some point.
Similarities of Parallel and Distributed Computing

- Multiple processors are involved
- Multiple processes that cooperate via some mechanism are operating concurrently.
- The processors (and processes) are interconnected by some network fabric (network protocol).
  - Fast Ethernet (MPI, MPICH, … etc.)
  - Myrinet (GM)
  - Quadrix (SC Software)
  - SCI (Scali Software).
- Parallel computing is often considered a sub-discipline of distributed computing.
  - One of the more difficult ones 😊
Distributed Computing Focus

- **Systems have loosely coupled resources**
  - Processors, disk, memory, tertiary storage.
  - Resources can be physically decentralized.
  - No shared memory or other resources.

- **Systems are heterogeneous, diverse, and dynamic.**
  - Resources change over time.
  - Different kinds of hardware involved
    - Multiple generations of hardware “co-exist”

- **Multiple applications (or instances) are run.**

- **Software is the key to providing the single system image to users.**
Parallel Computing Focus

- A single application is dedicated to the resource or a fraction thereof.
- The goal is to speed up the instance of that application.
- Resources are often tightly coupled and layered.
  - May have shared memory.
- Systems are often more homogeneous by design.
  - Technology usually doesn’t span generations.
- This is the focus of this class!!!
What is the Key word in HPC?

• **Performance**

• The exploitation of concurrency is the key to performance on ANY computer.

• **Even your Pentium processor uses concurrency!!**

  ➢ How?
Pipelines

• Pipelining is the use of a pipeline.
  ➢ When were pipelined instructions first used?
  ➢ 1950s. IBM stretch system (7030).

• In computers, a pipeline is the continuous and somewhat overlapped movement of instructions, data, arithmetic steps, etc.

• Pipelining is compared to a manufacturing assembly line in which different parts of a product are being assembled at the same time although ultimately there may be some parts that have to be preassembled.
  ➢ Even if there is some sequential dependency, the overall process can take advantage of those operations that can proceed concurrently.
Pipelines (2)

• **Without a pipeline,**
  - a processor fetches the first instruction, performs the operation, and then goes to get the next instruction, etc.
  - While fetching the instruction(s), the arithmetic part of the processor is idle. It must wait for the next instruction.

• **With pipelining,**
  - the architecture allows the next instruction to be fetched while the processor is performing arithmetic operations, holding them in a buffer close to the processor until each instruction operation can be performed.
  - The staging of instruction fetching is continuous. The result is an increase in the number of instructions that can be performed concurrently.
Pipelines (3)

- Computer processor pipelining is sometimes divided into an instruction pipeline and an arithmetic pipeline.
- The instruction pipeline represents the stages in which an instruction is moved through the processor, including its being fetched, perhaps buffered, and then executed.
- The arithmetic pipeline represents the parts of an arithmetic operation that can be broken down and overlapped as they are performed.
- Pipelines also apply to computer memory subsystems and moving data through various memory staging places. A cache is sort of a pipeline for data.
Vector Computers

• Are similar to RISC “arrays”
  ➢ A simple array extension of a scalar property

• There are “vector”
  ➢ Registers
  ➢ Buffers
  ➢ Instructions
  ➢ Memory operations

• Utilizes similar ideas as RISC pipelines.

• Usually no cache but streams data from a “flat” but “banked” memory
  ➢ The compiler is supposed to handle this part 😊
Vector Operations

A_1 A_2 A_3 • • • A_N

Main Memory

B_1 B_2 B_3 • • • B_N

Vector Registers

C_1 C_2 C_3 • • • C_N

Main Memory

Vector Register Length

N > VRL
Examples of Vector Computers

• Cray
  ➢ Cray-1, Cray-2, XMP, YMP, C90, T90, J90
  ➢ Cray X1 (first one is at Oak Ridge National Lab).

• Fujitsu

• NEC

• Hitachi

• Earth Simulator
  ➢ The ES is a distributed memory parallel computer
    with vector nodes 😊
Vector Computers are

• **Not as relevant since:**
  - Modern RISC processors are cheaper to build
  - Operations take fewer cycles on RISC
    - 4 to 25 now as opposed to 100 cycles in the past
  - Applications actually need more complex pipelines
    - A vector operation is a single concurrent operation on all vector elements.
    - Some Superscalar RISC processors do 2 Multiply and 2 add operations (4 total) at once.
  - But are making a comeback with the Earth Simulator and the Cray X1

• **Relevant for memory bound applications**
  - Bandwidth on RISC systems is usually limited.

• Earth Simulator is relevant!! Why?
What is a parallel Computer?

• A computer that utilizes concurrency on multiple processors to complete tasks.
  ➢ Multicomputer (separate computers connected via network)
  ➢ Multiprocessor (Symmetric Multiprocessor or SMP)

• Most computers today exploit some level of concurrency.
  ➢ Multiple floating point units
  ➢ Multiple processors
  ➢ Overlapping memory load/store
  ➢ Overlapping Disk I/O or Network I/O
  ➢ Multiple thing-a-ma-bobs!
Flynn’s Taxonomy

- Single DATA: SISD (von Neumann)
- Multiple DATA: SIMD (MasPar)
- Single INSTRUCTION: MISD
- Multiple INSTRUCTION: MIMD (IBM SP)
SISD or von Neumann

Diagram:
- Processor
- Memory
- I/O
SISD or von Neumann

CU₁

PU₁

MM₁

Instructions

Data
SISD Examples

PC
Macs
SGI
IBM
HP
VAX
MISD

Instruction

CU₁ → PU₁ ← MM₁
CU₂ → PU₂ → MM₂
CUₘ → PUₘ → MMₘ

Data
MISD Examples

Intel Iwarp
SIMD

CU

PU₁

PU₂

PUₘ

Data

MM₁

MM₂

MMₘ

Instruction
SIMD Examples

- Thinking Machines (CM2, CM5)
- MasPar
- DAP
- G4 Array Processor system
  - Altivec unit.
  - Single precision only 😞

- What is the most famous CM5?
MIMD (SM)
MIMD Examples (SM)

- Cray YMP, C90, J90, T90
- SGI Origin 2000
- Sun Enterprise, Ultra
- IBM AS400
- Cray-X1
- Also called SMP
  (Symmetric Multi-Processor)
MIMD (DM)

CU_1 \rightarrow PU_1 \leftarrow MM_1

CU_2 \rightarrow PU_2 \leftarrow MM_2

CU_m \rightarrow PU_m \leftarrow MM_m

Instruction

Communication

Data
Communication Subsystems

• Proprietary Company Owned Switch/Net
  ➢ CrayLink
  ➢ IBM SP Switch
  ➢ Paragon Mesh

• Third Party Proprietary Switch/Net
  ➢ Myrinet
  ➢ GigaNet (VIA)
  ➢ Quadrics Switch
  ➢ SCI (Scalable Coherent Interconnect)

• Ethernet
  ➢ Gigabit, Fast Ethernet (100mbs)
MIMD Examples (DM)

- Intel Touchstone Delta and Paragon
- N-cube
- IBM SP
- Cray T3D, T3E (special)
- Networks of Workstations
- Clusters (Compaq/Quadrics, IBM, …)
- Kendall Square Research (special)
- IBM SP
SIMD vs. MIMD Systems

• Single Instruction Stream Limits “general use”

• Can you run SIMD applications on MIMD systems?
  ➢ What would happen?

• Can you run MIMD applications on SIMD systems?
  ➢ What would happen?
What is the Architecture of current MPP systems?

- A global node based MIMD architecture
- Each node is an SMP
  - 4 processor (PC cluster, IBM SP, Compaq)
  - 8 processor (PC cluster, IBM SP, HP)
  - 16 processor (IBM SP, SGI)
  - 32 processor (IBM SP, SGI)
  - 128 processor (SGI)
MIMD (DM)

An SMP Unit

Instruction

Communication

CU_1 → PU_1 → MM_1 → MM_2 → PU_2 → CU_2

CU_m → PU_m → MM_m → MM_2 → PU_2 → CU_2
What is the Architecture of Future MPP systems?

- A global node based MIMD architecture
- Each node is an SMP
  - 16 processor (PC cluster, IBM SP, SUN)
  - 32 processor (PC cluster, IBM SP, SGI)
  - 512 processor (SGI)
  - Others will have bigger SMP nodes.
- Earth simulator
  - Cluster with Vector parallel nodes!
SMP Factoid

• **Global Address Space**
  - Program can see all of memory
  - easy to design parallel applications
  - straightforward to provide performance
  - synchronization has a modest cost.

• **Single instance of the Operating System**

• **Automatic Parallelizing compilers available**
  - Performance only available with programmer optimization.
Question

• Why do vendors like SMPs?
  ➢ Lower packaging costs
  ➢ Smaller footprint for total Macho-Flops
  ➢ Lower aggregate power costs.
    ▪ The biggest hindrance to more processing per “core”
  ➢ Automagic parallel compilers are possible.
Distributed Memory Factoid

• **Address Space Local to node**
  - Program can see only local memory
  - straightforward to design parallel applications
  - sometimes difficult to provide performance
  - synchronization is very costly.

• **Multiple instances of the Operating System**

• **Automatic Parallelizing compilers are impossible**
  - Some compilers are available but they are still in diapers.
Hybrid Systems

• Cray T3D and T3E
  ➢ Physically distributed memory
  ➢ fast memory interconnect
  ➢ Hardware based remote memory read and write operations.
  ➢ Globally addressable memory
    ▪ A single address space!!
Parallel vs. Sequential Programming

Algorithm
Sequential
Optimizing Compilers
Processor Architecture
Synchronization
Parallel
Programming Models
Memory Architecture
O/S
Deadlock
Message Passing
Device drivers, ...
Tip of Iceberg

Programming Model Considerations

Parallelizing your application is considered RE-BUGGING your code!!!
How do you write applications for these MIMD-DM systems?

- **SPMD**
  - Single Program Multiple Data
    - Flynn-ism
  - Do something different based on Process or Thread ID.

- The same program executes on all processors.
  - Via processes, threads, or both.

- The method and speed of addressing memory on each node is the key to performance.