Parallel Computer Architectures

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

- **Instruction-Level Parallelism**
  - Pipelining, superscalar, OOO, branch prediction, 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

![Diagram showing the components of Intel HyperThreading]

- Logical processor 0
- Logical processor 1
- Arch. state (registers)
- Execution units
- Cache(s)
- System bus

Time
Execution Units

Conventional Superscalar
Simultaneous Multithreading
Hardware Multithreading (4)

Examples

- 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 ⇒ 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)

Intel CPU Trends
(sources: Intel, Wikipedia, K. Olukotun)

Moore’s Law

Transistors (000)
Clock Speed (MHz)
Power (W)
Perf/Clock (ILP)
Multicore (3)

- **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 (4)

Source: Intel

More MIPS/watt

1.73x

1.73x

1.00x

1.13x

1.00x

0.87x

0.51x

1.02x

Single-Core

Raise Clock (20%)

Lower Clock (20%)

Dual-Core

Rraise Clock (20%)

PERFORMANCE

POWER

PERFORMANCE

POWER

PERFORMANCE

POWER

PERFORMANCE

POWER

Source: Intel
Multicore (5)

- **Core types**
  - Homogeneous
  - Heterogeneous

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

- **On-die interconnects**
  - Bus
  - Crossbar
  - Ring, Mesh, Torus, ...
Multicore (6)

- Intel dual-core processors
  - Homogeneous cores
  - Bus-based
  - Different cache configurations

![Diagram showing Intel dual-core processors with different configurations: Xeon 3000 (Conroe), Xeon 5000 (Dempsey), Xeon 5100 (Woodcrest), Xeon 5200 (Wolfdale). Each processor has two cores with variations in cache sizes and FSB speeds.]
Multicore (7)

- **Sun UltraSPARC T2 (Niagara2)**
  - Homogeneous
  - Bus-based
  - Shared L2 cache
  - 8 SPARC cores, 8 threads / core
  - 16KB I$ per core, 8KB D$ per core
  - 4MB shared L2, 8 banks, 16-way
Multicore (8)

- STI Cell Broadband Engine
  - Heterogeneous
  - Ring-based
  - One PPE based on 64-bit PowerPC
  - Eight SPEs, each with 256KB Local Store
  - Globally coherent DMA
  - EIB: Four 16B rings
**Multicore (9)**

- **Cavium Octeon CN3860**
  - System-on-a-chip for networking equipment
  - 16 MIPS cores (integer only)
Multicore (10)

- **Tilera Tile64**
  - Homogeneous
  - Mesh-based
  - Independent L2 cache
Multicore (11)

- **ARM Cortex-A9 MPCore**
  - 1 ~ 4 cores
  - ARMv7 architecture
  - L1: 16KB, 32KB, or 64KB
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

![Diagram of SMP architecture]

**Diagram:**
- Multiple processors
- Each processor has its own cache
- Interconnection network
- Memory and I/O components
Parallel Architectures (2)

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

![Diagram of distributed memory architecture](image)
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
    - 1v, 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

```mips
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

```assembly
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
**Intel MMX Technology**

- Introduced in Pentium MMX and Pentium II
- SIMD (Single-Instruction Multiple-Data) execution model
- For media and communications applications
- Eight 64-bit MMX registers (MM0-MM7) are shared with x87 FPU registers (R0-R7)
- Three new packed data types
MMX (2)

- **SIMD execution model**
  - Add, Subtract, Multiply, Multiply and Add, ...

```
Source 1
| X3 | X2 | X1 | X0 |
Source 2
| Y3 | Y2 | Y1 | Y0 |
Destination
| X3 OP Y3 | X2 OP Y2 | X1 OP Y1 | X0 OP Y0 |
```
SSE (1)

- Streaming SIMD Extensions
  - Introduced in Pentium III
  - For advanced 2-D/3-D graphics, motion video, image processing, speech recognition, audio synthesis, telephony, and video conferencing
  - New eight 128-bit XMM registers (XMM0-XMM7)
  - New 32-bit MXCSR register
    - Control and status bits for operations on XMM registers
  - New 128-bit packed single-precision FP data type
SSE (2)

- **SSE packed/scalar single-precision FP operations**
  - Add, Multiply, Divide, Reciprocal, Square root, Reciprocal of square root, Max, Min

- **SSE logical operations**
  - And, Or, Xor
SSE (3)

- **SSE shuffle operation**
  - Any two from src
  - Any two from dest

- **SSE unpack operation**
SSE2

- **Streaming SIMD Extensions 2 (SSE2)**
  - Introduced in Pentium 4 and Intel Xeon
  - For advanced 3-D graphics, speech recognition, video encoding/decoding, E-commerce, Internet, scientific and engineering applications
  - Six new data types
SSE3, SSSE3, SSE4

- **Streaming SIMD Extensions 3 (SSE3)**
  - Introduced in Pentium 4 (Prescott)
  - 13 new instructions

- **Supplemental SSE3 (SSSE3)**
  - Introduced in Intel Xeon 5100 and Intel Core 2
  - 16 new instructions each on MMX or XMM registers

- **Streaming SIMD Extensions 4 (SSE4)**
  - SSE4.1: new 47 instructions (Penryn)
  - SSE4.2: new 7 instructions (Core i7)
## Flynn’s taxonomy on parallel architectures

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