CSE 160
Lecture 14

Sorting (II)
Accelerating the particle method
CiLK
Announcements

• No lab session today
Today’s Lecture

• Parallel Sorting (II)
  ‣ Bitonic Sort
  ‣ Merge Sort

• Accelerated particle methods (A3)
Bitonic sort

- Classic parallel sorting algorithm:
  \(O(\log^2 n)\) time on \(n\) processors
- Also used in fast sorting on a GPU
- **Definition:** A *bitonic sequence* is a sequence of numbers \(a_0, a_1...a_{n-1}\) with at most 1 local maximum and 1 local minimum (endpoints wrap around)
  - There exists an index \(i\) where
    \[a_0 \leq a_i \leq a_1 \ldots \leq a_i \text{ and } a_i \geq a_{i+1} \geq a_{i+1} \ldots \geq a_{n-1}\]
  - We may cyclically shift the \(a_k\) while maintaining this relationship
- Merge property: We may merge two bitonic sequences in much the same way as we merge two *monotonic* sequences

\[
\begin{align*}
1,2,4,7,6,0 & \quad \uparrow \quad \downarrow \\
7,6,0,1,2,4 & 
\end{align*}
\]
Splitting property of bitonic sequences

• We can split a bitonic sequence $y$ into two bitonic sequences $L(y)$ and $R(y)$

$L(y) = \langle \min\{a_0, a_{n/2}\}, \min\{a_1, a_{n/2+1}\}, \ldots, \min\{a_{n/2+1}, a_{n-1}\} \rangle$

$R(y) = \langle \max\{a_0, a_{n/2}\}, \max\{a_1, a_{n/2+1}\}, \ldots, \max\{a_{n/2+1}, a_{n-1}\} \rangle$

• See the notes for a proof

All values in $L(y) < R(y)$

$L(y): 3 4 2 1$

$R(y): 7 5 8 9$
Sorting a bitonic sequence is easy

• Split the bitonic sequence \( y \) into two bitonic subsequences \( L(y) \) and \( R(y) \)
• Sort \( L(y) \) and \( R(y) \) recursively
• Merge the two sorted lists
  ‣ Since all values in \( L(y) \) are smaller than all values in \( R(y) \) we don’t need to exchange values in \( L(y) \) and \( R(y) \)
• When \( |L(.)| < 3 \), sorting is trivial
• We designate \( S(n) \) to be the sort on of an \( n \)-element bitonic sequence
Bitonic sort algorithm

• Create a bitonic sequence $y$ from an unsorted list
• Apply the previous algorithm to sort the bitonic sequence
• We need an algorithm to create the bitonic sequence $y$
Additional properties of bitonic sequences

- Any 2 element sequence is a bitonic sequence
- We can trivially construct a bitonic sequence from two monotonic sequences, one sorted in increasing order, the other in decreasing order
Inductive construction of the initial bitonic sequence

- Form matched pairs of 2-element bitonic sequences, pointing up and down \([B(2)]\)
- Trivially merge these into 4-element bitonic sequences
- Now form matched pairs of 4-element sequences \([B(4)]\)
- Apply \(S(4)\) to each sequence, sorting the first upward, the second downward
- Trivially merge into an 8-element bitonic sequence
- Continue until there is just one sequence
Implementing the bitonic sort algorithm

• Create a bitonic sequence $y$ from an unsorted list, $B(n)$
• Apply the previous algorithm to sort the bitonic sequence, $S(n)$
• We use comparators to re-order data
• We use a shuffle exchange network to form $L(y)$ and $R(y)$
  ‣ This network shuffles an n-element sequence by interleaving $x_0$, $x_{n/2}$, $x_1$, $x_{n/2+1}$, …
Comparators

• Given two values x & y, produce two outputs

• For an increasing comparator, the output is \( \min[x,y], \max[x,y] \)

• For a decreasing comparator, the output is \( \max[x,y], \min[x,y] \)
Bitonic merging network

- Converts a bitonic sequence into a sorted sequence

From *Introduction to Parallel Computing*, V. Kumar et al, Benjamin Cummings, 1994
Bitonic conversion network

Converts an unordered sequence into a bitonic sequence

\[ B(4) = S(4) + S(2) \]

From Introduction to Parallel Computing, V. Kumar et al, Benjamin Cummings, 2003
Today’s Lecture

• Parallel Sorting (II)
  ‣ Bitonic Sort
  ‣ Merge Sort

• Accelerated particle methods (A3)
Merge Sort algorithm

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

N values to be sorted

Treat as four lists of $M = N/4$

Sort each separately

Merge

Merge

Final sorted list
Serial Merge

-1 3 7 9 11

Thread 0

2 4 8 12 14

Thread 1

-1 2 3 4 7 8 9 11 2 14

• Merge Step
• Left most thread does the merging
  -1 3 7 9 11 2 4 8 12 14
• Sorts the merged list
  -1 2 3 4 7 8 9 11 2 14
• Parallelism diminishes as we move up the recursion tree
Parallel Merge

- If there are $N = m+n$ elements, then the larger of the recursive merges processes $3N/4$ elements.
Today’s Lecture

• Parallel Sorting (II)
  ‣ Bitonic Sort
  ‣ Merge Sort

• Accelerated particle methods (A3)
Computing the force

• The running time of the computation is dominated by the force computation, which runs in time $O(N^2)$
  
  Force on particle $i = \sum_{j=0}^{N-1} F(x_i, x_j)$
  
  $F(\ )$ is the force law

• We have to compute the distance check between each pair of particles, even if we don’t compute the force

  if $\text{dist}(x_i, x_j) > 0.01$ ⇒ $F(x,y) = 0$

  else

  $F(x,y) = C*(dx,dy)$

  Where

  $C = (0.01/r^2 - 1/ r^3)$

  $r^2 = \max( dx^2 + dy^2 , 10^{-6})$

  $(dx,dy) = ( (x_j - x_i), (y_j - y_i))$
Accelerating the force computation

```c
void apply_forces( particle_t* particles, int n) {
  #pragma omp parallel for shared(particles,n) schedule(dynamic)
  for( int i = 0; i < n; i++ ) {
    particles[i].ax = particles[i].ay = 0;
    for (int j = 0; j < n; j++) {
      dx    = particles[j].x - particles[i].x;
      dy    = particles[j].y - particles[i].y;
      r2    = dx * dx + dy * dy;
      if (r2 <= cutoff)
        particles[i].{ax,ay} += coef * {dx,dy};
    }
  }
}
```
Implementation

• We don’t need to compute all $O(N^2)$ distance tests
• To speed up the search for nearby particles, sort into a *chaining mesh* (Hockney & Eastwood, 1981)
• Compute forces one box at a time
• Consider particles in the 8 surrounding cells only

*Jim Demmel, U. C. Berkeley*