Issues in Multiprocessors

Which **programming model**

- 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**

- automatic (compiler) detection
    - implicitly parallel C & Fortran programs -- SUIF compiler
- language support
    - HPF
- runtime library constructs
    - coarse-grain, explicitly parallel C programs

Algorithm development

How to get **good parallel performance**

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

Flynn Classification

SISD: single instruction stream, single data stream

- single-context uniprocessors

SIMD: single instruction stream, multiple data streams

- Thinking Machines CM-1

MISD: multiple instruction streams, single data stream

- systolic arrays (Intel iWarp)

MIMD: multiple instruction streams, multiple data streams

- multiprocessors
- multithreaded processors

MIMD

Parallel programming **&** multiprogramming

Off-the-shelf processors

Lower-end

- bus-based (bottleneck)
- physically centralized memory
- uniform memory access (UMA)
- Sequent Symmetry, SPARCCenter, an Alpha model

Higher-end

- higher bandwidth interconnect
- physically distributed memory
- nonuniform memory access (NUMA)
- could have processor clusters
- SGI Challenge, Convex Examplar, Cray T3D, IBM SP-2, Intel Paragon

MIMD Programming Models

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

Address space organization for physically separated memory

- **distributed shared memory**
  - 1 (logical) global address space
- **multicomputers**
  - private address space/processor

Shared Memory vs. Message Passing

Why there is 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 private (to each processor) memory
    copy messages ⇒ ld/st between buffers
  - SM on MP machine
    ld/st ⇒ copy messages
    slooooooooooow

Shared Memory vs. Message Passing

Shared memory

+ simple parallel programming model
  - can focus on program semantics, not interprocessor communication
  - global shared address space
  - hardware maintains protection & data coherence
  - synchronize to order processor's accesses to shared data
  - like uniprocessor code so parallelizing is easier
  - not worry about data locality ***but***

    lower latency when data is local (memory, cache, block)

    false sharing
+ low latency (no message passing software) ***but***

  overlap of communication & computation

  latency-hiding techniques can be applied to MP
+ higher bandwidth for small transfers

  but *usually the only choice*

Shared Memory vs. Message Passing

Message passing

+ abstraction encapsulates cost
  - ***but*** *more complex programming model*

    additional language constructs

    program for nearest neighbor communication
+ no coherency hardware
+ good throughput on large transfers

  but *what about small transfers?*
+ more scalable

  but *large-scale SM has distributed memory also*

Cache Coherency

Cache coherent processors
- reading processor must get the most current value
- most current value is the last write

Cache coherency problem
- updates from 1 processor not known to others

Cache Coherency Protocols

Write-invalidate

(Sequent Symmetry, SGI Power/Challenge, SPARCCenter 2000)

- exclusive access for writes ("owner")
- invalidation (coherency) miss
- cache-to-cache transfers
- good for:
  - migratory sharing
  - multiple writes to same word or block
- false sharing
  - processors read & write to ***different*** words in a shared cache block
  - block ownership bounces between processor caches

Write-update

(SPARCCenter 2000)

- broadcast each write to shared data
- good for inter-processor contention

Cache Coherency Protocols

Snooping

- distributed protocol
    - coherency state associated with each cache block
- used on buses
    - broadcast medium
    - entire coherency operation is atomic wrt other processors
        - split-transaction buses: do not initiate a transaction for a block that has a transaction in progress
- snoop implementation:
    - separate tags & state
    - snoop on highest level cache
      inclusion property
      physical cache

Cache Coherency Protocols

Directory-based

- coherency state associated with memory block
    - which processors have read/write access
    - optimization
      space for 4 processors
      trap for more
- **home** node (**local** node/**remote** node)
- usually point-to-point communication ⇒ multipath interconnect
- additional states because more transitions

False Sharing

Processes share cache blocks, not data

Impact aggravated by:
- block size
- cache size
- large miss penalties

Reduced by:
- coherency protocols
- compiler optimizations
- cache-conscious programming

Examples

Alpha
- 4way MP
- bus-based, snooping
- write-invalidate **&** write-update
  - begin with write-update; snoop decides to switch
- MESI coherency protocol
  - shared/private, clean/dirty state, invalid
  - cache-to-cache transfers
- inclusion

R10000
- clusters of 4-way MPs
- snoop (duplicate snoop tags) or directory-based
- write-invalidate/write-update on page basis
- MESI
- inclusion

Examples

UltraSPARC

- 4way MP
- bus-based, snooping
- MESI
    - cache-to-cache transfers
    - no snarfing
- dual-ported tags

P6

- clusters of 4-way MPs
- bus-based, snooping
- MESI
    - cache-to-cache transfers
    - snarfing