Issues in Multiprocessors

Which **programming model for interprocessor communication**
- shared memory
  - regular loads & stores
- message passing
  - explicit sends & receives

Which **execution model**
- control parallel
  - identify & synchronize different asynchronous threads
- data parallel
  - same operation on different parts of the shared data space
Issues in Multiprocessors

How to express parallelism

- language support
  - HPF, ZPL
- runtime library constructs
  - coarse-grain, explicitly parallel C programs
- automatic (compiler) detection
  - implicitly parallel C & Fortran programs, e.g., SUIF & PTRANS compilers

Algorithm development

- embarrassingly parallel programs could be easily parallelized
- development of different algorithms for same problem
Issues in Multiprocessors

How to get good parallel performance

- recognize parallelism
- transform programs to increase parallelism without decreasing processor locality
- decrease sharing costs
Flynn Classification

**SISD**: single instruction stream, single data stream
- single-context uniprocessors

**SIMD**: single instruction stream, multiple data streams
- exploits data parallelism
- example: Thinking Machines CM

**MISD**: multiple instruction streams, single data stream
- systolic arrays
- example: Intel iWarp, streaming processors

**MIMD**: multiple instruction streams, multiple data streams
- multiprocessors
- multithreaded processors
- parallel programming & multiprogramming
- relies on control parallelism: execute & synchronize different asynchronous threads of control
- example: most processor companies have MP configurations
Figure 1. Connection Machine system organization.
Systolic Array

\[
\begin{align*}
a \cdot b & + x \cdot y = a \cdot w + b \cdot x + a \cdot y + b \cdot z \\
\text{matrix multiplication}
\end{align*}
\]

\[
\text{Systolic array}
\]
MIMD

Low-end

- bus-based
  - simple, but a bottleneck
  - simple cache coherency protocol
- physically centralized memory
- uniform memory access (UMA machine)
- Sequent Symmetry, SPARCCenter, Alpha-, PowerPC- or SPARC-based servers
Low-end MP

- Processor
  - One or more levels of cache
- Processor
  - One or more levels of cache
- Processor
  - One or more levels of cache
- Processor
  - One or more levels of cache

Main memory

I/O System
MIMD

High-end

- higher bandwidth, multiple-path interconnect
  - more scalable
  - more complex cache coherency protocol (if shared memory)
  - longer latencies
- physically distributed memory
- non-uniform memory access (NUMA machine)
- could have processor clusters
- SGI Challenge, Convex Examplar, Cray T3D, IBM SP-2, Intel Paragon
High-end MP

Interconnection Network

Processor + cache

Memory

I/O

Processor + cache

Memory

I/O

Processor + cache

Memory

I/O

Processor + cache

Memory

I/O

Processor + cache

Memory

I/O
Comparison of Issue Capabilities

- **Superscalar horizoneal waste**
- **Traditional Multithreading**
- **Single-chip Multiprocessor**
- **SMT**

### Issue Slots

- **Thread 1**
- **Thread 2**
- **Thread 3**
- **Thread 4**
- **Thread 5**
MIMD Programming Models

Address space organization for physically distributed memory
- distributed shared memory
  - 1 global address space
- multicomputers
  - private address space/processor

Inter-processor communication
- shared memory
  - accessed via load/store instructions
  - SPARCCenter, SGI Challenge, Cray T3D, Convex Exemplar, KSR-1&2
- message passing
  - explicit communication by sending/receiving messages
  - TMC CM-5, Intel Paragon, IBM SP-2
Shared Memory vs. Message Passing

Shared memory
+ simple parallel programming model
  • global shared address space
  • not worry about data locality **but**
    *get better performance when program for data placement*
    *lower latency when data is local*
    • **but** can do data placement if it is crucial, but don’t have to
  • hardware maintains data coherence
  • synchronize to order processor’s accesses to shared data
  • like uniprocessor code so parallelizing by programmer or compiler is easier
    ⇒ can focus on program semantics, not interprocessor communication
Shared Memory vs. Message Passing

Shared memory

+ low latency (no message passing software) \textit{but}
  
  \textit{overlap of communication \& computation}

  \textit{latency-hiding techniques can be applied to message passing machines}

+ higher bandwidth for small transfers \textit{but}
  
  \textit{usually the only choice}
Shared Memory vs. Message Passing

Message passing

+ abstraction in the programming model encapsulates the communication costs *but*
  
  *more complex programming model*
  
  *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 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?
Shared Memory vs. Message Passing

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
      sloooooooow

Who won?