Parallel Computer Architectures

Jin-Soo Kim (jinsoookim@skku.edu)  
Computer Systems Laboratory  
Sungkyunkwan University  
http://csl.skku.edu
Exploiting Parallelism

- **Instruction-Level Parallelism**
  - Pipelining, superscalar, speculation, branch prediction, OOO, VLIW, ...

- **Task-Level (or Thread-Level) Parallelism**
  - Simultaneous Multithreading
  - Multicore
  - Multiprocessors
  - Clusters

- **Data-Level Parallelism**
  - Vector architecture
  - SIMD instructions
Hardware Multithreading (1)

- **Run multiple threads of execution in parallel**
  - Replicate registers, PC, etc.
  - Fast switching between threads

- **Fine-grain multithreading**
  - Switch threads after each cycle
  - Interleave instruction execution
  - If one thread stalls, others are executed

- **Coarse-grain multithreading**
  - Only switch on long stall (e.g., L2-cache miss)
  - Simplifies hardware, but doesn’t hide short stalls (e.g., data hazards)
Simultaneous multithreading (SMT)

- In multiple-issue dynamically scheduled processor
  - Schedule instructions from multiple threads
  - Instructions from independent threads execute when function units are available
  - Within threads, dependencies handled by scheduling and register renaming
- Makes a single physical processor appear as multiple logical processors
- Uses processor resources more effectively
- Example: Intel Pentium 4 HT
  - Two threads: duplicated registers, shared function units and caches
Hardware Multithreading (3)

- Intel HyperThreading

**Logical Processor 0**
- Arch. state (registers)
- Execution units
- Cache(s)

**Logical Processor 1**
- Arch. state (registers)

**System bus**

**Time**

**Execution Units**

- Conventional Superscalar
- Simultaneous Multithreading
Examples

Hardware Multithreading (4)

- Issue slots
  - Thread A
  - Thread B
  - Thread C
  - Thread D

- Time

Issue slots
- Coarse MT
- Fine MT
- SMT
Future of multithreading

- Will it survive? In what form?

- Power considerations $\Rightarrow$ simplified microarchitectures
  - Simpler forms of multithreading
  - e.g., Sun UltraSPARC T2: fine-grained multithreading

- Tolerating cache-miss latency
  - Thread switch may be most effective

- Multiple simple cores might share resources more effectively
Multicore (1)

- Multicore is mainstream

This graph shows a forecast of the percentage of PCs shipping with a processor containing two or more processor cores.

Source:
Processor data: IDC Worldwide PC Semiconductor 2006-2011 Market Forecast
Multicore (2)

Source: IEEE Computer, 2011
Multicore (3)

- Single-processor performance

Source: IEEE Computer, 2011
Multicore (4)

- **Memory wall**
  - CPU 55%/year, Memory 10%/year (1986~2000)
  - Caches show diminishing returns

- **ILP wall**
  - Control dependency
  - Data dependency

- **Power wall**
  - Dynamic power $\propto$ Frequency\(^3\)
  - Static power $\propto$ Frequency
  - Total power $\propto$ The number of cores
Multicore (5)

Source: Intel
Multicore (6)

- **Core types**
  - Homogeneous
  - Heterogeneous

- **Cache organization**
  - Shared L2+ cache
  - Independent L2+ cache

- **On-die interconnects**
  - Bus
  - Crossbar
  - Ring, Mesh, Torus, ...
Parallel Architectures (1)

- **SMP: shared memory multiprocessors**
  - Hardware provides single physical address space for all processors
  - Synchronize shared variables using locks
  - Memory access time: UMA vs. NUMA
Parallel Architectures (2)

- Distributed memory architecture
  - Each processor has private physical address space
  - Hardware sends/receives messages between processors
Clusters

- A network of independent computers
- Each node has private memory and OS
- Connected using I/O system
  - E.g., Ethernet/switch, Internet
- Suitable for applications with independent tasks
  - Web servers, databases, simulations, ...
- High availability, scalable, affordable
- Problems
  - Administration cost (prefer virtual machines)
  - Low interconnect bandwidth
    » cf. processor/memory bandwidth on an SMP
Data-Level Parallelism

- **SIMD (Single-Instruction Multiple-Data) architecture**
  - Operate element-wise on vectors of data
    - E.g., MMX and SSE instructions in x86: Multiple data elements in 128-bit wide registers
  - All processors execute the same instruction at the same time
    - Each with different data address, etc.
  - Simplifies synchronization
  - Reduced instruction control hardware
  - Works best for highly data-parallel applications
Vector Processors (1)

- **Characteristics**
  - Highly pipelined function units
  - Stream data from/to vector registers to units
    - Data collected from memory into registers
    - Results stored from registers to memory

- **Example: Vector extension to MIPS**
  - 32 x 64-element registers (64-bit elements)
  - Vector instructions
    - lv, sv: load/store vector
    - addv.d: add vectors of double
    - addvs.d: add scalar to each element of vector of double
  - Significantly reduces instruction-fetch bandwidth
Vector Processors (2)

- Example: DAXPY ($Y = a \times X + Y$)

  - Conventional MIPS code

```
    l.d  $f0, a($sp) ; load scalar a
    addiu r4, $s0, #512 ; upper bound of what to load
  loop: l.d  $f2, 0($s0) ; load x(i)
    mul.d $f2, $f2, $f0 ; a \times x(i)
    l.d  $f4, 0($s1) ; load y(i)
    add.d $f4, $f4, $f2 ; a \times x(i) + y(i)
    s.d  $f4, 0($s1) ; store into y(i)
    addiu $s0, $s0, #8 ; increment index to x
    addiu $s1, $s1, #8 ; increment index to y
    subu $t0, r4, $s0 ; compute bound
    bne  $t0, $zero, loop ; check if done
```
Vector Processors (3)

- Example: DAXPY \( (Y = a \times X + Y) \) (cont’d)
  - Vector MIPS code

```
l.d  \$f0,a(\$sp)  ;load scalar a
lv   \$v1,0(\$s0)  ;load vector x
mulvs.d \$v2,\$v1,\$f0  ;vector-scalar multiply
lv   \$v3,0(\$s1)  ;load vector y
addv.d \$v4,\$v2,\$v3  ;add y to product
sv   \$v4,0(\$s1)  ;store the result
```
Vector Processors (4)

- Vector architectures and compilers
  - Simplify data-parallel programming
  - Explicit statement of absence of loop-carried dependencies
    - Reduced checking in hardware
  - Regular access patterns benefit from interleaved and burst memory
  - Avoid control hazards by avoiding loops
  - More general than ad-hoc media extensions (such as MMX, SSE)
    - Better match with compiler technology
## Summary

### Flynn’s taxonomy on parallel architectures

<table>
<thead>
<tr>
<th>Instruction Streams</th>
<th>Single</th>
<th>Multiple</th>
</tr>
</thead>
<tbody>
<tr>
<td>SISD: Uniprocessors</td>
<td>SIMD: Vector processors Multimedia extension (Intel MMX/SSE, ...)</td>
<td></td>
</tr>
<tr>
<td>MISD: Systolic array?</td>
<td>MIMD (cf. SPMD): Multicore multiprocessors, MPP, SMP, Cluster, ...</td>
<td></td>
</tr>
</tbody>
</table>