Principles of Parallel Algorithm Design (1)

Jinkyu Jeong (jinkyu@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu
Parallel Algorithm

- To solve a problem on multiple processors
- Typical parallel algorithm design
  - Identifying portions of work can be performed concurrently
  - Mapping concurrent pieces of work onto multiple processes running in parallel
  - Distributing input, output and intermediate data
  - Managing accesses to data shared by multiple processors
  - Synchronizing processors at various stages
- Some steps can be changed or omitted
Topics

- Introduction to Parallel Algorithms
  - Tasks and Decomposition
  - Processes and Mapping
  - Processes vs. Processors

- Decomposition Techniques
  - Recursive Decomposition
  - Data Decomposition
  - Exploratory Decomposition
  - Hybrid Decomposition

- Characteristics of Tasks and Interactions
  - Task Generation, Granularity, and Context
  - Characteristics of Task Interactions.
Topics – con’t

- Mapping Techniques for Load Balancing
  - Static and Dynamic Mapping

- Methods for Minimizing Interaction Overheads
  - Maximizing Data Locality
  - Minimizing Contention and Hot-Spots
  - Overlapping Communication and Computations
  - Replication vs. Communication
  - Group Communications vs. Point-to-Point Communication

- Parallel Algorithm Design Models
  - Data-Parallel, Work-Pool, Task Graph, Master-Slave, Pipeline, and Hybrid Models
Work Decomposition

- The first step in developing a parallel algorithm
- Divide work into tasks that can run concurrently
- Many different ways of decomposition
- Tasks may be same, different, or even indeterminate sizes
- Tasks can be independent or dependent

![Diagram of work decomposition with tasks T1 to T17 connected by arrows showing dependencies.](image-url)
In most cases, there are dependencies between different tasks
• Certain tasks can only start one some other tasks have finished
  – Ex) producer-consumer relationship

Task dependency can be drawn using directed acyclic graph (DAG)
• Node: task
• Weight of a node: size (load) of a task
• Edge: control dependency between tasks
Example: Dense Matrix-Vector Multiplication

- Computing each element of output vector \( y \) is independent
- Easy to decompose \( \rightarrow \) one task per element in \( y \)
- Observations
  - Tasks share \( b \)
  - No control dependency between tasks
  - Task size is uniform
Example: Database Query Processing

- Consider the execution of the query:
  
  \[
  \text{MODEL = "CIVIC" AND YEAR = 2001 AND (COLOR = "GREEN" OR COLOR = "WHITE")}
  \]

  on the following database:

<table>
<thead>
<tr>
<th>ID#</th>
<th>Model</th>
<th>Year</th>
<th>Color</th>
<th>Dealer</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>4523</td>
<td>Civic</td>
<td>2002</td>
<td>Blue</td>
<td>MN</td>
<td>$18,000</td>
</tr>
<tr>
<td>3476</td>
<td>Corolla</td>
<td>1999</td>
<td>White</td>
<td>IL</td>
<td>$15,000</td>
</tr>
<tr>
<td>7623</td>
<td>Camry</td>
<td>2001</td>
<td>Green</td>
<td>NY</td>
<td>$21,000</td>
</tr>
<tr>
<td>9834</td>
<td>Prius</td>
<td>2001</td>
<td>Green</td>
<td>CA</td>
<td>$18,000</td>
</tr>
<tr>
<td>6734</td>
<td>Civic</td>
<td>2001</td>
<td>White</td>
<td>OR</td>
<td>$17,000</td>
</tr>
<tr>
<td>5342</td>
<td>Altima</td>
<td>2001</td>
<td>Green</td>
<td>FL</td>
<td>$19,000</td>
</tr>
<tr>
<td>3845</td>
<td>Maxima</td>
<td>2001</td>
<td>Blue</td>
<td>NY</td>
<td>$22,000</td>
</tr>
<tr>
<td>8354</td>
<td>Accord</td>
<td>2000</td>
<td>Green</td>
<td>VT</td>
<td>$18,000</td>
</tr>
<tr>
<td>4395</td>
<td>Civic</td>
<td>2001</td>
<td>Red</td>
<td>CA</td>
<td>$17,000</td>
</tr>
<tr>
<td>7352</td>
<td>Civic</td>
<td>2002</td>
<td>Red</td>
<td>WA</td>
<td>$18,000</td>
</tr>
</tbody>
</table>
Example: Database Query Processing

- Task: generating an intermediate table of entries satisfying a particular clause
- Edge: task dependency (output $\rightarrow$ input)
  - $\text{MODEL} = \text{``CIVIC'' AND YEAR = 2001 AND (COLOR = \text{``GREEN'' OR COLOR = \text{``WHITE''}})$

```
<table>
<thead>
<tr>
<th>ID#</th>
<th>Model</th>
<th>Model</th>
<th>Year</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>4523</td>
<td>Civic</td>
<td>Civic</td>
<td>2001</td>
<td>Green</td>
</tr>
<tr>
<td>6734</td>
<td>Civic</td>
<td>Civic</td>
<td>2001</td>
<td>Green</td>
</tr>
<tr>
<td>4395</td>
<td>Civic</td>
<td>Civic</td>
<td>2001</td>
<td>White</td>
</tr>
<tr>
<td>7352</td>
<td>Civic</td>
<td>Civic</td>
<td>2001</td>
<td>White</td>
</tr>
<tr>
<td>7623</td>
<td>Civic</td>
<td>Civic</td>
<td>2001</td>
<td>White</td>
</tr>
<tr>
<td>6734</td>
<td>Civic</td>
<td>Civic</td>
<td>2001</td>
<td>White</td>
</tr>
<tr>
<td>5342</td>
<td>Civic</td>
<td>Civic</td>
<td>2001</td>
<td>Green</td>
</tr>
<tr>
<td>3845</td>
<td>Civic</td>
<td>Civic</td>
<td>2001</td>
<td>Green</td>
</tr>
<tr>
<td>4395</td>
<td>Civic</td>
<td>Civic</td>
<td>2001</td>
<td>Green</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>ID#</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>7623</td>
<td>Green</td>
</tr>
<tr>
<td>9834</td>
<td>Green</td>
</tr>
<tr>
<td>6734</td>
<td>White</td>
</tr>
<tr>
<td>5342</td>
<td>Green</td>
</tr>
<tr>
<td>8354</td>
<td>Green</td>
</tr>
<tr>
<td>3476</td>
<td>White</td>
</tr>
<tr>
<td>3476</td>
<td>Green</td>
</tr>
<tr>
<td>7623</td>
<td>Green</td>
</tr>
<tr>
<td>6734</td>
<td>White</td>
</tr>
<tr>
<td>5342</td>
<td>Green</td>
</tr>
<tr>
<td>8354</td>
<td>Green</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>ID#</th>
<th>Year</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>6734</td>
<td>2001</td>
<td>White</td>
</tr>
<tr>
<td>4395</td>
<td>2001</td>
<td>White</td>
</tr>
</tbody>
</table>
```

Civic AND 2001 AND (White OR Green)
Example: Database Query Processing

- A different way of task decomposition
  - **MODEL = “CIVIC” AND YEAR = 2001 AND (COLOR = “GREEN” OR COLOR = “WHITE”)**

- Different task decompositions may lead to significant differences with respect to parallel performance
Granularity of Task Decompositions

- **Granularity: task size**
  - The number of tasks determines its granularity

- **Fine-grained decomposition**
  - A large number of tasks

- **Coarse-grained decomposition**
  - A small number of tasks

- Fine-grained decomposition: task per element in y
- Coarse-grained decomposition: task per 3 elements in y
Degree of Concurrency

- Definition: number of tasks that can run in parallel
- May change over program execution

- Two metrics
  - Maximum degree of concurrency
    - Largest number of concurrent tasks
  - Average degree of concurrency
    - Average number of concurrent tasks

- Finer task granularity → larger degree of concurrency
- Coarser task granularity → smaller degree of concurrency
Critical Path Length

- Edges in task dependency graph serializes tasks
- Critical path: the longest weighted path in task dependency graph
- Critical path length bounds parallel execution time
Critical Path Length

Example: database query processing task dependency graphs

- Critical path?
- What is the shortest parallel execution time?
- How many processors are needed?
- Maximum degree of concurrency?
- Average degree of concurrency?

Number in each node is load of each task
Critical Path Length

- Example: dense matrix-vector multiplication

- Task dependency graph?
- Critical path?
- What is the shortest parallel execution time?
- How many processors are needed?
- Maximum degree of concurrency?
- Average degree of concurrency?
Limits on Parallel Performance

- Factors affecting parallel execution time
  - Minimum task granularity
    - Dense matrix-vector multiplication $\rightarrow n^2$ tasks
  - Dependencies between tasks
  - Communication overheads
    - Communication between concurrent tasks

- Measures of parallel performance
  - Speedup $= \frac{T_1}{T_p}$
  - Efficiency $= \frac{T_1}{(pT_p)}$
Task Interaction Graph

- **Tasks generally exchange data with others**
  - Example: database query processing
    - Output is passed to other task as input
  - Example: dense matrix-vector multiplication
    - If vector b is not replicated, tasks need to communicate elements of the vector

- **Task interaction graph**
  - Node: task
  - Edge: interaction or data exchange

- Task interaction graphs $\rightarrow$ data dependencies
- Task dependency graphs $\rightarrow$ control dependencies
Task Interaction Graph: Example

- **Sparse matrix-vector multiplication**
  - Task per element of the result vector
  - Only non-zero elements of matrix A participate
  - If vector b is also partitioned across tasks
    - Tasks should communicate to pass elements in b
    - No control dependency between tasks though

![Task Interaction Graph](image)
Task Interaction Graphs, Granularity, & Communication

- Finer task granularity increases associate overhead
- Example: sparse matrix-vector multiplication

- If a task is node 0 → 3 communications + 4 computations
- If a task is node 0, 4, 5 → 5 communications + 15 computations
- Observation: coarser-grain decomposition → smaller communication/computation ratio
Processes and Mapping

- Task: a certain amount of work
- Process (or thread): runnable element on processors
- Generally, # of tasks > # of cores
- Parallel algorithm must map task to processes
- Why processes rather than cores
  - Aggregate tasks into processes
    - Assign a collection tasks and associated data
  - OS maps processes to cores
Processes and Mapping

- Mapping tasks to processes is critical to parallel performance

- Criteria of task-process mapping
  - Task dependency graphs
    - Mapping independent tasks on separate processes
    - Minimum idling and optimal load balancing
  - Task interaction graph
    - Mapping tasks having interactions on the same process
    - Minimum communication
Processes and Mapping

- An appropriate mapping minimizes parallel execution time
  - Mapping independent tasks to different processes
  - Assigning tasks on critical path to processes as soon as possible
  - Minimizing interactions between threads
    - Dense interactions to the same process
  - Criteria often conflict with each other
    - No decomposition makes a single process (no interaction) but no speedup
Mapping example: database query processing

- No nodes in a level have dependency
- Tasks in a single level are assigned to different processes
Decomposition Techniques

- Decomposition
  - A task $\rightarrow$ subtasks (making concurrency)
  - No single recipe that works for all problems

- Commonly used techniques
  - Recursive decomposition
  - Data decomposition
  - Exploratory decomposition
  - Speculative decomposition
Recursive Decomposition

- Suitable to problems that are solved using divide-and-conquer
  - Divide-and-conquer results in natural concurrency

- Steps
  - Decompose a problem into a set of sub-problems
  - Recursively decompose each sub-problem
  - Stop when a desired granularity is reached.
Recursive Decomposition: Quicksort

- Sort a vector
  - Select a pivot
  - Partition the vector into vleft and vright
  - Recursively sort vleft and vright in parallel
Sometimes, algorithm should be changed to make concurrency

Find min in a vector

• Divide a vector into equal halves
• Recursively find min in the two vectors until minimum granularity is reached

1. procedure SERIAL_MIN (A, n)
2. begin
3. min = A[0];
4. for i := 1 to n - 1 do
5. if (A[i] < min) min := A[i];
6. endfor;
7. return min;
8. end SERIAL_MIN

1. procedure RECURSIVE_MIN (A, n)
2. begin
3. if (n = 1) then
4. min := A[0];
5. else
6. lmin := RECURSIVE_MIN (A, n/2);
7. rmin := RECURSIVE_MIN (&(A[n/2]), n - n/2);
8. if (lmin < rmin) then
9. min := lmin;
10. else
11. min := rmin;
12. endelse;
13. endelse;
14. return min;
15. end RECURSIVE_MIN
Recursive Decomposition: Min

- Example: if vector is \{4, 9, 1, 7, 8, 11, 2, 12\}
- Task dependency graph is
Data Decomposition

- **Tasks are decomposed based on data partitioning**
  - Computations are mostly independent of each data partition

- **Steps**
  - Identify the data on which computations are performed
  - Partition this data across various tasks
    - This partitioning induces a decomposition of the problem

- **Data can be partitioned in various ways**
  - Impacts performance of a parallel algorithm

- **Decomposition based on**
  - Output data partitioning
  - Input data partitioning
  - Input + output data partitioning
  - Intermediate data partitioning
Decomposition based on Output Data

- Each element of the output can be computed independently
- Partition the output across tasks
- Each task perform the computation for its output
- Example: dense matrix-vector multiplication

```
A     b
\begin{array}{ccc}
0 & 1 & n \\
\vdots & \ddots & \vdots \\
n-1 & & \end{array}
\begin{array}{c}
\end{array}
```

Task 1

Task 2

Task n

Example: dense matrix-vector multiplication
Decomposition based on Output Data

- n x n Matrix multiplication: $A \times B = C$
- Tasks are decomposed based on the partition (sub-matrices) of $C$

$$
egin{pmatrix}
A_{1,1} & A_{1,2} \\
A_{2,1} & A_{2,2}
\end{pmatrix} \cdot
egin{pmatrix}
B_{1,1} & B_{1,2} \\
B_{2,1} & B_{2,2}
\end{pmatrix} \rightarrow
egin{pmatrix}
C_{1,1} & C_{1,2} \\
C_{2,1} & C_{2,2}
\end{pmatrix}
$$

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: $C_{2,2} = A_{2,1}B_{1,2} + A_{2,2}B_{2,2}$
### Decomposition based on Output Data

- **Finer-grained partitioning of matrix multiplication**
  - Data partitioning != task partitioning

<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>
Decomposition based on Output Data

- Example: count the frequency of item sets in database transactions

- Partition computations by partitioning item sets
  - Each task computes total counts for each item set
Decomposition Based on Input Data

- **Output data decomposition is for**
  - Output can be naturally computed as a function of input
  - In many algorithms, it is not possible to partition output data
    - Ex) sum of a large integer array

- **Decomposition based on input data**
  - Partition input data
  - Computation for each input data
  - “Follow-up computation” to combine the results
### Decomposition Based on Input Data

**Example:** count the frequency of item sets in database transactions

```
<table>
<thead>
<tr>
<th>Database Transactions</th>
<th>Items</th>
<th>Itemset Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, B, C, E, G, H</td>
<td>A, B, C</td>
<td>1</td>
</tr>
<tr>
<td>B, D, E, F, K, L</td>
<td>D, E</td>
<td>3</td>
</tr>
<tr>
<td>A, B, F, H, L</td>
<td>C, F, G</td>
<td>0</td>
</tr>
<tr>
<td>D, E, F, H</td>
<td>A, E</td>
<td>2</td>
</tr>
<tr>
<td>F, G, H, K</td>
<td>C, D</td>
<td>1</td>
</tr>
<tr>
<td>A, E, F, K, L</td>
<td>D, K</td>
<td>2</td>
</tr>
<tr>
<td>G, H, L</td>
<td>C, D, K</td>
<td>0</td>
</tr>
<tr>
<td>D, E, F, K, L</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F, G, H, L</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Database Transactions</th>
<th>Items</th>
<th>Itemset Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, B, C, E, G, H</td>
<td>A, B, C</td>
<td>1</td>
</tr>
<tr>
<td>B, D, E, F, K, L</td>
<td>D, E</td>
<td>2</td>
</tr>
<tr>
<td>A, B, F, H, L</td>
<td>C, F, G</td>
<td>0</td>
</tr>
<tr>
<td>D, E, F, H</td>
<td>A, E</td>
<td>1</td>
</tr>
<tr>
<td>F, G, H, K</td>
<td>C, D</td>
<td>0</td>
</tr>
<tr>
<td>A, E, F, K, L</td>
<td>D, K</td>
<td>1</td>
</tr>
<tr>
<td>B, C, F</td>
<td>B, C, F</td>
<td>0</td>
</tr>
<tr>
<td>G, H, L</td>
<td>C, D, K</td>
<td>0</td>
</tr>
<tr>
<td>D, E, F, K, L</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F, G, H, L</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

- Partition computations by partitioning the input transactions
  - A task computes a local count
  - Aggregate local counts to produce total count
### Decomposition based on Input & Output Data

- **Partition tasks based on input and output data for higher degree of concurrency**
- **Example:** count the frequency of item sets in database transactions

#### Example Tasks

<table>
<thead>
<tr>
<th>Database Transactions</th>
<th>Items</th>
<th>Itemset Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, B, C, E, G, H</td>
<td>A, B, E</td>
<td>1</td>
</tr>
<tr>
<td>B, D, E, F, K, L</td>
<td>D, E</td>
<td>2</td>
</tr>
<tr>
<td>A, B, F, H, L</td>
<td>C, F, G</td>
<td>0</td>
</tr>
<tr>
<td>D, E, F, H</td>
<td>A, E</td>
<td>1</td>
</tr>
</tbody>
</table>

**Task 1**

<table>
<thead>
<tr>
<th>Database Transactions</th>
<th>Items</th>
<th>Itemset Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, B, C, E, G, H</td>
<td>A, B, C</td>
<td>0</td>
</tr>
<tr>
<td>B, D, E, F, K, L</td>
<td>D, E</td>
<td>1</td>
</tr>
<tr>
<td>A, B, F, H, L</td>
<td>C, F, G</td>
<td>1</td>
</tr>
</tbody>
</table>

**Task 2**

<table>
<thead>
<tr>
<th>Database Transactions</th>
<th>Items</th>
<th>Itemset Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, B, C, E, G, H</td>
<td>C, D</td>
<td>0</td>
</tr>
<tr>
<td>B, D, E, F, K, L</td>
<td>D, K</td>
<td>0</td>
</tr>
<tr>
<td>A, B, F, H, L</td>
<td>B, C, F</td>
<td>1</td>
</tr>
<tr>
<td>D, E, F, H</td>
<td>C, D, K</td>
<td>0</td>
</tr>
</tbody>
</table>

**Task 3**

<table>
<thead>
<tr>
<th>Database Transactions</th>
<th>Items</th>
<th>Itemset Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, E, F, K, L</td>
<td>A, E</td>
<td>0</td>
</tr>
<tr>
<td>B, C, D, G, H, L</td>
<td>D, E</td>
<td>1</td>
</tr>
<tr>
<td>G, H, L</td>
<td>C, F</td>
<td>0</td>
</tr>
</tbody>
</table>

**Task 4**

<table>
<thead>
<tr>
<th>Database Transactions</th>
<th>Items</th>
<th>Itemset Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, E, F, K, L</td>
<td>A, E</td>
<td>0</td>
</tr>
<tr>
<td>B, C, D, G, H, L</td>
<td>D, K</td>
<td>0</td>
</tr>
<tr>
<td>G, H, L</td>
<td>B, C, F</td>
<td>0</td>
</tr>
<tr>
<td>D, E, F, K, L</td>
<td>C, D, K</td>
<td>0</td>
</tr>
<tr>
<td>F, G, H, L</td>
<td>C, D</td>
<td>1</td>
</tr>
</tbody>
</table>
**Intermediate Data Partitioning**

- **Computation**
  - A sequence of transformation from the input to the output data.
  - Input data $\rightarrow$ intermediate data $\rightarrow$ output data

- **Partitioning intermediate data for higher concurrency**

- Sometimes, algorithm should be restructured

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

Example: dense matrix multiplication
Intermediate Data Partitioning

- **Example: dense matrix multiplication**
  - Decomposition of intermediate data yields 8 + 4 tasks

**Stage I**

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

**Stage II**

\[
\begin{pmatrix}
D_{1,1,1} & D_{1,1,2} \\
D_{1,2,1} & D_{1,2,2}
\end{pmatrix}
+ \begin{pmatrix}
D_{2,1,1} & D_{2,1,2} \\
D_{2,2,1} & D_{2,2,2}
\end{pmatrix}
\rightarrow
\begin{pmatrix}
C_{1,1} & C_{1,2} \\
C_{2,1} & C_{2,2}
\end{pmatrix}
\]

| Task 01: | \( D_{1,1,1} = A_{1,1} B_{1,1} \) |
| Task 02: | \( D_{2,1,1} = A_{1,2} B_{2,1} \) |
| Task 03: | \( D_{1,1,2} = A_{1,1} B_{1,2} \) |
| Task 04: | \( D_{2,1,2} = A_{1,2} B_{2,2} \) |
| Task 05: | \( D_{1,2,1} = A_{2,1} B_{1,1} \) |
| Task 06: | \( D_{2,2,1} = A_{2,2} B_{2,1} \) |
| Task 07: | \( D_{1,2,2} = A_{2,1} B_{1,2} \) |
| Task 08: | \( D_{2,2,2} = A_{2,2} B_{2,2} \) |
| Task 09: | \( C_{1,1} = D_{1,1,1} + D_{2,1,1} \) |
| Task 10: | \( C_{1,2} = D_{1,1,2} + D_{2,1,2} \) |
| Task 11: | \( C_{2,1} = D_{1,2,1} + D_{2,2,1} \) |
| Task 12: | \( C_{2,2} = D_{1,2,2} + D_{2,2,2} \) |
Intermediate Data Partitioning

- Example: dense matrix multiplication
  - Decomposition of intermediate data yields 8 + 4 tasks

| Task 01: $D_{1,1,1} = A_{1,1} B_{1,1}$ | Task 02: $D_{2,1,1} = A_{1,2} B_{2,1}$ |
| Task 03: $D_{1,1,2} = A_{1,1} B_{1,2}$ | Task 04: $D_{2,1,2} = A_{1,2} B_{2,2}$ |
| Task 05: $D_{1,2,1} = A_{2,1} B_{1,1}$ | Task 06: $D_{2,2,1} = A_{2,2} B_{2,1}$ |
| Task 07: $D_{1,2,2} = A_{2,1} B_{1,2}$ | Task 08: $D_{2,2,2} = A_{2,2} B_{2,2}$ |
| Task 09: $C_{1,1} = D_{1,1,1} + D_{2,1,1}$ | Task 10: $C_{1,2} = D_{1,1,2} + D_{2,1,2}$ |
| Task 11: $C_{2,1} = D_{1,2,1} + D_{2,2,1}$ | Task 12: $C_{2,2} = D_{1,2,2} + D_{2,2,2}$ |

- Task dependency graph
  - Higher concurrency than that of output data decomposition
The Owner Computes Rule

- Each process is assigned a particular data item
- Each process computes values associated with its assigned data item

Implications
- Input data decomposition
  - All computations that use the input data are performed by the process.
- Output data decomposition
  - The output is computed by the process to which the output data is assigned.
Exploratory Decomposition

- Exploration (search) of a state space of solutions
- Problem decomposition reflects the exploration space
  - Partitioning exploration space results in concurrency
  - Decomposition usually happens on-demand during the execution of the algorithm

- Examples
  - Discrete optimization
    - 0/1 integer programming
  - Theorem proving
  - Game playing
**Example: solving a 15 puzzle**

- Sequence of three moves from initial state to final state

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>15</td>
<td>12</td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>15</td>
<td>12</td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

(a) (b) (c) (d)
Exploratory Decomposition

- Example: solving a 15 puzzle

Initial state

Final state
Exploratory Decomposition Speedup

- Parallel formulation may change the amount of work performed
  - Can result in super- or sub-linear speedups

![Diagram]

- Total serial work: 2m+1
  - Total parallel work: 1

- Total serial work: m
  - Total parallel work: 4m
Speculative Decomposition

- Dependencies between tasks are not known a-priori
  - Making it impossible to identify independent tasks
- Conservative approach
  - Identify independent tasks only when they are guaranteed to not have dependencies
- Optimistic (speculative) approach
  - Schedule tasks even when they may potentially be erroneous
  - Ex) branch prediction in processor pipeline
- Drawbacks
  - Conservative approach may yield little concurrency
  - Speculative approach may require roll-back mechanism in the case of an error
Speculative Decomposition

- Example: discrete event simulation

- With a centralized time-ordered event list
  - Extract next event in time order
  - Process the event
  - If required, resulting events are inserted into the event list

- Optimistic event scheduling
  - Assume outcomes of all prior events
  - Speculatively process next event
  - If assumption is incorrect, roll back its effects and continue
Hybrid Decomposition

- Use a mix of decomposition techniques
  - Example: Quicksort
    - Recursive decomposition alone limits concurrency
  - Example: discrete event simulation
    - Some tasks have data-level concurrency
    - Mix speculative decomposition with data decomposition
- Example: find min
  - Data decomposition + recursive decomposition

```
3 7 2 9
11 4 5 8
7 10 6 13
1 19 3 9
```

Data decomposition

Recursive decomposition