Lecture 5

Applications:
N-body simulation, sorting, stencil methods
Announcements

• Quiz #1 in section on 10/13
• Midterm: evening of 10/30, 7:00 to 8:20 PM
• In Assignment 2, the following variation is suggested
  – Processors communicate in even/odd pairs
  – Messages in only one direction
  – Use MPI_Sendrecv_replace in lieu of Irecv and Send
MPI_Sendrecv

• Sends then receives a message

\[
\text{MPI\_Sendrecv( void *sbuf, int scount, MPI\_Datatype stype,}
\int dest, int sendtag,
void *rbuf, int rcount, MPI\_Datatype rtype,
int source, int recvtag,
MPI\_Comm comm, MPI\_Status *status )}
\]

• Sends then receives a message using one buffer

\[
\text{MPI\_Sendrecv\_replace( void *buf,}
\int count, MPI\_Datatype datatype,
int dest, int sendtag,
int source, int recvtag,
MPI\_Comm comm, MPI\_Status *status )}
\]
Today’s lecture

• More MPI applications
  – Sorting
  – N-body simulation
  – Stencil methods
Parallel Sorting

• Sorting is fundamental algorithm in data processing
  – Given an unordered set of keys $x_0, x_1, \ldots, x_{N-1}$
  – Return the keys in sorted order
• The keys may be character strings, floating point numbers, integers, or any object for which the relations $>$, $<$, and $==$ hold
• We’ll assume integers here
• Will talk about other algorithms later on
Compare and exchange sorts

• Simplest sort, based on the bubble sort algorithm
• The fundamental operation is compare-exchange
• Compare-exchange(a[j], a[j+1])
  – swaps its arguments if they are in decreasing order
  – satisfies the post-condition that $a[j] \leq a[j+1]$
  – Returns FALSE if a swap was made
    \begin{verbatim}
    for i = N-1 to 1 by -1 do
        done = TRUE;
        for j = 0 to i-1 do // Compare-exchange(a[j], a[j+1])
            if (a[i] < a[j]) { a[i] <-> a[j];
                done=FALSE; }
        end do
    if (done) break;
    end do
    \end{verbatim}
Compare and exchange sorts

- We cannot run bubble sort in parallel owing to the loop carried dependence in the inner loop.
- The value of $a[j]$ computed in iteration $j$ depends on the $a[i]$ computed in iterations $0, 1, ..., j-1$

```plaintext
def for i = N-1 to 1 by -1 do
    done = TRUE;
    for j = 0 to i-1 do
        done = Compare-exchange(a[j], a[j+1])
    end do
    if (done) break;
end do
```
Odd/Even sort

- If we re-order the comparisons we can parallelize the algorithm
  - number the points as even and odd
  - alternate between sorting the odd and even points
- This algorithm parallelizes since there are no loop carried dependences
- All the odd (even) points are decoupled
The algorithm

done = false;

for i = 0 to n-1 do
    for j = 0 to n-1 by 2 do // Even
        done &= Compare-exchange(a[j] , a[j+1]);
    end do

    for j = 1 to n-1 by 2 do // Odd
        done &= Compare-exchange(a[j] , a[j+1]);
    end do
    if (done) break;
end do
Parallelizing the algorithm

• Splitting up the data
• Handling communication
Partitioning

- Partitioning is the process of splitting up the data
- We partition the data into intervals, assigning each to a unique processor
- Many to one mappings can be useful
Communication

- Each interval needs data “owned” by a neighboring process
- At the borders with the neighboring interval
- We add “overlap” or “ghost” cells to hold a copy of the value
- Before each sorting sweep pass messages to refresh the copies
- We can use the original serial loop without having to put communication inside
Odd Even Transposition sort

- While odd/even sort parallelizes, it has a long running time.
- Only one data element at a time is swapped between neighboring processes, and each swaps value can take $N/P$ steps to move across to the next neighboring process.
- A more efficient algorithm moves blocks of data at each step.
- Uses the odd/even ordering, but this time over block numbers (process IDs).
- The fundamental operation is a block *compare-swap*.
The algorithm

• As a pre-processing step, each processor locally sorts its data using a fast serial algorithm like quicksort
• Processes exchange their data in odd-even pairs using block compare and swap
• Each process applies a local merge sort to extract the smallest (largest) N/P values, discards the rest
• What is the running time?
Odd-even merge sort in action

- N values to be sorted
- Treat as four lists of $M = N/4$
- Sort each separately
- Compare and swap
- Compare and swap
- Final sorted list
## Block compare and swap

<table>
<thead>
<tr>
<th>Processor 0</th>
<th>Processor 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1  3  7  9  11</td>
<td>2  4  8  12  14</td>
</tr>
</tbody>
</table>

- Each processor swaps data with its neighbor
  -1  3  7  9  11  2  4  8  12  14

- Sorts the merged list
  -1  2  3  4  7  8  9  11  2  14

- Processor 0 takes 5 smallest values: -1  2  3  4  7
- Processor 1 takes 5 largest values: 8  9  11  12  14
The algorithm

done = false;

for i = 0 to P-1 do

    for p = 0 to P-1 by 2 do  // Even
        done &= Block-Compare-swap(A_p, A_{p+1});
    end do

    for p = 1 to P-1 by 2 do  // Odd
        done &= Block-Compare-swap(A_p, A_{p+1});
    end do

if (done) break;

end do
Using collective communication in sorting

• Everyone needs to know everyone is done sorting
• To do this, we reduce the local flags, and then broadcast the result
• This operation is known as \texttt{Allreduce( )}
• Everyone obtains a copy of the reduced result

\begin{verbatim}
MPI_Allreduce (&l\_done, &done, 1, MPI\_INT, MPI\_SUM, WORLD)
\end{verbatim}
Performance of odd-even merge sort

• There are at most $P$ passes
• Normalize communication cost parameters $\alpha$ and $\beta$ to the cost of performing a compare
• Cost of a swap: $\alpha + \beta (N/P)$
• Running time is
  $$(N/P) \times \log(N/P) + P(\alpha + \beta (N/P) + N/P)$$

Local sort    swap    merge
Shell sort

• We need $O(P)$ passes in odd/even exchange sort
• How does the running time grow if we increase $P$ and $N$ together?
• Let’s configure the processors in a hypercube, and carry out swaps along each dimension of the hypercube
• This new algorithm is called shell sort
Shell sort algorithm

1. Sort locally
2. Compare/split along each dimension in the hypercube

At this point the list is nearly sorted

3. Perform final sorting with another method
   i.e. Odd/Even merge sort
Running time of shell sort

1. Local sort is \(O(N/P) \log N/P\)

2. Compare/split along each dimension in hypercube
   
   for \(i = 1\) to \((\log P)\)
   
   compare/split with partner along dimension \(I\)
   
   Cost = \((N/P) \log P\)

3. Final sorting with odd/even merge sort
   
   \(L\) rounds: \(L \times (N/P)\)
   
   In the worst case \(L = P\)
   
   When does this occur?
Motivating application: the N-body problem

- A classical problem
- Compute trajectories (in time) of a system of N bodies moving under mutual influence
- The bodies can be molecules, planets, stars, charged particles…

![Diagram of N-body problem](image)
Applications

- A force law governs the way the particles interact
- No general analytic (exact) solution when $N > 2$
- Numerical simulations needed
- The value of $N$ typically ranges from thousands to millions, depending on the problem
- See animation:
  http://www.dynamical-systems.org/threebody
Solving the N body problem

while (current time < end time)
    forall bodies i ∈ 1:N
        compute force $F_i$ induced by all bodies j ∈ 1:N
        update position $x_i$ by $F_i \Delta t$
    current time += $\Delta t$
end
Computing the force

• The running time of the computation is dominated by the force computation, so we ignore the push phase.

• The simplest approach is to use the direct method, with a running time of $O(N^2)$

$$\text{Force on particle } i = \sum_{j=0:N-1} F(x_i, x_j)$$

• $F(\cdot)$ is the force law.

• One example is the gravitational force law

$$G \frac{m_i m_j}{r_{ij}^2} \text{ where } r_{ij} = \text{dist}(x_i, x_j)$$

$G$ is the gravitational constant.
A simple parallel algorithm

- Each processor is assigned $N/P$ particles
- Processors circulate a copy of the particles – ring broadcast
- After $P-1$ steps every processor has seen every particle
- Where have we seen this pattern before?
Force laws

- Many force laws are localized in space
- Gravitational interactions decay as $1/r^2$
- Van Der Waals molecular bonding force (Lennart-Jones)

\[
F(r) = \begin{cases} 
  C \left( 2 - \frac{1}{30r} \right) & r < \frac{1}{30} \\
  C \left( \frac{1}{30r} \right)^5 & r \geq \frac{1}{30}, \\
  C = 0.1 & \text{otherwise}
\end{cases}
\]
Implementation

- To speed up the search for nearby particles, sort into a *chaining mesh* (Hockney & Eastwood, 1981)
- Compute forces one mesh box at a time
- Only consider particles in the 8 surrounding cells
- Reduces the cost of ignoring distant particles
- This is an irregular problem

*Jim Demmel, U. C. Berkeley*
Stencil-based methods

• Many physical problems are simulated on a uniform \textit{mesh} in 1, 2 or 3 dimensions
• Values are defined on a discrete set of points
• A \textit{mapping} from ordered pairs to \textit{physical observables} like temperature and pressure
• An application: differential equations
Other Applications

- Connected component labeling on a mesh
- Cellular automata: Conway’s Game of Life
Stencils and molecules

• We call the numerical operator that sweeps over the solution mesh a **stencil operator**
• One of the simplest molecules is the so called “5 point stencil”
• We may also have 9 point stencils and more points in higher dimensions
Conway’s Game of Life

forall (i,j) in 1:N x 1:N

\[ nn = \text{number of neighbors of } u[i,j] \]

\[ u[i,j] = \begin{cases} 
\text{if } u[i,j] \text{ then (nn == 2 or nn == 3)} \\
\text{else (nn == 3)} 
\end{cases} \]
Partitioning

- Partitioning splits up the data onto processors
- We express the different partitions in terms of a *processor geometry*
- For $P$ processors the geometries are of the form $p_0 \times p_1$, where $P = p_0 \times p_1$
- For $P=4$, are 3 possible geometries

\[\begin{array}{cccc}
1 \times 4 & \quad & \quad & \quad \\
\hline
0 & 1 & 2 & 3 \\
\end{array}\]

\[\begin{array}{ccc}
4 \times 1 & & \\
\hline
0 & & \\
1 & & \\
2 & & \\
3 & & \\
\end{array}\]

\[\begin{array}{cc}
2 \times 2 & \\
\hline
0 & 1 \\
2 & 3 \\
\end{array}\]
Ghost cells

- Each partition needs values found on neighboring processors
- Stores them in a boundary region: ghost cells
- Why don’t we communicate individual values?
Periodically refresh the ghost cells
Communication in 1D

- P divides N evenly
- N/P > 2
- For horizontal strips, data are contiguous
  Number of words moved: N
2D Processor geometry

• Assume $\sqrt{P}$ divides $N$ evenly and $N/\sqrt{P} > 2$
• Non-contiguous data
• Number of words moved $4 \times N/\sqrt{P}$
Summing up the communication

- 1-D decomposition
  \[ N \]

- 2-D decomposition
  \[ 4 \times \frac{N}{\sqrt{P}} \]

- When do we prefer one to the other?
The curse of dimensionality

- As we move to higher dimensional spaces
  - Many more possible processor geometries
  - Communication involves higher dimensional arrays
  - The relative fraction of communication increases for a fixed number of unknowns
- In 1D
  - There is only one possible processor geometry
  - Each process communicates at most 2 points
- In 2D
  - There are 1D and 2D geometries
  - Each process communicate a set of 1D arrays
- In D dimensions
  - D different sets of geometries
  - Each process communicates several (D-1) dimensional arrays