Implementing Reductions and Broadcasts

Laxmikant V. Kale

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
Types of reductions

• **Scalar Reductions**
  • Input: each processor holds a number
  • Output: at the end of the operation, one designated processor (say 0) has the **sum** of all numbers
  • Instead of sum, one can use any commutative-associative operation
    • Such as Max, min, and, or, product, ..

PE0  | 3  | 26
---|---|---
PE1  | 2  |
PE2  | 12 |
PE3  | 9  |
Types of reductions

• **Vector Reduction**

  • Input: each processor holds an array of numbers
  • Output: at the end of the operation, a designated processor has the sum of all numbers, respectively
    • i.e. \( \text{Sum}[i] = \text{sum}\{A[i]\} \)
  • Instead of sum, one can use any commutative-associative operation
    • Such as Max, min, and, or, product, ..

```
PE0  [3,2,7]  PE1  [1,2,3]  PE2  [4,5,6]  PE3  [7,8,9]
[15,17,25]
```
Types of reductions

- **Scalar (and Vector) All-Reduce**
  - Each processor holds a number
  - At the end of the operation, all processors have the *sum* of all numbers
  - Again, instead of sum, one can use any commutative-associative operation
    - Such as Max, min, and, or, product, ..

<table>
<thead>
<tr>
<th>PE0</th>
<th>3</th>
<th>26</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE1</td>
<td>2</td>
<td>26</td>
</tr>
<tr>
<td>PE2</td>
<td>12</td>
<td>26</td>
</tr>
<tr>
<td>PE3</td>
<td>9</td>
<td>26</td>
</tr>
</tbody>
</table>
Optimizing Reductions and Broadcasts

• Naïve algorithm: all send to PE 0. (O(P))

• Basic Spanning tree algorithm:
  • Organize processors in a k-ary tree
  • Leaves: send contributions to parent
  • Internal nodes: wait for data from all children, add mine,
  • Then, if I am not the root, send to my parent
  • What is a good value of k? Minimize the total time

\[(\alpha + \beta) \cdot k \log_k p\]

• Broadcasts algorithms are similar to reductions, but in reverse direction
• $P = 1024, K = 2$
  \[
  \alpha \cdot 2 \cdot \log_2(1024) = 20\alpha
  \]

• $P = 1024, K = 4$
  \[
  \alpha \cdot 4 \cdot \log_4(1024) = 20\alpha
  \]
Better spanning trees:

- Observation: Only 1 level of the tree is active at a time
  - Also, A PE can’t deal with data from second child until it has finished “receive” of data from 1st.
  - So, second child could delay sending its data, with no impact
  - It can collect data from someone else in the meanwhile
More complex ideas

• Observation: Only 1 level of the tree is active at a time
  • Exploit the actual values of send overhead, latency, and receive overhead
  • The picture on the right is not necessarily optimal for all values of receive overhead and latency. But you can make a more tuned spanning tree for that.

8 processes instead of 7,
But only 3 steps instead of 4
Hypercube topology

• Recursive definition:
  • a single node is a hypercube (of 0 dimensions)
  • Starting with 2 hypercubes, and connecting the corresponding nodes of each to the other, gives you a hypercube (of higher dimension)
Hypercube topology

• Alternative definition
  • A hypercube of dimension $K$ includes $2^K$ nodes, each with a serial number between 0 and $(2^K - 1)$
  • Each node is connected to $K$ neighbors, identified by changing exactly one bit in the $k$-bit representation of the node’s serial number
Hypercube topology

• Alternative definition
  • A hypercube of dimension $K$ includes $2^K$ nodes, each with a serial number between 0 and $(2^k - 1)$
  • Each node is connected to $K$ neighbors, identified by changing exactly one bit in the $k$-bit representation of the node’s serial number
Hypercube based spanning tree

- In each phase $i$, send data to neighbor in $i^{th}$ dimension if its serial number is smaller than mine
- Accumulate data from neighbors until it is my turn to send
- log P phases, with at most one receive per processor per phase
How will you use a virtual hypercube for AllReduce?

• Idea:
  • Send your data to your neighbor in $i^{th}$ dimension, receive from it, and add up
  • Repeat

• This is called dimensional exchange

• It works because it partitions the hypercube in two parts in each step
Reductions with large datasets

• What if \( n \) is large?
  • Example: simpler formulation of molecular dynamics:
    • Each PE has an array of forces for all atoms
    • Each PE is assigned a subset of pairs of atoms
    • Accumulated forces must be summed up across PEs

• New optimizations become possible with large \( n \):
  • Essential idea: use multiple concurrent reductions to keep all levels of the tree busy
  • Divide data (\( n \) bytes) into \( m \) segments of \( n/m \) bytes each
  • Start reduction for each segment.
    • \( n/m \) pipelined phases (i.e. phases overlap in time)
    • \( k \): branching factor, as before
    • So, approximately: \( (\alpha + (n/m)\beta) \cdot k \cdot m \) Instead of \( (\alpha + n\beta) \cdot k \cdot \log p \)
Concurrent reductions: load balancing!

• Leaves of the spanning tree are doing little work
  • Use a different spanning tree for successive reductions in the pipeline
    • E.g. first reduction uses a normal spanning tree rooted at 0, while second reduction uses a mirror-image tree rooted at (P-1)
    • This load balancing improve performance considerably