7.5 credit points

Instructor:
Sally A. McKee
What is a parallel computer?

A parallel computer is a collection of processing elements that cooperate to solve large problems (fast).
Why parallel computers?

Technology trends

Killer microprocessors

New performance demanding applications

Economics

A collection of killer microprocessors (integrated on the same chip)
A parallel computer is a collection of processing elements that cooperate to solve large problems (fast)

Programming issues:
- What programming model?

Architectural model issues:
- How big a collection?
- How powerful are the elements?
- How do the elements cooperate?

Performance — cross cutting issues:
- Impact of system design tradeoffs on application performance
- Impact of application design on performance
The goal of this course is to provide knowledge on
- Programming models and techniques for design of high-performance parallel programs
  - the data parallel model
  - the shared address-space model
  - the message-passing model
- Design principles for parallel computers
  - small-scale system design tradeoffs
  - scalable system design tradeoffs
  - interconnection networks
Course requirements and material

Course requirements:

→ Lab and approved projects
→ Final exam (written: concepts and problem solving)

Course material

→ Textbook: Culler, Singh, and Gupta “Parallel Computer Architecture: A Hardware/Software Approach” (available at Cremona)
→ Draft chapters from forthcoming text
→ NEED VOLUNTEERS FOR CLASS TESTING!
→ Solutions manual (from web site)
→ Copies of slides (from web site)

http://www.cse.chalmers.se/edu/course/EDA281
http://www.cse.chalmers.se/edu/year/2009/course/EDA281/

Visit regularly
Overview of parallel computer technology:
What it is? What it is for? What are the issues?

- Driving forces behind parallel computers (1.1)
- Evolution behind today’s parallel computers (1.2)
- Fundamental design issues (1.3)
- Methodology for designing parallel programs (2.1 – 2.2)
Three driving forces

- Application demands (coarse-grain parallelism abounds):
  - Scientific computing (e.g., modeling of phenomena in science)
  - Engineering computing (e.g., CAD and design analysis)
  - Commercial computing (e.g., media and information processing)

- Technology trends
  - Transistor density growth high
  - Clock frequency improvement moderate

- Architecture trends
  - Diminishing returns on instruction-level parallelism
Parallelism in sequential programs

A sequential program on a superscalar processor:

- **Programming model:**
  - Sequential

- **Architecture:**
  - Instruction-level parallelism
  - Register (memory) communication
  - Pipeline interlocking for synchronization

---

for i = 0 to N-1
  a[(i+1) mod N] := b[i] + c[i];
for i = 0 to N-1
  d[i] := C*a[i];

Iteration: 0 1 2 ... N-1
Loop 1 a[1] a[2] ... a[0]
Loop 2 a[0] a[1] ... a[N-1]

---

Gap between model and architecture has increased
A parallel programming model

Extended semantics to express
- units of parallelism at the
  - instruction level
  - thread level
  - program level
- communication and coordination between units of parallelism at the
  - register level
  - memory level
  - I/O level
### Programming model vs. parallel architecture

<table>
<thead>
<tr>
<th>CAD</th>
<th>Databases</th>
<th>Scientific modeling</th>
<th>Parallel applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiprogramming</td>
<td>Shared address</td>
<td>Message passing</td>
<td>Data parallel</td>
</tr>
<tr>
<td>Compiler or library</td>
<td>Operating system support</td>
<td>Communication abstraction</td>
<td>User/system boundary</td>
</tr>
<tr>
<td>Communication hardware</td>
<td>Hardware/software boundary</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Physical communication medium</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Three key concepts**

- Communication abstraction supports programming models
- Communication architecture (ISA plus primitives for comm/synch)
- Hardware/software boundary to define which parts of the communication architecture are implemented in hardware or software
Shared address space (SAS) model

Programming model

- Parallelism
  - parts of a program, called *threads*
- Communication and coordination among threads
  - through a shared global address space

```c
for_all i = 0 to P-1
  for j = i0[i] to in[i]
    a[(j+1) mod N] := b[j] + c[j];
  barrier;
for_all i = 0 to P-1
  for j = i0[i] to in[i]
    d[j] := C*a[j];
```

Communication abstraction supported by HW/SW interface
Message passing model

Programming model

- Process-level parallelism (private addresses)
- Communication and coordination via explicit messages

```plaintext
for_all i = 0 to P-1
  for j = i0[i] to in[i]
    index = (j+1) mod N;
    a[index] := b[j] + c[j];
    if j = in[i] then
      send(a[index], (j+1) mod P, a[j]);
    end_for
  end_for
 barrier;
for_all i = 0 to P-1
  for j = i0[i] to in[i]
    if j = i0[i] then
      recv(tmp, (P+j-1) mod P, a[j]);
    end_for
  end_for
```
Data parallel systems

- Programming model
  - Operations performed in parallel on each element of data structure
  - Logically, single thread of control performing sequential or parallel steps
  - Conceptually, a processor associated with each data element

- Architectural model
  - Array of many simple, cheap processors with little memory each (processors don’t sequence through instructions)
  - Attached to a control processor that issues instructions
  - Specialized and general communication, cheap global synchronization

Original motivations
- Matches simple differential equation solvers
- Centralizes high cost of instruction fetch/sequencing
Pros and cons of data parallelism

Example

\[
\begin{align*}
\text{parallel} & \quad (i:0->N-1) \quad a[(i+1) \text{ mod } N] := b[i] + c[i]; \\
\text{parallel} & \quad (i:0->N-1) \quad d[i] := C * a[i];
\end{align*}
\]

Evolution and convergence:

- Popular when cost savings of centralized sequencer is high
- Parallelism is limited to specialized regular computations
  - Much of parallelism can be exploited at instruction level
  - Coarser levels of parallelism can be exposed for multiprocessors and message-passing machines

*New data parallel programming model: SPMD*

*Single-Program Multiple-Data*
A generic parallel architecture

A generic modern multiprocessor (shared address or message passing architecture)

Node: processor(s), memory system, plus a *communication assist*

- Network interface and communication controller
- Scalable network
- Convergence allows lots of innovation, now within framework
  - Integration of assist with node, what operations, how efficiently...
Summary

The course in a nutshell:

- Efficient mapping of algorithms to models (Chs. 2-3)
- Design principles of small-scale shared address space machines (Chs. 5-6)
- Design principles for large-scale message passing and shared-address space machines (Chs. 7, 10)
- Design principles for memory systems in large-scale multiprocessors (Ch. 8)