Contents

• Preliminaries of parallel algorithm design
• Decomposition Techniques
  – Recursive decomposition
  – Data decomposition
  – Exploratory decomposition
  – Hybrid decomposition
Recursive Decomposition

- Divide and conquer strategy
  - Divide a problem into a set of independent subproblems
  - Each subproblem is solved by recursively applying a similar division into smaller subproblems followed by a combination of their results
- Quicksort problem
  - Quicksort divides (splits), but not always evenly
  - Simple implementation uses \( O(n) \) split
  - There are \( O(\log n) \) levels on average
  - The implementation would be \( O(n \log n) \)
  - Needs a parallel split to be fast

Quicksort – Balanced Example

![Quicksort task-dependency graph based on recursive decomposition for sorting a sequence of 12 numbers.](image)
A Recursive Program to Find a Minimum in an Array

1. procedure RECURSIVE.MIN (A, n)
2. begin
3. if (n = 1) then
4. \( m = A[0]; \)
5. else
6. \( lmin := \text{RECURSIVE.MIN}(A, n/2); \)
7. \( rmin := \text{RECURSIVE.MIN}(A[n/2:], n - n/2); \)
8. if \( (lmin < rmin) \) then
9. \( m := lmin; \)
10. else
11. \( m := rmin; \)
12. endelse;
13. endelse;
14. return \( m; \)
15. end RECURSIVE.MIN

Algorithm 3.2 A recursive program for finding the minimum in an array of numbers \( A \) of length \( n \).
Data Decomposition

- Step 1: Partition the data
- Step 2: Partition computations
- Usually tasks on partitions are similar
  - Matrix-matrix multiplication can be partitioned to have each task compute \( \frac{1}{4} \) of the output: similar tasks
- Partitioning can be on input, output or intermediate data or a combination

Figure 3.9 The task-dependency graph for finding the minimum number in the sequence (4, 9, 1, 7, 8, 11, 2, 12). Each node in the tree represents the task of finding the minimum of a pair of numbers.
Partitioning Output Data

- For matrix multiplication each of 4 tasks could compute ¼ of the output matrix
- Each task owns ¼ of C
- Using the same partitioning of C, one can devise alternative decompositions

Partition of Matrix Multiplication

\[
\begin{pmatrix}
A_{1,1} & A_{1,2} \\
A_{2,1} & A_{2,2}
\end{pmatrix}
\begin{pmatrix}
B_{1,1} & B_{1,2} \\
B_{2,1} & B_{2,2}
\end{pmatrix}
\rightarrow
\begin{pmatrix}
C_{1,1} & C_{1,2} \\
C_{2,1} & C_{2,2}
\end{pmatrix}
\]

(a)

Task 1: \(C_{1,1} = A_{1,1}B_{1,1} + A_{1,2}B_{2,1}\)
Task 2: \(C_{1,2} = A_{1,1}B_{1,2} + A_{1,2}B_{2,2}\)
Task 3: \(C_{2,1} = A_{2,1}B_{1,1} + A_{2,2}B_{2,1}\)
Task 4: \(C_{2,2} = A_{2,1}B_{1,2} + A_{2,2}B_{2,2}\)

(b)

Figure 3.10  (a) Partitioning of input and output matrices into 2 x 2 submatrices. (b) A decomposition of matrix multiplication into four tasks based on the partitioning of the matrices in (a).
Matrix Multiplication Tasks

<table>
<thead>
<tr>
<th>Decomposition I</th>
<th>Decomposition II</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 1: $C_{1,1} = A_{1,1}B_{1,1}$</td>
<td>Task 1: $C_{1,1} = A_{1,1}B_{1,1}$</td>
</tr>
<tr>
<td>Task 2: $C_{1,1} = C_{1,1} + A_{1,2}B_{2,1}$</td>
<td>Task 2: $C_{1,1} = C_{1,1} + A_{1,2}B_{2,1}$</td>
</tr>
<tr>
<td>Task 3: $C_{1,2} = A_{1,1}B_{1,2}$</td>
<td>Task 3: $C_{1,2} = A_{1,2}B_{2,2}$</td>
</tr>
<tr>
<td>Task 4: $C_{1,2} = C_{1,2} + A_{1,2}B_{2,2}$</td>
<td>Task 4: $C_{1,2} = C_{1,2} + A_{1,1}B_{1,2}$</td>
</tr>
<tr>
<td>Task 5: $C_{2,1} = A_{2,1}B_{1,1}$</td>
<td>Task 5: $C_{2,1} = A_{2,2}B_{2,1}$</td>
</tr>
<tr>
<td>Task 6: $C_{2,1} = C_{2,1} + A_{2,2}B_{2,1}$</td>
<td>Task 6: $C_{2,1} = C_{2,1} + A_{2,1}B_{1,1}$</td>
</tr>
<tr>
<td>Task 7: $C_{2,2} = A_{2,1}B_{1,2}$</td>
<td>Task 7: $C_{2,2} = A_{2,1}B_{1,2}$</td>
</tr>
<tr>
<td>Task 8: $C_{2,2} = C_{2,2} + A_{2,2}B_{2,2}$</td>
<td>Task 8: $C_{2,2} = C_{2,2} + A_{2,2}B_{2,2}$</td>
</tr>
</tbody>
</table>

Figure 3.11 Two examples of decomposition of matrix multiplication into eight tasks.

Computing Itemset Frequencies

- Database transaction identified by letter
  - Letter might be 1 item to sell: A means shoes
  - Transaction could be one sale of several items
- Itemset is a set of letters
  - AFK could mean sale of shoes, socks and shirt
- Task
  - Given a sequence of transactions and a sequence of itemsets, count the number of transactions matching each itemset
Computing Itemset Frequencies

Partitioning Input Data

- Output partitioning is not always possible
  - finding minimum, sorting
- Input partitioning may be a useful alternative
  - Works well for minimum, sum, etc
- Input is distributed to tasks
- Tasks work on their portion of input data
- Results are combined
- Much like Divide and Conquer
Partitioning Data

Partitioning the intermediate data among the tasks

<table>
<thead>
<tr>
<th>Task 1</th>
<th>Task 2</th>
<th>Task 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>N, O, P, Q</td>
<td>R, S, T, U</td>
<td>V, W, X, Y</td>
</tr>
<tr>
<td>Z, A, B, C</td>
<td>D, E, F, G</td>
<td>H, I, J, K</td>
</tr>
<tr>
<td>L, M, N, O</td>
<td>P, Q, R, S</td>
<td>T, U, V, W</td>
</tr>
</tbody>
</table>

Figure 3.13: Some decompositions for computing frequent itemsets in a transaction database

Dr. Joe Zhang

PDC-6: Algorithm Design (1)

15

Partitioning Intermediate Data

- Many algorithms have multiple stages
- Output of one stage is input to another
  - Intermediate data
- Sometimes intermediate data is not explicitly formed as a data structure
  - Could cause some algorithm changes
- Matrix multiplication example*

Dr. Joe Zhang

PDC-6: Algorithm Design (1)

16
Exploratory Decomposition

- Searching a state space for a solution
  - Solving a puzzle
  - Computing moves in a game
- Start in the start state
- Start parallel tasks from different second states
- Kill unfinished tasks when one task finds a solution

15-Puzzle Problem

Figure 3.17  A 15-puzzle problem instance showing the initial configuration (a), the final configuration (d), and a sequence of moves leading from the initial to the final configuration.
15-Puzzle Problem

Exploratory Decomposition Speedup

- Perhaps task 1 is unlucky and task 4 is lucky
  - Exploratory solution is great, speedup is ~4
- Perhaps task 1 is lucky
  - Exploratory solution wastes effort, no speedup

Figure 3.19 An illustration of anomalous speedups resulting from exploratory decomposition.

Difference between data decomposition and exploratory decomposition?
Data Decomposition vs. Exploratory Decomposition

- **Data decomposition**
  - The tasks are performed in their entirety
  - Each task performs useful computations towards the solution
  - No task will be killed

- **Exploratory decomposition**
  - The aggregate amount of work performed by a parallel formulation can be either smaller or greater than that searched by a serial algorithm
  - Once a solution is found by one task, the other tasks will be killed

---

Hybrid Decompositions

![Hybrid Decomposition Diagram]

*Figure 3.21* Hybrid decomposition for finding the minimum of an array of size 16 using four tasks.

Dr. Joe Zhang  
PDC-6: Algorithm Design (1)  
21

Dr. Joe Zhang  
PDC-6: Algorithm Design (1)  
22
Summary

• Decomposition Techniques
  – Recursive decomposition
  – Data decomposition
  – Exploratory decomposition
  – Hybrid decomposition

CSC630/CSC730: Parallel Computing

Questions?