Lecture 20

Final Exam Review
Announcements

• Final examination
  ‣ Thursday, March 17, in this room: 3pm to 6pm
  ‣ You may bring your textbook and one piece of notebook sized paper

• Office hours during examination week
  ‣ Wednesday
    • 11AM to 12 noon
    • 4pm to 5pm
  ‣ Or by appointment
Terms and concepts

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

• SPMD, MIMD, SIMD
• NUMAs and SMPs
• Processor Memory Gap
• Cache coherence and consistency
• Snooping
• False sharing
• Data dependencies, loop carried dependence
• Critical sections, race conditions
• SSE (Vectorization)
• GPU
Streaming SIMD Extensions (SSE)

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

```c
float a[N], b[N], c[N];
for (i=0; i<N; i++)
a[i] = b[i] * c[i];
```

\[ r[0:3] = a[0:3]*b[0:3] \]
Address Organization

- Multiprocessors and multicomputers
- Shared memory, message passing
Interconnect

- Interconnection networks: hypercube, ring, mesh, k-ary d-cube, diameter and bisection bandwidth
- Embedding: ring into hypercube
- How many parallel/unique paths between any two nodes in a hypercube?
- Message startup, half power point $n_{1/2}$, peak bandwidth
α-β model of communication time

- Triton.sdsc.edu

\[ \alpha = 3.2 \ \mu\text{sec} \]

\[ \beta_\infty = 1.12 \ \text{GB/sec} \]

\[ @N = 8\text{MB} \]

\[ N_{1/2} \approx 20 \ \text{KB} \]
Memory hierarchy pyramid
Intel Clovertown Memory Hierarchy

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

Access latency (clocks)

3
14*

Associateativity

8
16

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

Line Size = 64B (L1 and L2)

Sam Williams et al.

©2010 Scott B. Baden / CSE 160 / Winter 2010
Algorithms

• Sorting
  ‣ Enumeration sort, odd-even sort, bucket sort, sample sort
• Numerical integration
• Mandelbrot set
• Stencil methods
  ‣ Ghost cells, partitioning
• Matrix multiplication: SUMMA
• LU Decomposition (Gaussian elimination)
  ‣ Cyclic decompositions
Why numerically intensive applications?

• Highly repetitive computations are prime candidates for parallel implementation

• Improve quality of life, economically and technologically important
  › 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 and #4
  - 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

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

- There currently exists no tool that can convert 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
Parallel program design and implementation

- Code organization and re-use
- Parallelizing serial code (code reorganization and restructuring)
- How to design with parallelism in mind
- Performance
SPMD Implementation techniques

- Pthreads and OpenMP
- MPI
- Owner compute rule
- Processor geometry and data distribution
  - BLOCK
  - Cyclic
  - Block Cyclic

©2010 Scott B. Baden / CSE 160 / Winter 2010
Blocked Decompositions

- [Block, *]
- [*, Block]
- [Block, Block]
- [Cyclic, *]
- [Cyclic, Cyclic]
- [Cyclic(2), Cyclic(2)]
As with LU decomposition:

All processors should get about the same amount of work if we choose a reasonable chunk size.

What are the tradeoffs?
Implementation techniques - Multitreading

- Mutexes and barriers
- Static and dynamic scheduling
- Critical sections
OpenMP Flow control

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”);
Under the hood of a race condition

• Consider this statement, assume \( x == 0 \)
  \[
  x = x + 1;
  \]

• Generated code
  
  1. \( r_1 \leftarrow (x) \)
  2. \( r_1 \leftarrow r_1 + #1 \)
  3. \( r_1 \rightarrow (x) \)

• Possible interleaving with two threads
  
  - **P1**
    1. \( r_1 \leftarrow x \)
    2. \( r_1 \leftarrow r_1 + #1 \)
    3. \( x \leftarrow r_1 \)
  
  - **P2**
    1. \( r_1 \leftarrow x \)
    2. \( r_1 \leftarrow r_1 + #1 \)
    3. \( x \leftarrow r_1 \)

  \( r_1(P1) \) gets 0
  
  \( r_2(P2) \) also gets 0
  
  \( r_1(P1) \) set to 1
  
  \( r_1(P1) \) set to 1
  
  P1 writes its R1
  
  P2 writes its R1
Workload partitioning paradigms

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

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

• Pitfalls
  ‣ Program complexity
    • Partitioning, synchronization, parallel control flow
    • Data dependencies
    • Shared vs. local state (globals like errno)
    • Thread-safety
  ‣ New storage class: shared
  ‣ New aspects of debugging
    • Race conditions
    • Deadlock
Implementation techniques – Message passing

• Message passing API
  ‣ Blocking and non-blocking communication
  ‣ Collective communication
    • Gather/scatter, All-to-all, reduction, broadcast
    • Allreduce, GatherV, AlltoallV

• MPI: Communicators and message filtering

• Ghost cells
Messaging system design and implementation

- Eager vs. rendezvous communication
- Hypercube algorithms for collectives
- Message buffering, completion
Message passing in perspective

• Benefits
  ‣ Processes communicate explicitly, no anonymous updates
  ‣ Message arrival provides synchronization

• Pitfalls
  ‣ Must replace synchronization with explicit data motion (±)
  ‣ Can’t rely on the cache to move data between processes living on separate processing nodes
  ‣ Ghost cells, collectives
  ‣ Harder to incrementally parallelize code than with threads
What will multicore processors look like?

• Still an open book
• Exposed hierarchies - cache coherence limited
• NUMA
In class exercises
Questions

1. Performance
2. Iteration to thread mapping
3. Load balancing
4. Removing data dependencies
5. Dependence analysis
6. Cache coherence
7. Ghost Cells
1. Performance (Q1)

- We run on 16 processors
- An application with 2 phases
  - Phase 1: perfectly parallelizable, $E_{16} = 100\%$
  - Phase 2: $E_{16} = 25\%$ efficiency on 16 processors
    Serial section: $f$ of the overall running time on one processor (best serial algorithm): $fT_1$
- Express $T_{16}$ in terms of $f$, as a fraction of the form $x/y$
2. 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 3
  2 2 3 3 0 0 1 1 2 2 2 2 2 2 2 2
```

$HW/Examples/Assign$
3. Load Balancing

• Workload

    int Work[1024];
    #pragma omp parallel shared(N,iters) private(i)
    #pragma omp for
    for (i = 0; i < N; i++)
        f(Work[i]);

• All updates take the same amount of time: 1 second
  1. What is the running time on 4 processors?
  2. What is the running time on 5 processors?
  3. Workload distribution scheme?
3a. Load Balancing

• Now, iterations take varying times to update.
• To balance the workloads, we break each array into 32 contiguous chunks.
• 8 of these chunks complete in 10 seconds, and the rest 5 seconds.

a. What is the best possible running time we can expect on 4 processors, using any workload distribution scheme, and ignoring overhead costs?

b. What is the best possible running time we can expect on 5 processors?

c. Describe the workload distribution scheme(s) used to balance the workload
4. 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 $i = N/2+1$ 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

for $i = 1$ to $N$


for $i = 1$ to $N/2$


for $i = N/2+1$ to $N$

5. Loop Dependence Analysis

• Can we run parallelize these with openmp?

1. for (i=0; i<n-1; i++)
   \[ x[i] = x[i+1]; \]

2. for (i=0; i<n-1; i+=2)
   \[ x[i] = x[i+1]; \]

3. for (i=1; i<n; i++)
   \[ x[i] = y[i]; \]

4. for (i=0; i<n-1; i++)
   \[ x[L[i]] = y[K[i]] \]
6. Cache coherence

```cpp
const int BIG_NUMBER = ... 
Shared int sharedVar = 0;

while(sharedVar < BIG_NUMBER){
    CRITICAL SECTION:
        sharedVar++;
}
```

- We run with 2 threads
- We run with 100 threads
- What can prevent threads from updating the shared variable?
7. Ghost cells (Q3)

• Image smoother

repeat until done:
  for \( i = 1: \) N-2
    for \( j = 1: \) N-2
      \[ U^{new}[i, j] = (U[i+1, j] + U[i-1, j] + U[i, j+1] + U[i, j-1]) / 4; \]
      Swap \( U^{new} \) and \( U \)

• What is the max number of ghost cells that a processor will need to transmit when \( P \) does not divides \( N \) evenly?

\[ \frac{N}{\sqrt{(P)}} \]
Good luck!