PRINCIPLES OF PARALLEL ALGORITHM DESIGN

- Dependency graphs. Parallelism in algorithms
- Characteristics on tasks and interactions.
- Decomposition methods
- Mapping and load balancing
Designing parallel programs is all about tasks [which will run in parallel]

Tasks can have fine or coarse granularity

Tasks can communicate with each other ...

... either explicitly as in message passing, or through memory access + some synchronization mechanism - as in shared memory models
The two prevailing programming viewpoints

Shared memory model

- Single program forks tasks
  - Open MP, threads

- Important: in both cases, a standard language (e.g., FORTRAN, C, C++) is augmented by either a communication library (MPI) or directives (openMP).

Message passing model

- Usually single program
  - Multiple Data
- MPI, PVM, ..
Designing parallel programs

There are four main steps:

1. Decomposition into tasks

2. Identify Communication patterns (local, global, broadcast, ...)

3. Agglomeration and load balancing (grouping tasks together to improve performance, simplify coding, achieve balanced work among PEs, ...)


[See Ian Foster “Designing and building parallel programs”]
First issue: identify “tasks” considered as indivisible units of work. “Fine-grain” parallelism: tasks are grouped in very small units, e.g., at the level of arithmetic operations. “Coarse grain” parallelism: tasks are bigger.

Tasks depend on each other: the result of one task may be required by another task.

Important tool: Dependency Graph, which sets the dependencies. There is an (oriented) edge from task $A$ to task $B$ if $B$ cannot be started before $A$ is completed.
A simple example

Consider the following tasks

S1: \( A = B + C \)  
First task
S2: \( B = B \times A \)  
S1 must be done first
S3: \( A = A + 1.0 \)  
S1 must be done first
S4: \( C = A \times 2 + 3 \)  
S3 must be done first
S5: \( X = A + B \)  
S3, S2 must be done first

Dependence graph:
Possible schedule:
(1) S1;
(2) S2, S3;
(3) S4, S5.
Example: triangular system solutions

Example: Solve the linear system of equations

\[ Ax = b \]

where \( A \) has the following structure:

12 tasks altogether [one for solving each equation]
Task number $i$: solve equation number $i$.

Dependency: unknown $i$ requires previous unknowns to which it is coupled by an equation. Thus, $x_6$ requires unknowns $x_5$ and $x_2$.

Dependency graph:

Sets of equations that can be solved in parallel:

$\{1\}, \{2, 5\}, \{3, 6, 9\}, \{4, 7, 10\}; \{8, 11\}; \{12\}$
Some definitions

➤ Degree of concurrency = max number of tasks which can be performed in parallel

➤ Critical path = longest (directed) path between any two nodes in the graph == Longest path from a start node (no incoming edges) to a finish node (no outgoing edges). Path not unique.

➤ Nodes can be weighted to include a cost measure in case the tasks have different costs

➤ Weighted path length = the sum of the weights of the nodes along the path. “Critical path length” = maximum path length. If weights represent time, the algorithm cannot be executed in less time than the critical path length.
Mapping: the mechanism by which tasks are assigned to processors. May also refer to method by which Data is assigned to processors.

Example: For the example of the sparse triangular solve, one can define the weights on the nodes as the number of operations performed to compute $x_i$. Observe that at each node this cost is

$$w_i = 1 + 2 \times (# \text{ incoming edges})$$

What is the critical path length in this case?
Decomposition

Decomposition refers to the process of splitting the initial computation into independent tasks.

Recursive decomposition: Divide and conquer paradigm

Example: Recursive Quick-sort [sorting will be covered in detail later]

Example: Search a data item in an array.
Data decomposition: Concurrency is unraveled by primarily splitting data. Best examples: Adding \( n \) numbers, Adding 2 matrices, product of matrices, ..

Example: Go back to the sum of \( n \) numbers

\[
S = x_1 + x_2 \cdots + x_n = S_1 + S_2 \quad \text{where}
S_1 = x_1 + x_2 \cdots + x_m
S_1 = x_{m+1} + x_{m+2} \cdots + x_n
\]
Example: Matrix-vector product

To compute: \( y = Ax \) where \( A \) is \( m \times n \), \( x \in \mathbb{R}^n \).

Consider row-based algorithm:

\[
y_i = A(i,:) \cdot x
\]

\( y_i \) = Dot product of row \( i \) with \( x \)

Can execute these tasks in parallel

Can also divide the rows into blocks.

We say that we have defined a task with ‘coarser’ granularity.

The result now is a block of \( y \) instead of just one component
In summary: Major types of decomposition

**Data Decomposition**
- Problem divided based on splitting data

**Recursive Decomposition**
- Divide and conquer - e.g. Quicksort

**Exploratory decomposition:**
- Is usually related to problems which involve a search space (best example: game of chess). Different search spaces are explored in parallel by different processors

**Speculative decomposition:**
- In case of branches in the algorithm, different “speculative” branches can be taken by different processors - [see text]
Mapping and load balancing

One of the biggest sources of loss of efficiency is a poor “load balancing”, where some processors are idle while others are busy.

Two two ways to achieve load-balancing:

1. Static mapping: estimate various task and communication costs and schedule tasks statically. Tasks / data are distributed in a certain way at the start (no changes during execution).

2. Dynamic mapping: Tasks / data are distributed/ redistributed during execution.
Static mapping schemes: typically achieved by *mapping the data*.

We will see many such examples when considering matrix algorithms. Another example: Domain decomposition by graph partitioning [see text]

Dynamic scheduling schemes: best example is the ...

**Master-Worker model.** A master processor determines where each task is to be performed. May send off tasks to other processors or may give pointers as to what to do (see next example). May collect results back - or may probe processors to make decisions for next steps. Main point: all decisions, in particular which task performed where, are determined by the Master
The Master-Worker model

Workers
P0 → Send Tasks
P1
P2
P3
P4

Send Tasks → Receive results

Example: A real-life example from Materials Science. Problem is to compute a dense symmetric matrix. Each row is *very* expensive to compute.

▶ Only lower portion of $K$ matrix saved [incomplete rows]
For each row, solve Poisson’s equation, then compute all row entries.
Master-worker model: A master sends indices $i_1, i_2$ of a block to be computed. Many factors are considered to determine the block $i_1 - i_2$. 

Master Proc. Compute $K(j1:j2,:)$

Workers Compute $K(i1:i2,:)$

Save rows of $K$ to disk

Compute $K(j1:j2,:)$ Compute $K(i1:i2,:)$
Task pool model

- Processors take tasks from a global pool (again “global view” but tasks may be distributed). The pool itself can be dynamic.

- Can be entirely decentralized - (distributed decisions) or centralized (bookkeeping of tasks is done in one location).

- Tasks are essentially moved around – so one requirement: little data to be moved along.
Spawn new tasks
Dispatch some to non-busy processors and perform a subset when finished - compare my busy-time with others. Broadcast availability/