

Lecture 4: Principles of Parallel Algorithm Design (part 3)

Exploratory Decomposition

- Decomposition according to a search of a state space of solutions
- Example: the 15-puzzle problem
 - Determine any sequence or a shortest sequence of moves that transforms the initial configuration to the final configuration.

1	2	3	4
5	6	↑	8
9	10	7	11
13	14	15	12

A

1	2	3	4
5	6	7	8
9	10	←	11
13	14	15	12

B

1	2	3	4
5	6	7	8
9	10	11	↑
13	14	15	12

C

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

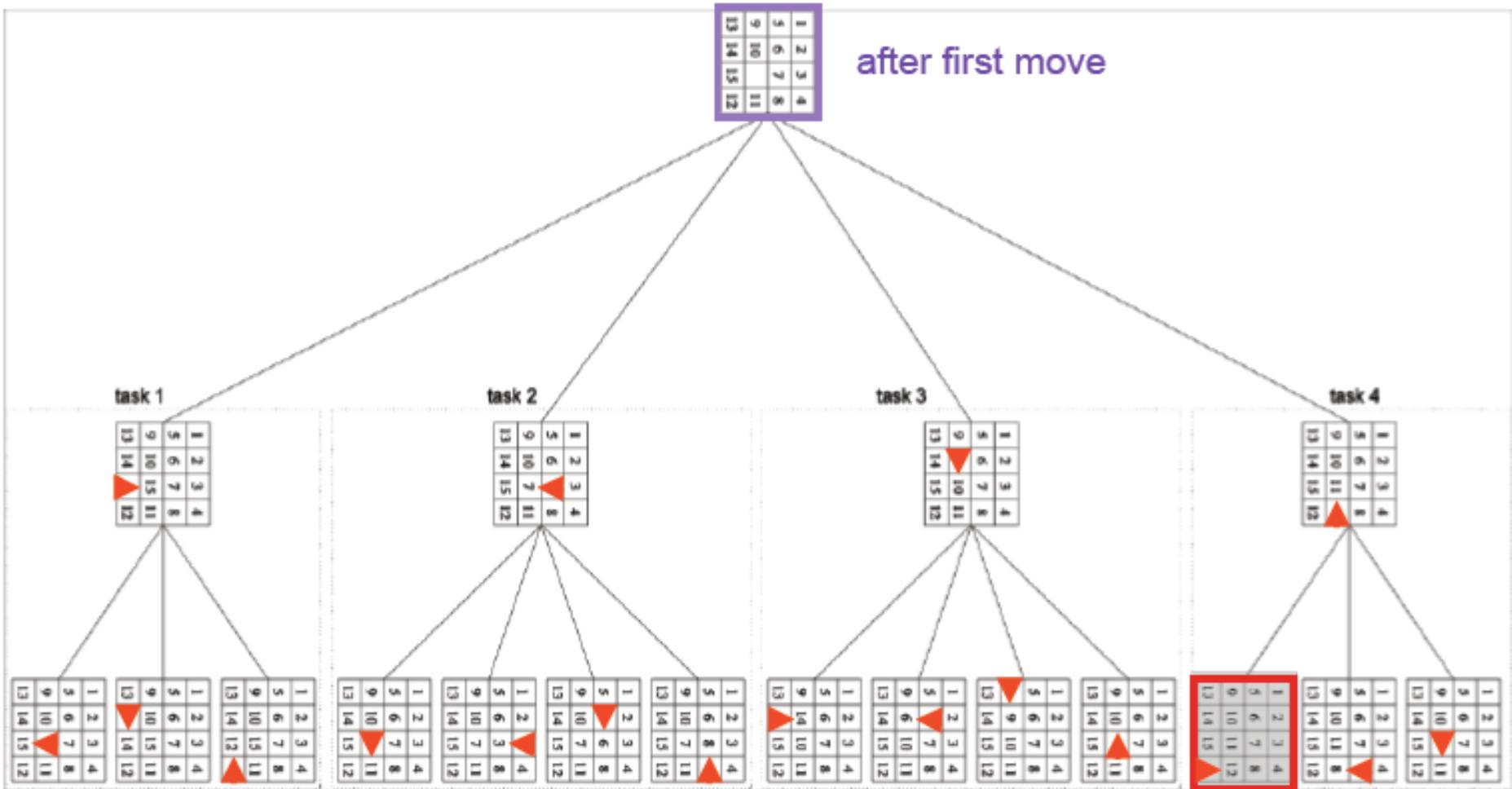
D



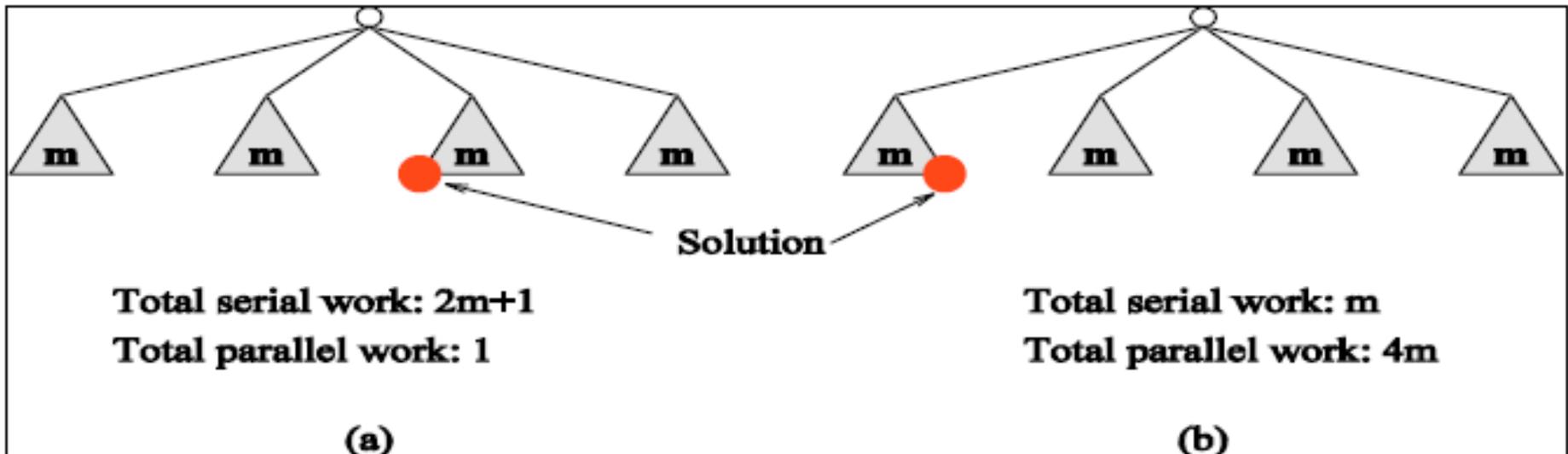
Huarong Road Game

- Solution algorithm

- Subsequent configurations are generated based on current configuration.
- Each configuration is then explored as an independent task.



- Difference between data-decomposition and exploratory decomposition
 - Tasks induced by data-decomposition are performed entirely and each task performs useful computation towards the solution of problem.
 - Tasks induced by exploratory can be terminated before finishing as soon as desired solution is found.
- Work induced by exploratory decomposition and performed by parallel formulation can be either smaller or greater than that performed by serial algorithm

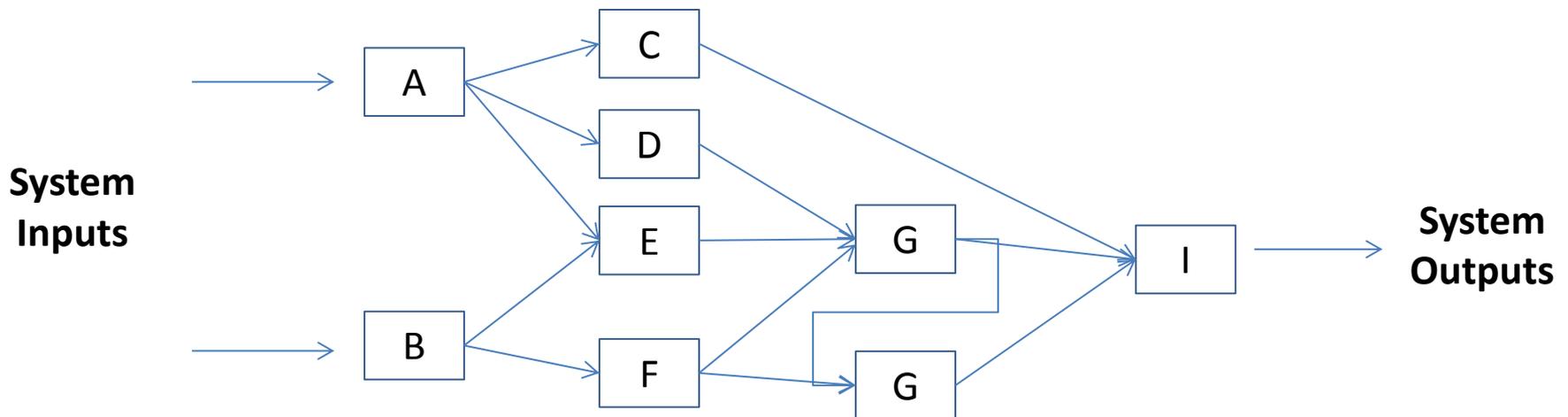


Speculative Decomposition

- This decomposition is used when a program may take one of many possible computationally significant branches depending on the output of other computations that precede it.
- While one task is performing the computation whose output is used in deciding the next configuration, other tasks can concurrently start the computations of the next stage.
 - The scenario is similar to evaluating one or more of the branches of a *switch* statement in C in parallel before the input for the *switch* is available.

Example: Speculative Decomposition

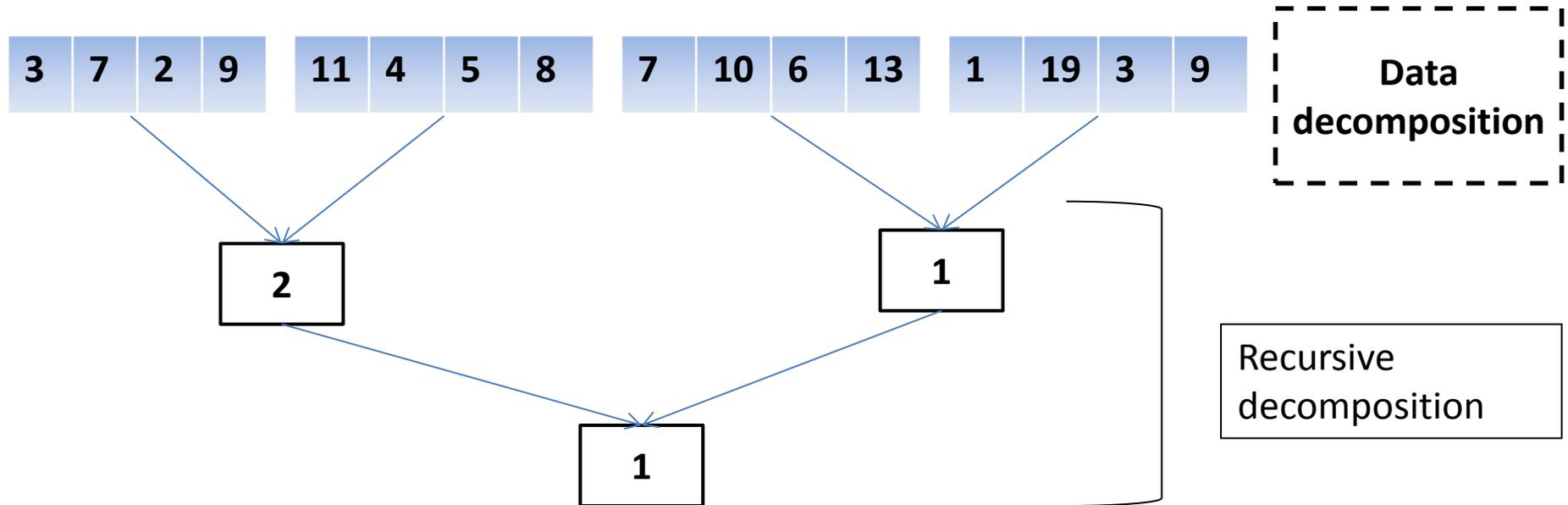
- Parallel *discrete event simulation*
 - The nodes of a directed network have input buffer of jobs. After processing the job, the node put results in the input buffer of nodes which are connected to it by outgoing edges. A node has to wait if the input buffer of one of its outgoing neighbors is full. There is a finite number of input job types.



- Inherently sequential problem
- Can be improved by starting simulating a subpart of the network, each assume one of several possible inputs to that stage (*overlapping different computations*).

Hybrid Decomposition

- Use several decomposition methods together
- Example: finding the minimum of any array of size 16 using 4 tasks.



Characteristics of Tasks

Key characteristics of tasks influencing choice of mapping and performance of parallel algorithm:

1. Task generation

- Static or dynamic generation
 - *Static*: all tasks are known before the algorithm starts execution. Data or recursive decomposition often leads to static task generation.
Ex. Matrix-multiplication. Recursive decomposition in finding min. of a set of numbers.
 - *Dynamic*: the actual tasks and the task-dependency graph are not explicitly available *a priori*. Recursive, exploratory decomposition can generate tasks dynamically.
Ex. Recursive decomposition in Quicksort, in which tasks are generated dynamically.

2. Task sizes

- Amount of time required to compute it: *uniform, non-uniform*

3. Knowledge of task sizes

- Ex. Size of task in 15-puzzle problem is unknown.

4. Size of data associated with tasks

- Data associated with the task must be available to the process performing the task. The size and location of data may determine the data-movement overheads.

Characteristics of Task Interactions

1) Static versus dynamic

- Static: interactions are known prior to execution.

2) Regular versus irregular

- Regular: interaction pattern can be exploited for efficient implementation.

3) Read-only versus read-write

4) One-way versus two-way

Static vs. Dynamic Interactions

- Static interaction
 - Tasks and associated interactions are predetermined: task-interaction graph and times that interactions occur are known: matrix multiplication
 - Easy to program
- Dynamic interaction
 - Timing of interaction or sets of tasks to interact with can not be determined prior to the execution.
 - Ex. Puzzle game. The tasks has exhausted its work can pick up an unexplored state from the queue of another busy task and start exploring it.
 - Difficult to program using message-passing; Shared-memory space programming may be simple

Regular vs. Irregular Interactions

- Regular interactions

- Interaction has a spatial structure that can be exploited for efficient implementation: ring, mesh

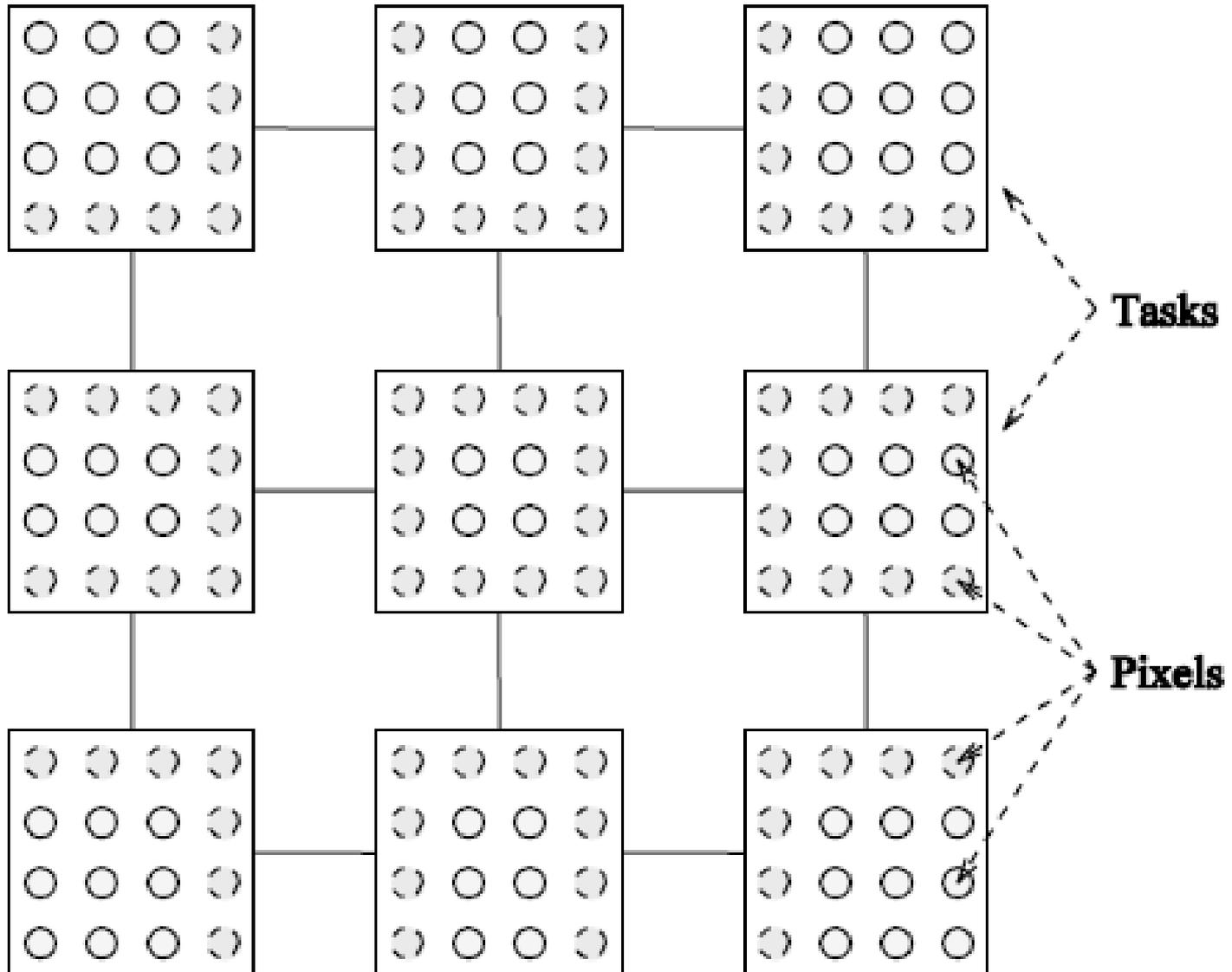
- Example: Explicit finite difference for solving PDEs. Image dithering.

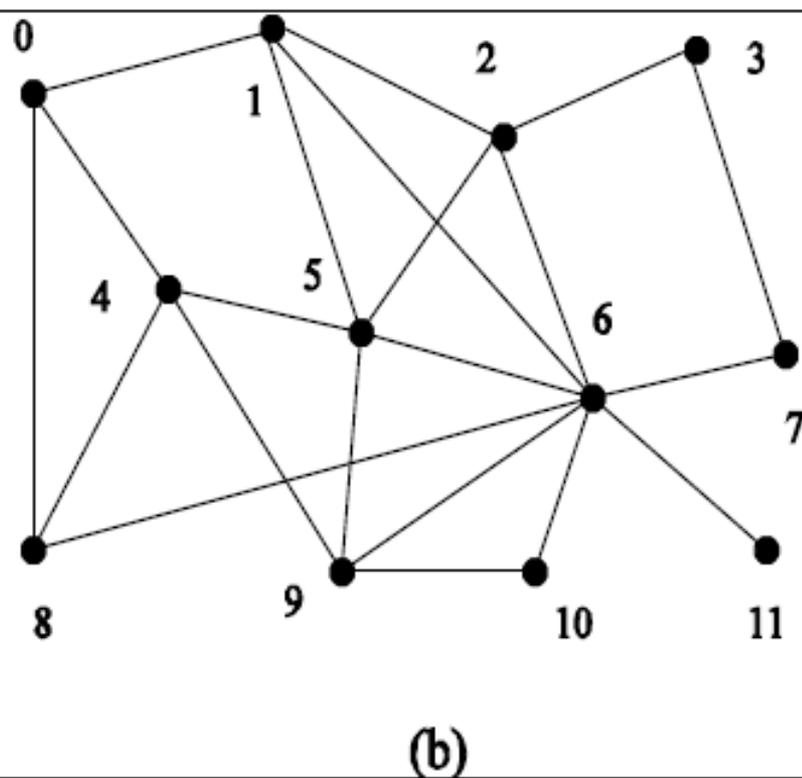
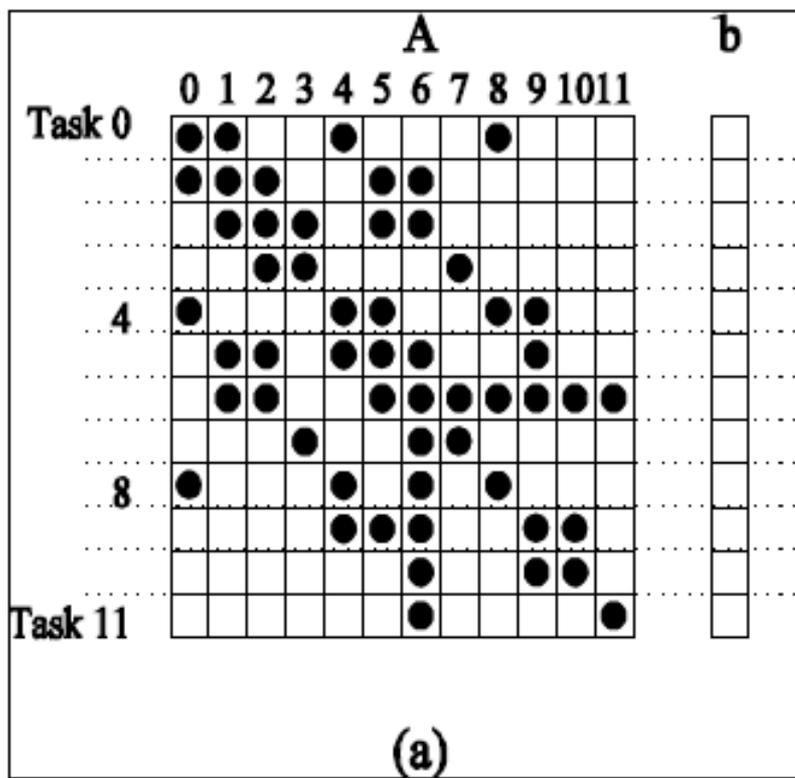
- Irregular Interactions

- Interactions has no well-defined structure

- Example: Sparse matrix-vector multiplication

Static regular interaction for image dithering





Read-Only vs. Read-Write Interactions

- Read-only interactions
 - Tasks only require read-only interactions
 - Example: matrix-matrix multiplication

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

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: $C_{2,2} = A_{2,1}B_{1,2} + A_{2,2}B_{2,2}$

- Read-write interactions
 - Multiple tasks need to read and write on some shared data

One-Way vs. Two-Way Interactions

- One-way interactions
 - One of a pair of communicating tasks initiates the interaction and completes it without interrupting the other one.
 - Example: read-only can be formulated as one-way
- Two-way interactions
 - Both tasks involve in interaction
 - Example: read-write can be formulated as two-way