What is a Parallel Architecture?

A parallel computer is a collection of processing elements that cooperate to solve large problems fast.

Some broad issues:

- **Resource Allocation:**
  - how many processing elements (PEs)?
  - how powerful are the PEs?
  - how much memory?

- **Data access, Communication and Synchronization**
  - how do the PEs cooperate and communicate?
  - how are data transmitted between PEs?
  - what are the abstractions and primitives for cooperation?

- **Performance and Scalability**
  - how does it all translate into performance?
  - how does it scale?

Issues in Multiprocessors

Which **programming model for interprocessor communication**

- shared memory
  - regular loads & stores
  - IBM Power 7 (8), Intel Core 2 Quad (4), Cray T3D, Sun Niagara 3 (16), AMD Quad Phenon (4), Sun Ultra Enterprise (72)

- message passing
  - can directly access only private address space
  - explicit sends & receives for shared data
  - IBM BlueGene/L (64), Intel Paragon
Shared Memory vs. Message Passing

Shared memory
+ simple parallel programming model
  • global shared address space
  • not worry about data locality \textit{but}
    \textit{get better performance when program for data placement}
    \textit{lower latency when data is local}
  • \textit{but} can do data placement if it is crucial, but don’t have to
  • hardware maintains data coherence & threads synchronize to order processor’s accesses to shared data
  • like uniprocessor code so parallelizing by programmer or compiler is easier
  \Rightarrow \text{can focus on program semantics, not inter-processor communication or data layout}

Spring 2011 CSE 471 - Multiprocessors 3

Shared Memory vs. Message Passing

Shared memory
+ low latency (no message passing software) \textit{but}
  overlap of communication & computation
  latency-hiding techniques can be applied to message passing machines
+ higher bandwidth for small transfers \textit{but}
  usually the only choice

Spring 2011 CSE 471 - Multiprocessors 4
Shared Memory vs. Message Passing

Message passing
+ abstraction in the programming model encapsulates the communication costs but
  overheads: copying, buffer management, protection
  additional language constructs
  need to program for nearest neighbor communication
+ no coherency hardware
+ good throughput on large transfers but
  what about small transfers?
+ more scalable (memory latency for uniform memory doesn’t scale with the number of processors) but
  large-scale SM has distributed memory also
  * hah! so you’re going to adopt the message-passing model?

Why there was a debate
• little experimental data
• not separate implementation from programming model
• can emulate one paradigm with the other
  • MP on SM machine
    message buffers in local (to each processor) memory
    copy messages by ld/st between buffers
  • SM on MP machine
    ld/st becomes a message copy
    sloooooooooow

Who won?
Issues in Multiprocessors

Which **execution model**
- control parallel
  - identify & synchronize different asynchronous threads
- data parallel
  - same operation on different parts of the shared data space
- dataflow (later)

Issues in Multiprocessors

How to **express error-free parallelism** (hardest problem)
- language support
  - HPF, ZPL
- runtime library constructs to support threads
  - coarse-grain, explicitly parallel C programs
- automatic (compiler) thread creation
  - implicitly parallel C & Fortran programs, e.g., SUIF & PTRANS compilers
- HW & compiler support for maintaining correctness (today’s efforts)
**Flynn’s Taxonomy**

Classifies computers by control & data streams

<table>
<thead>
<tr>
<th>Single Instruction, Single Data (SISD) (single-context uniprocessor)</th>
<th>Single Instruction, Multiple Data (SIMD) (single PC: Vector, CM-2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple Instruction, Single Data (MISD) (systolic arrays, streaming processors, GPUs)</td>
<td>Multiple Instruction, Multiple Data MIMD (Clusters, SMP servers)</td>
</tr>
</tbody>
</table>

**Systolic Architectures**

Replace single processor with array of regular (or specialized) processing elements
Orchestrate data flow for high throughput with less memory access
Important Issues

- the programming model debate for inter-processor communication
- Flynn’s taxonomy