Chapter 7
Multicores, Multiprocessors, and Clusters
Parallel Programming

- Parallel software is the problem
- Need to get significant performance improvement
  - Otherwise, just use a faster uniprocessor, since it’s easier!
- Difficulties
  - Partitioning
  - Coordination
  - Communications overhead
Strong vs Weak Scaling

• Strong scaling: problem size fixed
• Weak scaling: problem size proportional to number of processors
Shared Memory

- **SMP: Symmetric Multiprocessor**
  - Hardware provides single physical address space for all processors
  - Synchronize shared variables using locks
  - Memory access:
    - UMA (uniform) vs. NUMA (nonuniform)
Example: Sum Reduction

- Sum 100,000 numbers on 100 processor UMA
  - Each processor has ID: $0 \leq P_n \leq 99$
  - Partition 1000 numbers per processor
  - Initial summation on each processor
    
    ```
    sum[P_n] = 0;
    for (i = 1000*P_n; i < 1000*(P_n+1); i = i + 1)
      sum[P_n] = sum[P_n] + A[i];
    ```

- Now need to add these partial sums
  - Reduction: divide and conquer
  - Half the processors add pairs, then quarter, ...
  - Need to synchronize between reduction steps
Example: Sum Reduction

```c
half = 100;
repeat
    synch();
    if (half%2 != 0 && Pn == 0)
        sum[0] = sum[0] + sum[half-1];
        /* Conditional sum needed when half is odd;
         Processor0 gets missing element */
    half = half/2; /* dividing line on who sums */
    if (Pn < half) sum[Pn] = sum[Pn] + sum[Pn+half];
until (half == 1);
```
Message Passing

- Each processor has private physical address space
- Hardware sends/receives messages between processors
Loosely Coupled Clusters

• Network of independent computers
  ▪ Each 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
    ▪ c.f. processor/memory bandwidth on an SMP
Sum Reduction (Again)

- Sum 100,000 on 100 processors
- First distribute 1000 numbers to each
  - The do partial sums
    
    ```
    sum = 0;
    for (i = 0; i<1000; i = i + 1)
        sum = sum + AN[i];
    ```

- Reduction
  - Half the processors send, other half receive and add
  - The quarter send, quarter receive and add, ...
Sum Reduction (Again)

- Given send() and receive() operations

```plaintext
limit = 100; half = 100;/* 100 processors */
repeat
    half = (half+1)/2; /* send vs. receive dividing line */
    if (Pn >= half && Pn < limit)
        send(Pn - half, sum);
    if (Pn < (limit/2))
        sum = sum + receive();
    limit = half; /* upper limit of senders */
until (half == 1); /* exit with final sum */
```

- Send/receive also provide synchronization
- Assumes send/receive take similar time to addition
Multithreading

- Performing 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

- 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
- Example: Intel Pentium-4 HT
  - Two threads: duplicated registers, shared function units and caches
Multithreading Example
Future of Multithreading

- Will it survive? In what form?
- Power considerations $\Rightarrow$ simplified microarchitectures
  - Simpler forms of multithreading
- Tolerating cache-miss latency
  - Thread switch may be most effective
- Multiple simple cores might share resources more effectively
Instruction and Data Streams

- An alternate classification

<table>
<thead>
<tr>
<th>Instruction Streams</th>
<th>Single</th>
<th>Multiple</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single</td>
<td>SISD: Intel Pentium 4</td>
<td>SIMD: SSE instructions of x86</td>
</tr>
<tr>
<td>Multiple</td>
<td>MISD: No examples today</td>
<td>MIMD: Intel Xeon e5345</td>
</tr>
</tbody>
</table>
SIMD

- Operate elementwise 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
GPU Architectures

- Processing is highly data-parallel
  - GPUs are highly multithreaded
  - Use thread switching to hide memory latency
    - Less reliance on multi-level caches
  - Graphics memory is wide and high-bandwidth

- Trend toward general purpose GPUs
  - Heterogeneous CPU/GPU systems
  - CPU for sequential code, GPU for parallel code

- Programming languages/APIs
  - DirectX, OpenGL
  - C for Graphics (Cg), High Level Shader Language (HLSL)
  - Compute Unified Device Architecture (CUDA)
Example: NVIDIA Tesla

Streaming multiprocessor

8 × Streaming processors