Introduction

- Why parallel architectures?
  - Absolute performance
  - Cost performance
  - Reliability and availability

- Key enabling factors
  - Advances in microprocessor and interconnect technology
  - Advances in software technology
Parallel Machines

• Definition: "A parallel computer is a collection of processing elements that can cooperate and communicate to solve large problems fast." --- Almasi and Gottlieb.

• "... a collection of PEs..."
  • How many? How powerful is each? Scalability?
  • Few very powerful (e.g., Cray YMP) or many weak ones (e.g., CM-2). Cost is a factor.

• "... that can communicate ..."
  • How do PEs communicate? (e.g. shared memory vs message passing)
  • Interconnection network architecture? (bus, crossbar, multistage, ...)
  • Evaluation criteria: cost, latency, throughput, scalability, fault tolerance
• "... and cooperate ...

  • Important issues are synchronization, granularity, and autonomy

  • Synchronization allows sequencing of actions for correctness
    • Variety of primitives (test&set, fetch&add, ...); scalability issues

  • Granularity corresponds to the size of subtasks in program
    • Granularity ↓ =⇒ Parallelism ↑; Communication ↑; Overhead ↑
    • Typical grain sizes vs. number of instructions
      • program level 10^6+ instructions
      • task level 10^3-10^6 instructions
      • loop level 10-1000 instructions
      • stmt/instr level 2-10 instructions

  • As for autonomy, the tradeoff is between SIMD and MIMD machines
    • MIMD machines are more general purpose, but overheads can be large if PEs need to synchronize frequently
• "... to solve large problems fast ...
  • General purpose vs. special purpose machines?
    • Any machine can do well on some problems!!

• What applications are amenable to parallel processing:
  • Highly (often embarassingly) parallel applications
    • Many scientific codes that exploit data parallelism (monte-carlo)
  • Medium parallel applications
    • Many engineering applications (finite-element pgms, VLSI-CAD)
  • Not-so-parallel applications
    • Examples are compilers, editors, ... (do we care?)

• Application and machine scaling issues
Classification of Parallel Architectures

- Michael Flynn's classification

- Model-based classification
Flynn's Classification

- Based on notions of:
  - instruction streams, and
  - data streams

- The parallel organizations are characterized by the multiplicity of hardware provided to service I and D streams:
  - SISD: Single Instruction and Single Data stream (SparcStation)
  - SIMD: Single Instruction and Multiple Data streams (CM-2)
  - MISD: Multiple Instruction and Single Data streams (CMU Warp)
  - MIMD: Multiple Instruction and Multiple Data streams (Sequent)
Shared Memory Architectures

- Key Feature: All processors in the system can **directly** access all memory locations in the system, thus providing a convenient and cheap (?) mechanism for multiple processors to share data.
  - convenient: (i) location transparency; (ii) abstraction supported is same as that on current day uniprocessors.
  - cheap: as compared to other models (more later).

- Memory can be centrally placed or distributed in such machines.
- Better name is **Single Address Space** machines.
• Programming Model:
    • Variety of parallel models can be easily supported. Examples:
        • Fork-Join model; Task Queue model; Data parallel model.
    • Parallel threads use shared memory for communication and for synchronization.

• A problem traditionally cited with such machines is **Scalability**.

• However, the programming model supported is very general, and if cost-effective and scalable machines can be built, other models can be very easily emulated.
Message Passing Architectures

- Processors can directly access only local memory, and all communication and synchronization happens via messages.

- Some issues:
  - Sending a message often incurs many overheads: building a header; copying data into network buffers; sending data; receiving data into buffers; copying data from kernel to user process' address space.  
    >>>> Many of these steps require OS intervention.
  - Synchronization using messages is based on various handshake protocols (as in I/O buses).
  - One of the main advantages is easy scalability.
• From one perspective, single-address space machines can be viewed as message-passing machines.
  • Reading a remote location is simply sending a message to the remote node asking for the contents of that location.

• The difference boils down to efficiency and performance.
  • A single-address space machine is specialized to handle memory read/write messages in hardware and is thus faster at such operations. It also leaves the processor free to handle other computational tasks.

• A variety of programming models can be supported:
  • Actor model or Concurrent Object-Oriented Programming.
Dataflow Architectures

- In the dataflow model, instructions are activated by the availability of data operands for instructions.

- In contrast, in control flow models, computation is a series of instructions with implicit or explicit sequencing between them.

\[
\begin{align*}
\ a &= (b+1)*(b-c) \\
\ d &= c^e \\
\ f &= a*d \\
\end{align*}
\]

- One of the advantages is that all dependencies are explicitly present in the dataflow graph, so parallelism is not hidden from hardware.

Dataflow graph
The general structure of a dataflow machine is as follows:

- Many variations exist:
  - Static dataflow versus dynamic dataflow machines
  - Data-driven machines versus demand-driven machines

Some issues:
- granularity of operations (locality issues and macro dataflow)
- efficient handling of complex data structures like arrays
- complexity of matching store and memory units
- problems due to excess parallelism
Systolic Architectures

- Basic principle: By replacing a single PE by a regular array of PEs, and by carefully orchestrating the flow of data between the PEs, high throughput can be achieved without increasing memory bandwidth requirements.

- Distinguishing features from regular pipelined computers:
  - Array structure can be non linear (e.g, hexagonal)
  - Pathways between PEs may be multidirectional
  - PEs may have local instruction and data memory, and are more complex than the stage of a pipelined computer.
- Example: Systolic array for 1-D convolution

\[
y(i) = \sum_{j=1}^{k} w(j) x(i-j)
\]

- Issues:
  - System integration: Shipping data from host array and back
  - Cell architecture and communication architecture (Warp, iWarp)
  - Software for automatically mapping computations to systolic arrays
  - General purpose systolic arrays
SIMD Data Parallel Architectures

- Programming model assumes that there is a processor associated with each member of a collection of data.

- Example:
  - Each PE contains an employee record with his/her salary
  - Code: If salary > 100K then
    ```
    salary = salary * 1.05
    ```
  - else
    ```
    salary = salary * 1.10
    ```
  - Logically, the whole operation takes a single cycle.
  - Notion of enabled processors.

- Other examples: Document searching, graphics, image processing, ...

- Machines: Maspar MP-1, Thinking Machines CM-2 and CM-5.