Lecture 10

Midterm Review
Announcements

• Testing time – final choice
  ‣ 7pm to 9pm
Terms and concepts

- Know the definition and significance of ....
- Parallel speedup and efficiency, super-linear speedup, strong scaling, weak scaling
- Amdahl’s law, Gustafson’s “law,” serial bottlenecks
- Strong and weak scaling
- Granularity
- SSE

©2010 Scott B. Baden / CSE 160 / Winter 2010
Streaming SIMD Extensions (SSE)

- SSE (SSE4 on Intel Nehalem), Altivec
- Short vectors: 128 bits (256 bits coming)

\[ r[0:3] = a[0:3] \times b[0:3] \]

```c
float a[N], b[N], c[N];
for (int i=0; i<N; i++)
a[i] = b[i] \times c[1];
```

Jim Demmel
Terms and concepts

- SPMD, MIMD, SIMD
- Multiprocessors and multic绪mers
- NUMAs and SMPs
- Processor Memory Gap
- Cache coherence and consistency
- Snooping
- False sharing
- Data dependencies, loop carried dependence
- Critical sections, race conditions
Memory hierarchy pyramid

- CPU Register
- Cache
  - Level 1
  - Level 2
- RAM
  - Physical RAM
  - Virtual Memory

Storage Device Types
- ROM/BIOS
- Removable Drives
- Network/Internet Storage
- Hard Drive

Input Sources
- Keyboard
- Mouse
- Removable Media
- Scanners/Camera/Mic/Video
- Remote Source
- Other Sources

Temporary Storage Areas

Permanent Storage Areas
Intel Clovertown Memory Hierarchy

- Ieng-203
- Intel Xeon X5355 (Intro: 2006)
- Two “Woodcrest” dies on a multichip module

Line Size = 64B (L1 and L2)

Access latency (clocks)

<table>
<thead>
<tr>
<th>Core2</th>
<th>Core2</th>
</tr>
</thead>
<tbody>
<tr>
<td>32K L1</td>
<td>32K L1</td>
</tr>
<tr>
<td>4MB Shared L2</td>
<td>4MB Shared L2</td>
</tr>
</tbody>
</table>

Associativity

<table>
<thead>
<tr>
<th>Core2</th>
<th>Core2</th>
</tr>
</thead>
<tbody>
<tr>
<td>32K L1</td>
<td>32K L1</td>
</tr>
<tr>
<td>4MB Shared L2</td>
<td>4MB Shared L2</td>
</tr>
</tbody>
</table>

8
16

* Software-visible latency will vary depending on access patterns and other factors

Sam Williams et al.
Nehalem’s Memory Hierarchy

- Source: *Intel 64 and IA-32 Architectures Optimization Reference Manual*, Table 2.7

<table>
<thead>
<tr>
<th>Latency (cycles)</th>
<th>Associativity</th>
<th>Line size (bytes)</th>
<th>Write update policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non- inclusive</td>
<td>4</td>
<td>8</td>
<td>Writeback</td>
</tr>
<tr>
<td>Non- inclusive</td>
<td>10</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Inclusive</td>
<td>35+</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>4MB for Gainestown</td>
<td>8</td>
<td>64</td>
<td></td>
</tr>
</tbody>
</table>

realworldtech.com
Why do we look at numerically intensive applications?

- Highly repetitive computations are prime candidates for parallel implementation
- Tight loop nests are the “low hanging fruit”
- Many real world applications
  - Data Mining
  - Image processing
  - Simulations – financial modeling, weather, biomedical

- We can classify applications according to Patterns of communication and computation that persist over time and across implementations

  Phillip Colella’s 7 Dwarfs

Courtesy of Randy Bank
Classifying the application domains

- Patterns of communication and computation that persist over time and across implementations
- Structured grids
  - Lab #2
- Dense linear algebra:
  - Matrix multiply, Vector-Mtx Mpy Gaussian elimination
- N-body methods
- Sparse linear algebra
  - In a sparse matrix, we take advantage of knowledge about the locations of non-zeros, improving some aspect of performance
- Unstructured Grids
- Spectral methods (FFT)
- Monte Carlo

Courtesy of Randy Bank

©2010 Scott B. Baden / CSE 160 / Winter 2010
Application-specific knowledge is important

• There currently exists no tool that can turn a serial program into an efficient parallel program
  
  … for all applications … all of the time… on all hardware

• The more we know about the application…
  … specific problem … math/physics … initial data …
  … context for analyzing the output…
  … the more we can improve productivity

• Issues
  ‣ Data motion and locality
  ‣ Load balancing
  ‣ Serial sections
Sparse matrix vector multiply

// y[i] += A[i,j] × x[j]

#pragma parallel for schedule
dynamic,chunk

for i = 0 : N-1 // rows
    i0 = ptr[i]
    i1 = ptr[i+1] – 1
    for j = i0 : i1 // cols
        y[ ind[j] ] +=
            val[ j ] * x[ ind[j] ]
    end j
end i
Implementation techniques

- pthreads
- OpenMP
- Mutexes and barriers
- Processor geometry
- Layouts: BLOCK, Cyclic, Block Cyclic
- Elimination data dependencies
- Red-Black/ Odd-even
OpenMP II: Low control

```c
printf("Start\n");
N = 1000;

#pragma omp parallel for
for (i=0; i<N; i++)
    A[i] = B[i] + C[i];

M = 500;

#pragma omp parallel for
for (j=0; j<M; j++)
    p[j] = q[j] - r[j];

printf("Finish\n");
```
Layouts

[Block, *]

[*, Block]

[Block, Block]

[Cyclic, *]

[Cyclic, Cyclic]

[Cyclic(2), Cyclic(2)]
Work sharing paradigms

- Task parallelism
- Data parallel
- Self scheduling
Multithreading in perspective

• Benefits
  ‣ Harness parallelism to improve performance
  ‣ Ability to multitask to realize concurrency, e.g. display

• Pitfalls
  ‣ Program complexity
    • Partitioning, synchronization, complicated parallel control flow
    • Data dependencies
    • Shared vs. local state (globals like errno)
    • Thread-safe code, re-entrant functions, protect the use of globals
  ‣ New aspects of performance
    • Complicated locality models
  ‣ New aspects of debugging
    • Race conditions
    • Deadlock
Under the hood of a race condition

- Consider this statement, assume \( x == 0 \)
  \[ x = x + 1; \]
- Generated code
  - \( r1 \leftarrow (x) \)
  - \( r1 \leftarrow r1 + #1 \)
  - \( r1 \rightarrow (x) \)
- Possible interleaving with two threads
  
  \[
  \begin{align*}
  P1 & \\
  r1 & \leftarrow x \\
  r1 & \leftarrow r1 + #1 \\
  x & \leftarrow r1
  \\
  P2 & \\
  r1 & \leftarrow x \\
  r1 & \leftarrow r1 + #1 \\
  x & \leftarrow r1
  \end{align*}
  \]

  \( r1(P1) \) gets 0
  \( r1(P1) \) set to 1
  \( r1(P2) \) also gets 0
  \( r1(P1) \) set to 1
  \( P1 \) writes its R1
  \( P2 \) writes its R1
Threaded program design and implementation

• Code organization and re-use
• Parallelizing serial code (code reorganization and restructuring)
• How to design with parallelism in mind
• Performance
What will multicore processors look like?

• Still and open book
• Exposed hierarchies - cache coherence limited
• Locality domains
In class exercises
Questions

1. Time constrained scaling
2. Tree Summation
3. Synchronization
4. Iteration to thread mapping
5. Printing the letters of the alphabet
6. Removing data dependencies
7. Dependence analysis
8. Performance
1. Time constrained scaling

- Sum $N$ numbers on $P$ processors
- Let $N \gg P$
- Determine the largest problem that can be solved in time $T=10^4$ time units on 512 processors
- Let time to perform one addition = 1 time unit
- Let $\beta =$ time to add a value inside a critical section
Performance model

• Local additions: \( N/P - 1 \)
• Reduction: \( \beta (\lg P) \)
• Since \( N \gg P \)
  \[ T(N,P) \sim \left( \frac{N}{P} \right) + \beta (\lg P) \]
• Determine the largest problem that can be solved in time \( T = 10^4 \) time units on \( P=512 \) processors, \( \beta = 1000 \) time units
• Constraint: \( T(512,N) \leq 10^4 \)
  \[ \Rightarrow \left( \frac{N}{512} \right) + 1000 (\lg 512) \]
  \[ = \left( \frac{N}{512} \right) + 1000*(9) \leq 10^4 \]
  \[ \Rightarrow N \leq 5 \times 10^5 \text{ (approximately)} \]
2. Tree Summation

- Input: an array $x[]$, length $N >> P$
- Output: Sum of the elements of $x[]$
- Goal: Compute the sum in $\lg P$ time
  
  ```
  sum = 0;
  for i=0 to N-1
    sum += x[i]
  ```

- Assume $P$ is a power of 2, $K = \lg P$
- Starter code
  ```
  for m = 0 to K-1 {
  }
  ```
Visualizing the Summation

Thread 0
Thread 2
Thread 4
Thread 6

0 1 2 3 4 5 6 7
0+1 2+3 4+5 6+7
0...3 4..7
0..7
3. Synchronization

List all possible outputs that result from running the following

```
#pragma omp parallel for shared(j,k)
for ( int i=0, j=0, k=0; i< 5; i++ )
    j = j + 10;
    k = j + 10;
}
cout << "k = " << k << endl;
```
4. Iteration to thread mapping

```c
#pragma omp parallel shared(N, iters) private(i)
#pragma omp for
for (i = 0; i < N; i++)
    iters[i] = omp_get_thread_num();
N = 9, # of openMP threads = 3
0 0 0 1 1 1 2 2 2
N = 16, # of openMP threads = 4, schedule(static,2)
0 0 1 1 2 2 3 3 0 0 1 1 2 2 3 3
N = 9: 0 0 1 1 2 2 0 0 1
N = 16, # of openMP threads = 4, schedule(dynamic,2)
3 3 0 0 1 1 2 2 3 3 3 3 3 3 3
2 2 3 3 0 0 1 1 2 2 2 2 2 2 2
```

©2010 Scott B. Baden / CSE 160 / Winter 2010
5. Printing the letters of the alphabet

#pragma omp parallel private(i){
    int LettersPerThread = 26 / omp_get_num_threads();
    int ThisThread = omp_get_thread_num();
    int Start = 'a' + ThisThread * LettersPerThread;
    int End = 'a' + (ThisThreadNum + 1) * LettersPerThread;
    for (i = Start; i < End; i++)
        printf ("%c", i);
}

©2010 Scott B. Baden / CSE 160 / Winter 2010
6. Removing data dependencies

- B initially: 0 1 2 3 4 5 6 7
- B on 1 thread: 0 9 10 11 12 21 22 23
- B on 2 threads: 0 17 18 19 12 13 14 15
- How can we split into 2 loops so that each loop parallelizes, the result it correct?

```plaintext
for i = 1 to N
    B[i] += B[N-i];
```
Splitting a loop

• For iterations \(i=\frac{N}{2}+1\) to \(N\), \(B[200-i]\) reference newly computed data
• All others reference “old” data
• \(B\) initially: \(0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\)
• Correct result: \(0\ 9\ 10\ 11\ 12\ 21\ 22\ 23\)

\[
\begin{align*}
\text{for } i &= 1 \text{ to } N \\
B[i] &= B[N-i];
\end{align*}
\]

\[
\begin{align*}
\text{for } i &= \frac{N}{2}+1 \text{ to } N \\
B[i] &= B[N-i];
\end{align*}
\]
7. Loop Dependence Analysis

- Can we run parallelize these with openmp?

1. for i = 1 to N
   A[i] = A[i] + B[i-1];

2. for i = 1 to N

3. A[0] = 0;
   for i = 1 to N-1

4. for i = 1 to N{
   A[i] = B[i];
   C[i] = A[i] + B[i];
   E[i] = C[i+1];
}
Nested loop dependence analysis

• Can we parallelize the inner loops as shown?

**LOOP #1**
for \( j = 0 \) to \( n-1 \)
   for \( i = 0 \) to \( n-1 \)
     \[ A[i, j+1] = A[i, j]; \]

**LOOP #2**
for \( j = 0 \) to \( n-1 \)
   for \( i = 0 \) to \( n-1 \)
     \[ A[i, j+1] = A[i, j]; \]
8. Performance

- You observe the following running times for a parallel program running a fixed workload \( N \)
- Assume that the only losses are due to serial sections
- What is the speedup and efficiency on 8 processors?
- What will the running time be on 4 processors?
- What is the maximum possible speedup on an infinite number of processors?
- What fraction of the total running time on 1 processor corresponds to the serial section?
- What fraction of the total running time on 2 processors corresponds to the serial section?

<table>
<thead>
<tr>
<th>NT</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10000</td>
</tr>
<tr>
<td>2</td>
<td>6000</td>
</tr>
<tr>
<td>8</td>
<td>3000</td>
</tr>
</tbody>
</table>