CSE 332: Parallel Sorting

Richard Anderson, Steve Seitz
Winter 2014

Announcements

- Project 3 PartA due Thursday night

Recap

Last week
- simple parallel programs
- common patterns: map, reduce
- analysis tools (work, span, parallelism)
- Amdahl’s Law

Now
- parallel quicksort, merge sort
- useful building blocks: prefix, pack

Parallelizable?

Fibonacci (N)

Parallelizable?

Prefix-sum:

<table>
<thead>
<tr>
<th>input</th>
<th>6</th>
<th>3</th>
<th>11</th>
<th>10</th>
<th>8</th>
<th>2</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>output</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$output[i] = \sum_{0}^{i-1} input[i]$

First Pass: Sum

<table>
<thead>
<tr>
<th>Sum[0,7]:</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
</tr>
</tbody>
</table>

6
First Pass: Sum

2nd Pass: Use Sum for Prefix-Sum

Prefix-Sum Analysis

Parallel Prefix, Generalized

Pack

Pack:

Output array of elements satisfying test, in original order
Parallel Pack?

Pack

<table>
<thead>
<tr>
<th>input</th>
<th>6</th>
<th>3</th>
<th>11</th>
<th>10</th>
<th>8</th>
<th>2</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>output</td>
<td>6</td>
<td>3</td>
<td>2</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

determining which elements to include is easy

determining where each element goes in output is hard
  - seems to depend on previous results

Parallel Pack

1. map test input, output [0,1] bit vector

<table>
<thead>
<tr>
<th>input</th>
<th>6</th>
<th>3</th>
<th>11</th>
<th>10</th>
<th>8</th>
<th>2</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>test</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

test: \( X < 8? \)

2. transform bit vector into array of indices into result array

| pos  | 1 | 2 | 3 | 4 |

3. map input to corresponding positions in output

| output | 6 | 3 | 2 | 7 |   |   |   |   |
  - if (test[i] == 1) output[pos[i]] = input[i]

Parallel Pack Analysis

- Parallel Pack
  1. map: \( O(\text{span}) \)
  2. sum-prefix: \( O(\text{span}) \)
  3. map: \( O(\text{span}) \)
- Total: \( O(\text{span}) \)

Parallel Pack

1. map test input, output [0,1] bit vector

<table>
<thead>
<tr>
<th>input</th>
<th>6</th>
<th>3</th>
<th>11</th>
<th>10</th>
<th>8</th>
<th>2</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>test</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

test: \( X < 8? \)

2. prefix-sum on bit vector

| pos  | 1 | 2 | 3 | 4 |

3. map input to corresponding positions in output

| output | 6 | 3 | 2 | 7 |   |   |   |   |
  - if (test[i] == 1) output[pos[i]] = input[i]

Sequential Quicksort

QuickSort (review):

1. Pick a pivot \( O(1) \)
2. Partition into two sub-arrays \( O(n) \)
   - A. values less than pivot \( O(n) \)
   - B. values greater than pivot \( O(n) \)
3. Recursively sort A and B \( 2T(n/2), \text{avg} \)

Complexity (avg case)

- \( T(n) = n + 2T(n/2) \)
- \( T(0) = T(1) = 1 \)
- \( O(n \log n) \)

How to parallelize?
Parallel Quicksort

QuickSort
1. Pick a pivot \( O(1) \)
2. Partition into two sub-arrays \( O(n) \)
   A. values less than pivot
   B. values greater than pivot
3. Recursively sort A and B in parallel \( T(n/2), \text{avg} \)

Complexity (avg case)
- \( T(n) = n + T(n/2) \)
- \( T(0) = T(1) = 1 \)
- Span: \( O(\ ) \)
- Parallelism (work/span) = \( O(\ ) \)

[Page 19]

Parallel Partition

Partition into sub-arrays
A. values less than pivot
B. values greater than pivot

What parallel operation can we use for this?

[Page 20]

Parallel Quicksort

QuickSort
1. Pick a pivot \( O(1) \)
2. Partition into two sub-arrays \( O(\ ) \) span
   A. values less than pivot
   B. values greater than pivot
3. Recursively sort A and B in parallel \( T(n/2), \text{avg} \)

Complexity (avg case)
- \( T(n) = \bigcdot O(\ ) + T(n/2) \)
- \( T(0) = T(1) = 1 \)
- Span: \( O(\ ) \)
- Parallelism (work/span) = \( O(\ ) \)

[Page 21]

Sequential Mergesort

Mergesort (review):
1. Sort left and right halves \( 2T(n/2) \)
2. Merge results \( O(n) \)

Complexity (worst case)
- \( T(n) = n + 2T(n/2) \)
- \( T(0) = T(1) = 1 \)
- \( O(n \log n) \)

How to parallelize?
- Do left + right in parallel, improves to \( O(n) \)
- To do better, we need to...

[Page 22]
Parallel Merge

How to merge two sorted lists in parallel?

1. Choose median M of left half
2. Split both arrays into < M, >=M
   - how?
3. Do two submerges in parallel

When we do each merge in parallel:
+ we split the bigger array in half
+ use binary search to split the smaller array
+ And in base case we copy to the output array

Parallel Mergesort Pseudocode

```
Merge(arr[], left1, left2, right1, right2, out[], out1, out2)

int leftSize = left2 - left1
int rightSize = right2 - right1
// Assert: out2 - out1 = leftSize + rightSize
if (leftSize + rightSize < CUTOFF)
    sequential merge and copy into out[out1..out2]
int mid = (left2 - left1)/2
binarySearch arr[right1..right2] to find j such that
    arr[j] ≤ arr[mid] ≤ arr[j+1]
Merge(arr[], left1, mid, right1, j, out[], out1, out1+mid+j)
Merge(arr[], mid+1, left2, j+1, right2, out[], out1+mid+j+1, out2)
```
Analysis

Parallel Merge (worst case)
- Height of partition call tree with \( n \) elements: \( O(\quad) \)
- Complexity of each thread (ignoring recursive call): \( O(\quad) \)
- Span: \( O(\quad) \)

Parallel Mergesort (worst case)
- Span: \( O(\quad) \)
- Parallelism (work/span): \( O(\quad) \)

Subtlety: uneven splits

\[
\begin{array}{cccc}
0 & 4 & 6 & 8 \\
\hline
1 & 2 & 3 & 5
\end{array}
\]
- but even in worst case, get a 3/4 to 1/4 split
- still gives \( O(\log n) \) height

Parallel Quicksort vs. Mergesort

Parallelism (work/span)
- quicksort: \( O(n / \log n) \) avg case
- mergesort: \( O(n / \log^2 n) \) worst case