My teacher left the room during a test so we all started sharing answers. Then I look up and she was staring right at me 😳🤷‍♂️😢
# Parallel Computing

<table>
<thead>
<tr>
<th>Levels</th>
<th>Components</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applications</td>
<td>Programs (Data and Computations)</td>
<td>1. Pthread, OpenMP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. Pthread, OpenMP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3. CUDA, OpenCL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4. Pthread, OpenMP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5. MPI, socket</td>
</tr>
<tr>
<td>System Software</td>
<td>Libraries</td>
<td>Directives</td>
</tr>
<tr>
<td></td>
<td>Compilers</td>
<td>Parallelization</td>
</tr>
<tr>
<td></td>
<td>Operating Systems : per Node</td>
<td>Compiler ??</td>
</tr>
<tr>
<td></td>
<td>Drivers : per GPU/Co-Processor</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Distributed Systems : multi-node</td>
<td></td>
</tr>
<tr>
<td>Hardware Resources</td>
<td>1. CPU – Multi-core 10s</td>
<td>Replication</td>
</tr>
<tr>
<td></td>
<td>2. SMP – Multi-CPU 10s</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3. GPU – Many-core 1000s</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4. Co-Processors:Many-core 100s</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5. Clusters/Clouds - Many</td>
<td></td>
</tr>
</tbody>
</table>
Exploratory Decomposition

- Decompose problems whose underlying computations correspond to a search of a space for solutions
  - Partition the search space into smaller parts
  - Search each one of them until solutions are found

- Example: The 15-puzzle problem
  - 15 tiles numbered 1 through 15 in a 4 x 4 grid
  - One blank tile
  - Four possible moves: up, down, left, and right
  - The initial and final configurations are specified
  - The objective: determine any sequence or a shortest sequence of moves
The 15-Puzzle Problem

• Solved using tree-search techniques
• Starting from the initial configuration, all possible (2,3,4) successor configurations are generated
  – Occupy the empty slot by one of its neighbors
  – Find a path from one of the new configurations to the final one

• A state space graph: the configuration space generated by the tree search
  – Each node of the graph is a configuration
  – Each edge of the graph connects configurations that can be reached from one another by a single move of a tile
The 15-Puzzle Problem (cont’d)

• A few levels of configurations starting from the initial configuration are generated serially

• Each node is assigned to a task to explore further until at least one of them finds a solution
  – It can inform the others to terminate their search
Exploratory vs. Data Decomposition

• Exploratory decomposition appears similar to data decomposition
  – The search space can be thought of as being the data partitioned

• Differences:
  – Data decomposition: each task performs useful computations towards the solution of the problem
  – Exploratory decomposition: unfinished tasks can be terminated as soon as an overall solution is found
    • The work performed by the parallel formulation can be either smaller or greater than that performed by the serial algorithm

![Diagram showing the comparison between exploratory and data decomposition](image)
Speculative Decomposition

• Used when a program may take one of many possible computationally significant branches depending on the output of preceding computations
  – One task performs the computations whose output will be used in deciding the next computation
  – Other tasks can concurrently start the computations of the next stage

• Similar to evaluating branches in a `switch` statement in C
  – Evaluate multiple branches in parallel
  – Correct branch will be used and other branches will be discarded

• The parallel run time is smaller than the serial run time by the amount of time to evaluate the condition
  – It is used to perform next stage’s computation
  – At least some wasteful computation
    • Only the most promising branch is taken up a task in parallel
      – If different, roll back and take the correct one
Parallel Discrete Event Simulation

- The speedup due to speculative decomposition can add up if there are multiple speculative stages
- Start next stage’s simulation (assuming one of several possible inputs to that stage)
- When the actual input is available:
  - All or part of the work would have been finished, if the speculation was correct
  - The simulation of this stage is restarted with the most recent correct input if the speculation was incorrect
Speculative vs. Exploratory Decomposition

- **What is unknown**
  - In speculative one: the *input at a branch* leading to multiple parallel tasks is unknown
  - In exploratory one: the *output of the multiple tasks* originating at a branch is unknown

- **The amount of work**
  - In speculative one: performs more aggregate work than its serial counterpart
  - In exploratory one: perform more, less, or the same amount of aggregate work depending on the location of the solution in the search space
Hybrid Decompositions

• When a computation is structured into multiple stages, sometimes it’s necessary to apply different types of decomposition in different stages

• Example: finding the minimum of an array
  – Not enough processors when using pure recursive decomposition
Designing a Parallel Algorithm

1. Identify the concurrency available in a problem and decompose it into tasks (executed in parallel)

2. Design a parallel algorithm to assign (map) tasks onto the available processes
   • The nature of the tasks
   • The interactions among tasks
Characteristics of Tasks

- **Task generation**
  - Static: all the tasks are known before the algorithm starts execution
    - Data decomposition: matrix-multiplication, LU factorization
    - Recursive decomposition: finding the minimum of a list of numbers
  - Dynamic: the actual tasks and the task-dependency graph are not explicitly available a priori, although the high level rules or guidelines are known
    - Recursive decomposition: quicksort
      - Tasks are generated dynamically
      - The size and shape of the task tree are determined by the input array
  - Either static or dynamic:
    - Exploratory decomposition: 15-puzzle problem
      - A preprocessing task expands the search tree in a breadth-first manner to generate predefined number of configurations
      - These configurations are mapped and run on processes in parallel, and they can generate dynamic tasks later
Characteristics of Tasks (cont’d)

- **Task sizes:** the relative amount of time required to complete the task
  - Uniform: the tasks require roughly the same amount of time
    - Matrix multiplication
  - Non-uniform: the amount of time required by the tasks varies significantly
    - Quicksort
- **Knowledge of task sizes:** influences the choice of mapping scheme
  - Known: matrix multiplication
  - Unknown: 15-puzzle problem (how many moves to lead to the solution)
- **Size of data associated with tasks:** (location) determines if excessive data-movement overhead will be incurred
  - Small input: 15-puzzle
  - Small output: computing the minimum of a sequence
  - Same order of input/output: Quicksort
Characteristics of Inter-Task Interactions

• Different parallel algorithms $\Rightarrow$ different tasks $\Rightarrow$ different types of interactions

• The nature of interactions $\Rightarrow$ programming paradigms and mapping schemes

• Static versus Dynamic
  – Static: the task-interaction graph and the stage of the computation at which each interaction occurs are known
    • Programmed easily in shared-address-space and message-passing paradigms
    • Matrix multiplication
  – Dynamic: the timing of interactions or the set of tasks to interact with cannot be determined prior to the execution
    • Hard to synchronize senders and receivers in message-passing
      – Additional synchronization or polling responsibility
    • 15-puzzle problem
      – The finished task can pick up an unexplored state from the queue of another busy task and start exploring it
Characteristics of Inter-Task Interactions (cont’d)

- **Regular versus Irregular (spatial structure)**
  - Regular: an interaction pattern has some structure that can be exploited for efficient implementation
    - Image dithering (each pixel weight: values of original one and neighbors)
  - Irregular: no such regular pattern exists
    - Harder to handle, particularly in message-passing paradigm
    - Sparse matrix-vector multiplication (the access pattern for the vector depends on the structure of the sparse matrix)
Characteristics of Inter-Task Interactions (cont’d)

• Read-only versus Read-Write
  – Sharing of data among tasks => inter-task interaction
  – Type of sharing => the choice of the mapping
  – Read-only: tasks require only a read-access to the data shared among many concurrent tasks
    • Matrix multiplication
  – Read-Write: read and write on some shared data
    • 15-puzzle problem (an exhaustive search)
    • Heuristic search: use a heuristic to provide a relative approximate indication of each state from the solution (potential number of moves)
      – The number of tiles that are out of place
    • Priority queue: shared data and tasks (read/write)
      • Put the states resulting from an expansion into the queue
      • Pick up the next most promising state for the next expansion
Characteristics of Inter-Task Interactions (cont’d)

• **One-way versus Two-way**
  - Two-way: the data or work needed by a task or a subset of tasks is explicitly supplied by another task or subset of tasks
    - Predefined producers and consumers
    - Read-write
  - One-way: only one of a pair of communicating tasks initiates the interaction and completes it without interrupting the other one
    - Read-only, read-write
  - Shared-address-space: supports both one-way and two-way interactions equally easily
  - Message-passing: does NOT support one-way interactions
    - The source must explicitly send the data to the recipient
    - Converting one-way to two-way interactions via program restructuring
      - Static: known *a priori* => introducing matching interaction operations
      - Dynamic: restructuring (polling, checking for pending requests after regular intervals)
Mapping Techniques for Load Balancing

• To achieve a small execution time => minimize overheads
• Overheads:
  – Interaction: inter-process interaction
  – Idling: some processes may spend being idle
    • To satisfy the constraints imposed by the task-dependency graph
• Overheads => functions of mapping
• Good mapping:
  – Reducing interaction time
  – Reducing idle time
• Conflicting objectives
  – Mapping tasks onto the same process => unbalanced workload (against concurrency)
  – Balance the load among processes => may cause heavy interactions
Mapping Techniques for Load Balancing (cont’d)

- Assigning a balanced aggregate load of tasks to each process is necessary but not sufficient condition for reducing process idling
- Poor synchronization can lead to idling
  - One task waits to send or receive data from others
- A good mapping: balance both computations and interactions at each stage
Static Mapping

• Mapping: determined by programming paradigm and the characteristics of tasks and interactions

• Statically generated tasks: either static or dynamic

• Static mapping: distribute the tasks among processes prior to the execution of the algorithm

• A good mapping:
  – The knowledge of task sizes
  – The size of data associated with tasks
  – The characteristics of inter-task interactions
  – Parallel programming paradigm

• Optimal mapping for non-uniform tasks: NP-complete
  – Heuristics
Dynamic Mapping

• Distribute the work among processes during the execution
• If tasks are generated dynamically => mapped dynamically
• Unknown task sizes => dynamic mappings are more effective
• Large data associated with the computation
  – Data-movement cost may outweigh other advantages => static
  – May work well in shared-address-space paradigm (read-only)
    • Physical data movement on NUMA and cc-UMA
Schemes for Static Mapping

- Static mapping is often used in conjunction with
  - Data partitioning
  - Task partitioning

- Static mapping is used for mapping certain problems that are expressed naturally by a static task-dependency graph
Array Distribution Schemes

• In a decomposition based on partitioning data, the tasks are closely associated with portions of data
  – The owner-computes rule
  – Mapping data $\Rightarrow$ mapping tasks

• Block distributions: distribute an array and assign uniform contiguous portions of the array to different processes
  – For d-dimensional array: each process receives a contiguous block of array entries along a specified subset of array dimensions
  – Suitable for a case with a locality of interaction
    • Computation of an element requires other nearby elements
One-Dimensional Partitioning

- An $n \times n$ two-dimensional array $A$
- $p$ parts
- Each partition contains a block of $n/p$ consecutive rows/columns of $A$
  - The $k$th part (in row-wise case) contains: $kn/p \ldots (k+1)n/p-1$ (rows)
Multi-Dimensional Partitioning

- 2-D array
- \( p = p_1 \times p_2 \) processes
- Each block: \( \frac{n}{p_1} \times \frac{n}{p_2} \)
- Examples: 4 x 4 and 2 x 8 process grid
- Matrix multiplication \( C = A \times B \): partition \( C \) by block distribution
Higher Dimensional Distribution

• **Allow to use more processes**
  – Matrix multiplication $C = A \times B$
    • One-dimensional distribution: up to $n$
    • Two-dimensional distribution: up to $n^2$

• **Help in reducing the amount of interactions**

• **Example:**
  – One-dimensional partitioning along the rows
    • $n/p$ rows of $A$ and the entire matrix of $B$
    • Total accessed data: $n*(n/p) + n^2 = n^2/p + n^2$
  – Two-dimensional distribution
    • $n/p^{1/2}$ rows of $A$ and $n/p^{1/2}$ columns of $B$
    • Total accessed data: $n*(n/p^{1/2}) + n*(n/p^{1/2}) = O(n^2/p^{1/2})$
      – Smaller than $O(n^2)$ in one-dimensional case
Data Sharing in Matrix Multiplication

(a)

(b)
Dense LU Factorization

- If the amount of work for different elements of a matrix, a block distribution can potentially lead to load imbalances
- A nonsingular square matrix $A \Rightarrow$ product of
  - A lower triangular matrix $L$ with a unit diagonal
  - An upper triangular matrix $U$
  - $A = LU$

\[
\begin{pmatrix}
A_{1,1} & A_{1,2} & A_{1,3} \\
A_{2,1} & A_{2,2} & A_{2,3} \\
A_{3,1} & A_{3,2} & A_{3,3}
\end{pmatrix}
\rightarrow
\begin{pmatrix}
L_{1,1} & 0 & 0 \\
L_{2,1} & L_{2,2} & 0 \\
L_{3,1} & L_{3,2} & L_{3,3}
\end{pmatrix}
\cdot
\begin{pmatrix}
U_{1,1} & U_{1,2} & U_{1,3} \\
0 & U_{2,2} & U_{2,3} \\
0 & 0 & U_{3,3}
\end{pmatrix}
\]

1: $A_{1,1} \rightarrow L_{1,1}U_{1,1}$
2: $L_{2,1} = A_{2,1}U_{1,1}^{-1}$
3: $L_{3,1} = A_{3,1}U_{1,1}^{-1}$
4: $U_{1,2} = L_{1,1}^{-1} A_{1,2}$
5: $U_{1,3} = L_{1,1}^{-1} A_{1,3}$
6: $A_{2,2} = A_{2,2} - L_{2,1}U_{1,2}$
7: $A_{3,2} = A_{3,2} - L_{3,1}U_{1,2}$
8: $A_{2,3} = A_{2,3} - L_{2,1}U_{1,3}$
9: $A_{3,3} = A_{3,3} - L_{3,1}U_{1,3}$
10: $A_{2,2} \rightarrow L_{2,2}U_{2,2}$
11: $L_{3,2} = A_{3,2}U_{2,2}^{-1}$
12: $U_{2,3} = L_{2,2}^{-1} A_{2,3}$
13: $A_{3,3} = A_{3,3} - L_{3,2}U_{2,3}$
14: $A_{3,3} \rightarrow L_{3,3}U_{3,3}$