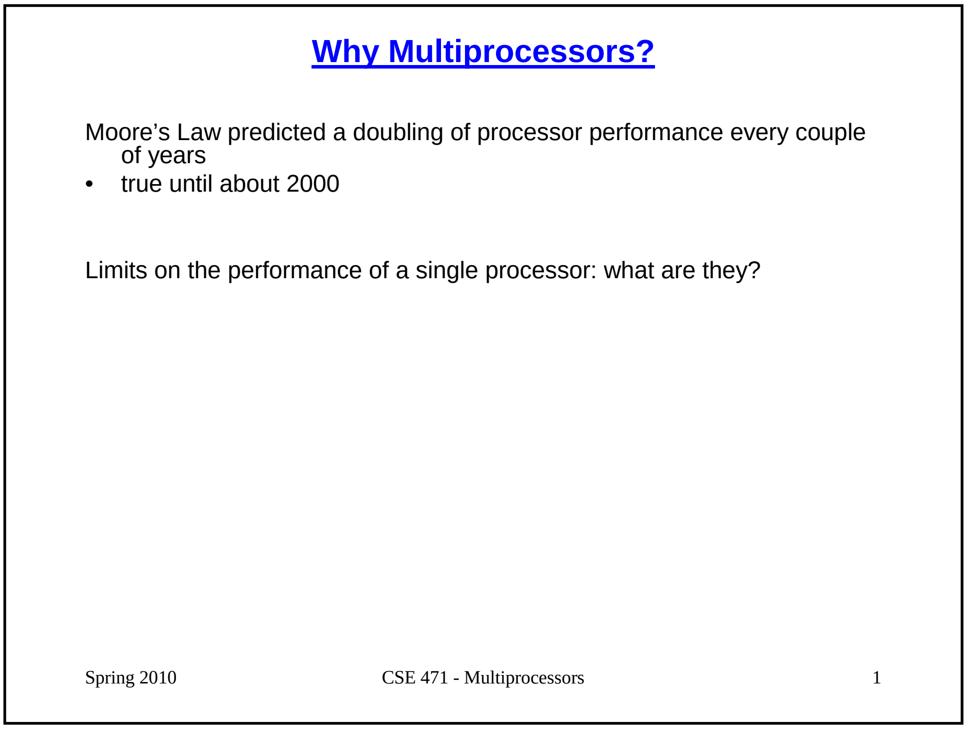
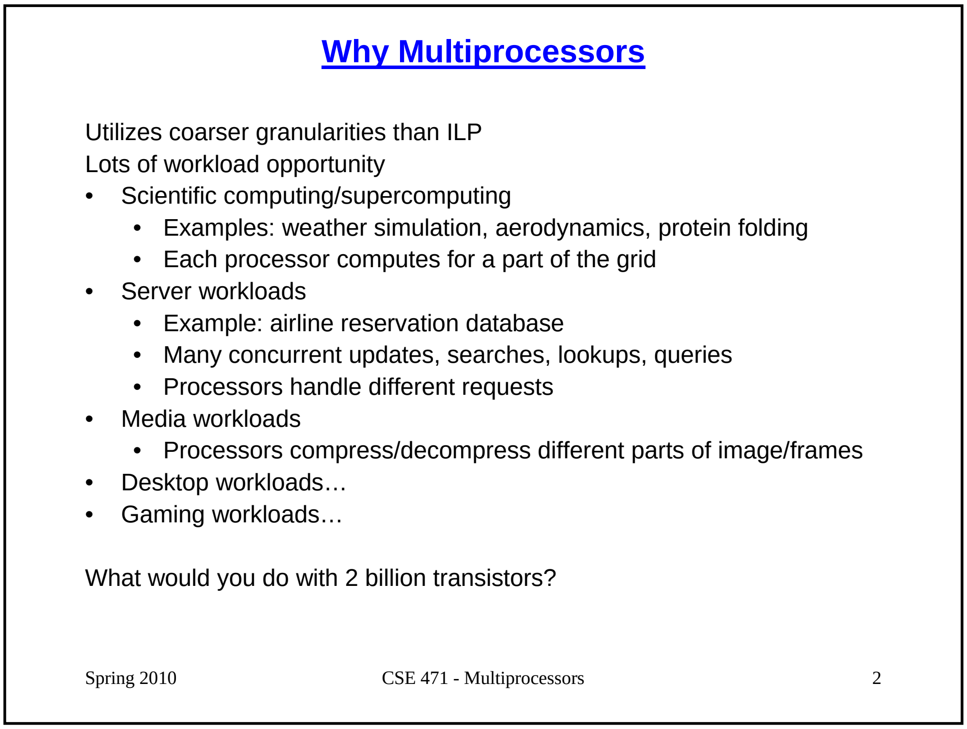


Why Multiprocessors?

Moore’s Law predicted a doubling of processor performance every couple of years

- true until about 2000

Limits on the performance of a single processor: what are they?



Why Multiprocessors

Utilizes coarser granularities than ILP

Lots of workload opportunity

- Scientific computing/supercomputing
    - Examples: weather simulation, aerodynamics, protein folding
    - Each processor computes for a part of the grid
- Server workloads
    - Example: airline reservation database
    - Many concurrent updates, searches, lookups, queries
    - Processors handle different requests
- Media workloads
    - Processors compress/decompress different parts of image/frames
- Desktop workloads…
- Gaming workloads…

What would you do with 2 billion transistors?

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 Niagra 3 (16), AMD Quad Phenon (4), Sun Ultra Enterprise (72)
- message passing
  - explicit sends & receives
  - IBM BlueGene/L (64), Intel Paragon

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 error-free parallelism** (hardest problem)

- 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
- HW support for maintaining correctness (today’s efforts)

Application 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

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 or data layout

Shared Memory vs. Message Passing

Shared memory

- + low latency (no message passing software) ***but***
  - *overlap of communication & computation*
  - *latency-hiding techniques can be applied to message passing machines*
- + higher bandwidth for small transfers ***but***
  - *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 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?

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
    sloooooooooow

Who won?

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, today’s streaming processors (e.g., the ATI GPU (320))

MIMD: multiple instruction streams, multiple data streams

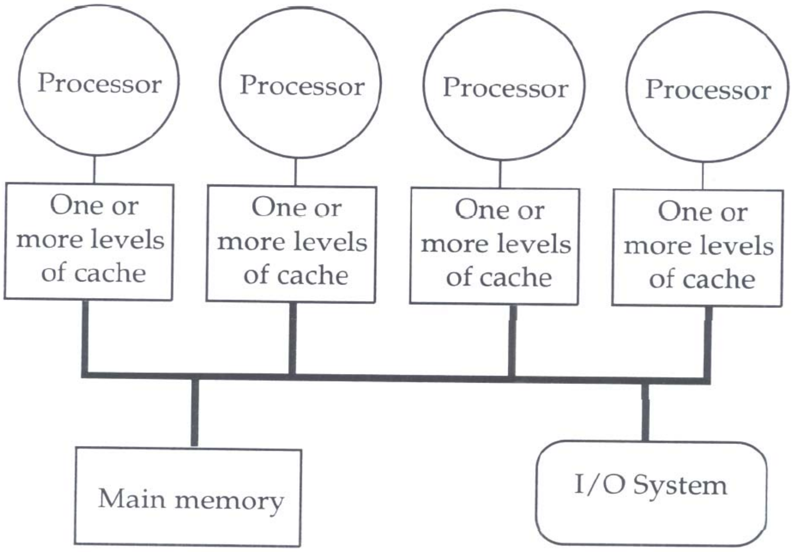
- multiprocessors
- multithreaded processors
- parallel programming **&** multiprogramming
- relies on control parallelism: execute & synchronize different asynchronous threads of control
- example: most processor companies have CMP configurations

MIMD

Low-end

- bus-based
  - simple, but a bottleneck
  - simple cache coherency protocol
- physically centralized memory
- uniform memory access (UMA machine)
- most of today’s CMPs (SunFire (16))

Low-end MP

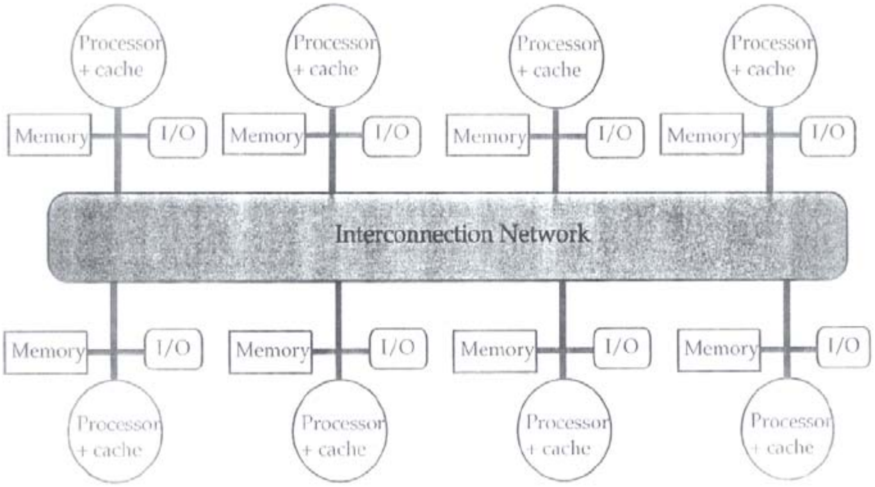

Processor
Processor
Processor
Processor
One or more levels of cache
One or more levels of cache
One or more levels of cache
One or more levels of cache
Main memory
I/O System

MIMD

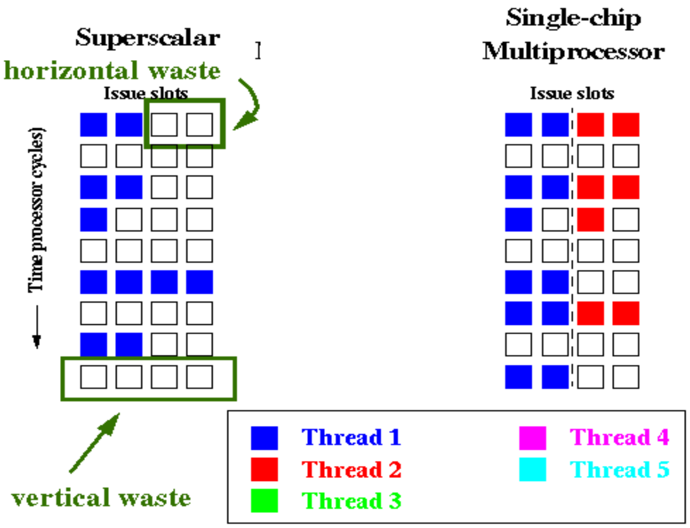
High-end

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

High-end MP



Comparison of Issue Capabilities


Superscalar
horizontal waste
Issue slots
Time (processor cycles)
vertical waste
Single-chip
Multiprocessor
Issue slots
🟦 Thread 1
🟥 Thread 2
🟩 Thread 3
Thread 4
Thread 5