CSE 332: Parallel Algorithms

Richard Anderson
Spring 2016
Announcements

Project 2: Due tonight
Project 3: Available soon
Analyzing Parallel Programs

Let $T_P$ be the running time on $P$ processors.

Two key measures of run-time:

- **Work**: How long it would take 1 processor = $T_1$
- **Span**: How long it would take infinity processors = $T_\infty$

**Speed-up** on $P$ processors: $T_1 / T_P$
Amdahl’s Fairly Trivial Observation

• Most programs have
  1. parts that parallelize well
  2. parts that don’t parallelize at all

• Let $S =$ proportion that can’t be parallelized, and normalize $T_1$ to 1

  $$1 = T_1 = S + (1 - S)$$

• Suppose we get perfect linear speedup on the parallel portion:

  $$T_P = S + (1-S)/P$$

• So the overall speed-up on $P$ processors is (Amdahl’s Law): $T_1 / T_P = 1 / (S + (1-S)/P)$

  $$T_1 / T_\infty = 1 / S$$
Results from Friday

• Parallel Prefix
  – $O(N)$ Work
  – $O(\log N)$ Span

• Quicksort
  – Partition can be solved with Parallel Prefix
  – Overall result
    • $O(N \log N)$ work, $O(\log^2 N)$ Span
Prefix sum

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>6</td>
<td>9</td>
<td>20</td>
<td>30</td>
<td>38</td>
<td>40</td>
<td>47</td>
<td>55</td>
</tr>
</tbody>
</table>

Sum $[0,7]$: 
  Sum$<0$: 
    Sum $[0,3]$: 
      Sum$<0$: 
        Sum $[0,1]$: 
          Sum$<0$: 
    Sum $[2,3]$: 
      Sum$<2$: 
      Sum $[4,5]$: 
        Sum$<4$: 
    Sum $[4,7]$: 
      Sum$<4$: 
    Sum $[6,7]$: 
      Sum$<6$: 

Parallel Partition

• Pick pivot

8 1 4 9 0 3 5 2 7 6

• Pack (test: <6)

1 4 0 3 5 2 8 9 7

• Right pack (test: >=6)
Parallel Quicksort

Quicksort
1. Pick a pivot \( O(1) \)
2. Partition into two sub-arrays \( O(\log n) \) 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)
- \( T(n) = O(\log n) + T(n/2) \)
- \( T(0) = T(1) = 1 \)
- Span: \( O(\log^2 n) \)
- Parallelism (work/span) = \( O(\frac{n}{\log n}) \)
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…
Parallel Merge

How to merge two sorted lists in parallel?
Parallel Merge

1. Choose median M of left half \( \mathcal{O}(\ ) \)
2. Split both arrays into \(< M, \geq M\) \( \mathcal{O}(\ ) \)
   - how?
Parallel Merge

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
// We will assume leftSize > rightSize without loss of generality

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

\[\begin{array}{cccc}
0 & 4 & 6 & 8 \\
1 & 2 & 3 & 5 \\
\end{array}\]
- 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