

# Principle Of Parallel Algorithm Design (cont.)

---

Alexandre David  
B2-206



## Chapter 3 cont.

---

- Characteristics of Tasks and Interactions (3.3).
- Mapping Techniques for Load Balancing (3.4).
- Methods for Containing Interaction Overhead (3.5).
- Parallel Algorithm Models (3.6).



## So Far...

---

- Decomposition techniques.
  - Identify tasks.
  - Analyze with task dependency & interaction graphs.
  - Map tasks to processes.
- Now properties of tasks that affect a good mapping.
  - Task generation, size, and size of data.



## Task Generation

- Static task generation.
  - Tasks are known beforehand.
  - Apply to well-structured problems.
- Dynamic task generation. Model-checker
  - Tasks generated on-the-fly.
  - Tasks & task dependency graph not available beforehand.


09-03-2007

Alexandre David, MVP'07

4

The well-structured problem can typically be decomposed using data or recursive decomposition techniques.

Dynamic tasks generation: Exploratory or speculative decomposition techniques are generally used, but not always. Example: quicksort.



## Task Sizes

---

- Relative amount of time for completion.
  - Uniform – same size for all tasks.
    - Matrix multiplication.
  - Non-uniform. Model-checker
    - Optimization & search problems.

Typically the size of non-uniform tasks is difficult to evaluate beforehand.

## Size of Data Associated with Tasks

- Important because of locality reasons.
- Different types of data with different sizes
  - Input/output/intermediate data.
- Size of context – cheap or expensive communication with other tasks.

Example of 15-puzzle has a small context: easy to communicate the tasks to different processes.



## Characteristics of Task Interactions

- Static interactions.
  - Tasks and interactions known beforehand.
  - And interaction at pre-determined times.
- Dynamic interactions.
  - Timing of interaction unknown.
  - Or set of tasks not known in advance.

09-03-2007

Alexandre David, MVP'07

7

Static vs. dynamic.

Static or dynamic interaction pattern.

Dynamic harder to code, more difficult for MPI.



## Characteristics of Task

### Interactions

- Regular interactions.
  - The interaction graph follows a pattern.
- Irregular interactions.
  - No pattern.

Model-checker

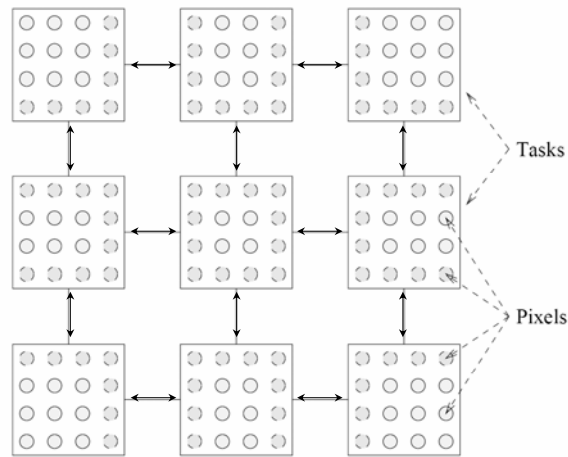
Regular vs. irregular.

Regular patterns can be exploited for efficient implementations.

Dynamic harder to code, more difficult for MPI.



## Example: Image Dithering



**Figure 3.22** The regular two-dimensional task-interaction graph for image dithering. The pixels with dotted outline require color values from the boundary pixels of the neighboring tasks.

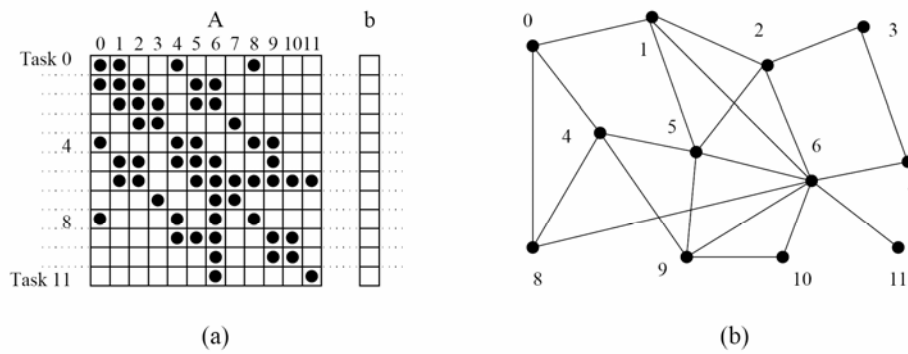
09-03-2007

Alexandre David, MVP'07

9

The color of each pixel is determined as the weighted average of its original value and the values of the neighboring pixels. Decompose into regions, 1 task/region. Pattern is a 2-D mesh. Regular pattern.

# Example: Sparse Matrix\*Vector



**Figure 3.6** A decomposition for sparse matrix-vector multiplication and the corresponding task-interaction graph. In the decomposition Task  $i$  computes  $\sum_{0 \leq j \leq 11, A[i,j] \neq 0} A[i,j].b[j]$ .

09-03-2007

Alexandre David, MVP'07

10

Irregular pattern. Interaction pattern depends on the values in A.

## Characteristics of Task Interactions

- Data sharing interactions:
  - Read-only interactions.
    - Read only data associated with *other* tasks.
  - Read-write interactions.
    - Read & modify data of *other* tasks.

09-03-2007

Alexandre David, MVP'07

11

Read-only vs. read-write.

Read-only example: matrix multiplication (share input). Read-write example: 15-puzzle with shared priority list of states to be explored; Priority given by some heuristic to evaluate the distance to the goal.

## Characteristics of Task Interactions

- One-way interactions.
  - Only one task initiates and completes the communication *without* interrupting the other one.
- Two-way interactions.
  - Producer – consumer model.

09-03-2007

Alexandre David, MVP'07

12

One-way vs. two-way.

One-way more difficult with MPI since MPI has an explicit send & receive set of calls. Conversion one-way to two-way with polling or another thread waiting for communication.



## Mapping Techniques for Load Balancing

---

- Map tasks onto processes.
- Goal: minimize overheads.
  - Communication.
  - Idling.
- Uneven load distribution may cause idling.
  - Constraints from task dependency → wait for other tasks.

09-03-2007

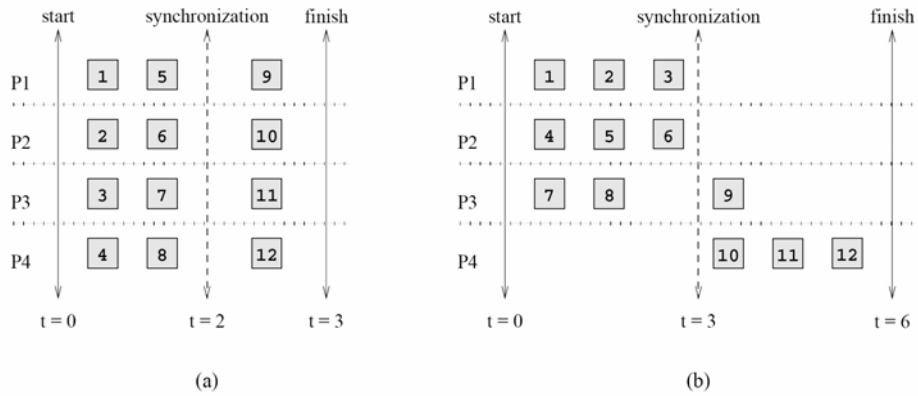
Alexandre David, MVP'07

13

Minimizing communication may contradict minimizing idling. Put tasks that communicate with each other on the same process but may unbalance the load -> distribute them but increase communication.

Load balancing is not enough to minimize idling.

# Example



**Figure 3.23** Two mappings of a hypothetical decomposition with a synchronization.

09-03-2007

Alexandre David, MVP'07

14

Global balancing OK but due to task dependency P4 is idling.



## Mapping Techniques

- Static mapping.
  - NP-complete problem for non-uniform tasks.
  - Large data compared to computation.
- Dynamic mapping.
  - Dynamically generated tasks.
  - Task size unknown.

09-03-2007

Alexandre David, MVP'07

15

Even static mapping may be difficult: The problem of obtaining an optimal mapping is an NP-complete problem for non-uniform tasks. In practice simple heuristics provide good mappings.

Cost of moving data may out-weight the advantages of dynamic mapping.

In shared address space dynamic mapping may work well even with large data, but be careful with the underlying architecture (NUMA/UMA) because data may be moved physically.



## Schemes for Static Mapping

---

- Mappings based on data partitioning.
- Mappings based on task graph partitioning.
- Hybrid mappings.



## Array Distribution Scheme

- Combine with “owner computes” rule to partition into sub-tasks.

row-wise distribution

$P_0$
$P_1$
$P_2$
$P_3$
$P_4$
$P_5$
$P_6$
$P_7$

column-wise distribution

$P_0$	$P_1$	$P_2$	$P_3$	$P_4$	$P_5$	$P_6$	$P_7$
-------	-------	-------	-------	-------	-------	-------	-------

1-D block distribution scheme.

09-03-2007

Alexandre David, MVP'07

17

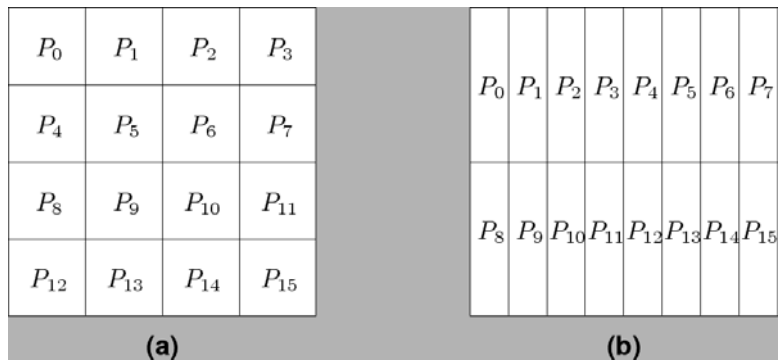
Data partitioning mapping.

Mapping data = mapping tasks.

Simple block-distribution.



## Block Distribution cont.



Generalize to higher dimensions:  $4 \times 4$ ,  $2 \times 8$ .



## Example: Matrix\*Matrix

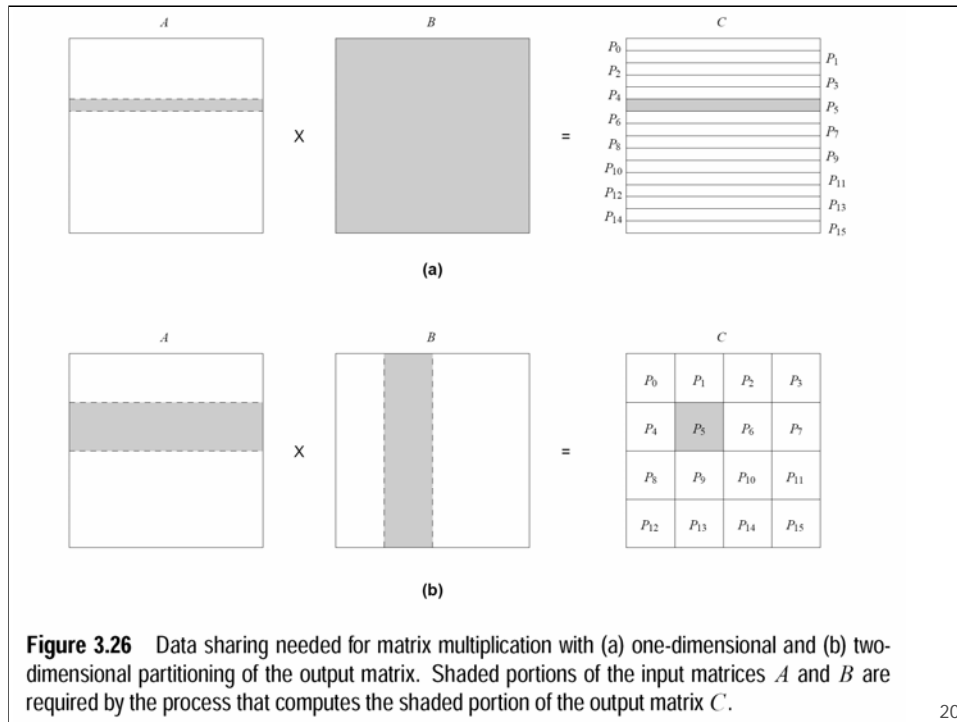
- Partition output of  $C=A*B$ .
- Each entry needs the same amount of computation.
- Blocks on 1 or 2 dimensions.
- Different data sharing patterns.
- *Higher dimensional* distributions
  - means we can use *more processes*.
  - sometimes *reduces* interaction.

09-03-2007

Alexandre David, MVP'07

19

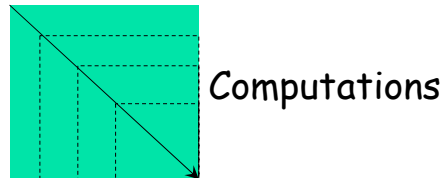
In the case of matrix  $n*n$  multiplication, 1-D  $\rightarrow$   $n$  processes at most, 2-D  $n^2$  processes at most.



$O(n^2/\sqrt{p})$  vs.  $O(n^2)$  shared data.

## Imbalance Problem

- If the amount of *computation* associated with data *varies* a lot then *block decomposition* leads to *imbalances*.
- Example: LU factorization (or Gaussian elimination).



Exercise on LU-decomposition.

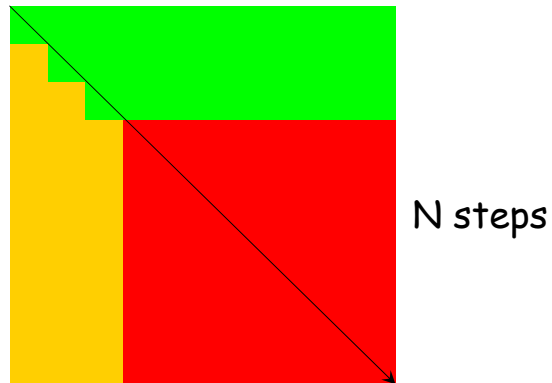
## LU Factorization

- Non singular square matrix A (invertible).
- $A = L*U$ .
- Useful for solving linear equations.

The diagram shows the equation  $A = L \times U$ . Matrix A is represented by a solid magenta square. Matrix L is represented by a green square with a white diagonal line from the top-left to the bottom-right. Matrix U is represented by a yellow square with a white diagonal line from the top-left to the bottom-right. The matrices are arranged from left to right, separated by an equals sign and a multiplication symbol.

# LU Factorization

In practice we work on  $A$ .



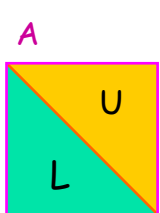
# LU Algorithm

```

Proc LU(A)
begin
  for k := 1 to n-1 do
    for j := k+1 to n do
      A[j,k] := A[j,k]/A[k,k]
    endfor
    for j := k+1 to n do
      for i := k+1 to n do
        A[i,j] := A[i,j] - A[i,k]*A[k,j]
      endfor
    endfor
  endfor
end

```

$U[k,k]$   
 Normalize L  
 $U[k,j] := A[k,j]/L[k,k]$   
 $L[j,k]$   
 $L[i,k]$   $U[k,j]$



09-03-2024





## Another Variant

---

```
for k := 1 to n-1 do
  for j := k+1 to n do
    A[k,j] := A[k,j]/A[k,k]
    for i := k+1 to n do
      A[i,j] := A[i,j] - A[i,k]*A[k,j]
    endfor
  endfor
endfor
```

More common in the litterature.

# Decomposition

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

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

**Figure 3.27** A decomposition of LU factorization into 14 tasks.

Load imbalance for individual tasks. Load imbalance from dependencies.

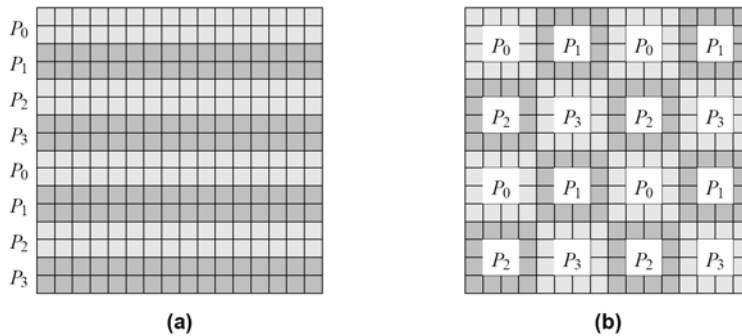


## Cyclic and Block-Cyclic Distributions

---

- Idea:
  - Partition an array into many *more blocks than available processes*.
  - Assign partitions (tasks) to processes in a round-robin manner.
- → each process gets several non adjacent blocks.

## Block-Cyclic Distributions



- a) Partition  $16 \times 16$  into  $2 \times 4$  groups of 2 rows.  
 $\alpha p$  groups of  $n / \alpha p$  rows.
- b) Partition  $16 \times 16$  into square blocks of size  $4 \times 4$  distributed on  $2 \times 2$  processes.  
 $\alpha^2 p$  groups of  $n / \alpha^2 p$  squares.

09-03-2007

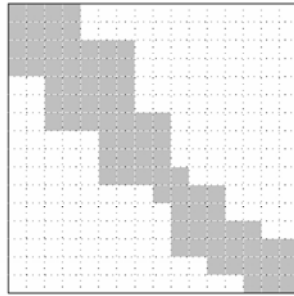
Alexandre David, MVP'07

28

Reduce the amount of idling because all processes have a sampling of tasks from *all parts* of the matrix.

**But** lack of locality may result in performance penalties + leads to high degree of interaction. Good value for  $\alpha$  to find a compromise.

# Randomized Distributions



(a)

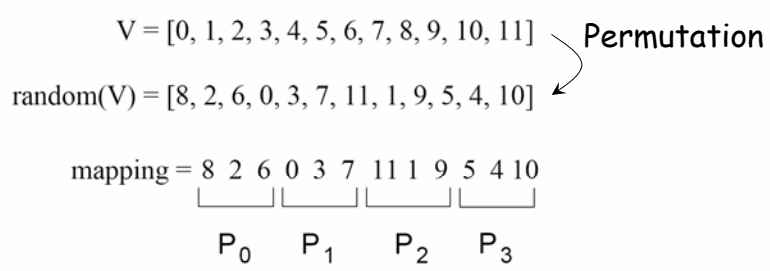
$P_0$	$P_1$	$P_2$	$P_3$	$P_0$	$P_1$	$P_2$	$P_3$
$P_4$	$P_5$	$P_6$	$P_7$	$P_4$	$P_5$	$P_6$	$P_7$
$P_8$	$P_9$	$P_{10}$	$P_{11}$	$P_8$	$P_9$	$P_{10}$	$P_{11}$
$P_{12}$	$P_{13}$	$P_{14}$	$P_{15}$	$P_{12}$	$P_{13}$	$P_{14}$	$P_{15}$
$P_0$	$P_1$	$P_2$	$P_3$	$P_0$	$P_1$	$P_2$	$P_3$
$P_4$	$P_5$	$P_6$	$P_7$	$P_4$	$P_5$	$P_6$	$P_7$
$P_8$	$P_9$	$P_{10}$	$P_{11}$	$P_8$	$P_9$	$P_{10}$	$P_{11}$
$P_{12}$	$P_{13}$	$P_{14}$	$P_{15}$	$P_{12}$	$P_{13}$	$P_{14}$	$P_{15}$

(b)

**Irregular distribution with regular mapping!  
Not good.**

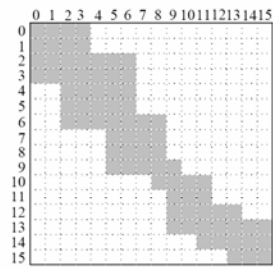


# 1-D Randomized Distribution

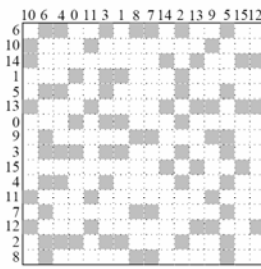


**Figure 3.32** A one-dimensional randomized block mapping of 12 blocks onto four process (i.e.,  $\alpha = 3$ ).

# 2-D Randomized Distribution



(a)



(b)

$P_0$	$P_1$	$P_2$	$P_3$
$P_4$	$P_5$	$P_6$	$P_7$
$P_8$	$P_9$	$P_{10}$	$P_{11}$
$P_{12}$	$P_{13}$	$P_{14}$	$P_{15}$

(c)

2-D block random distribution.

Block mapping.

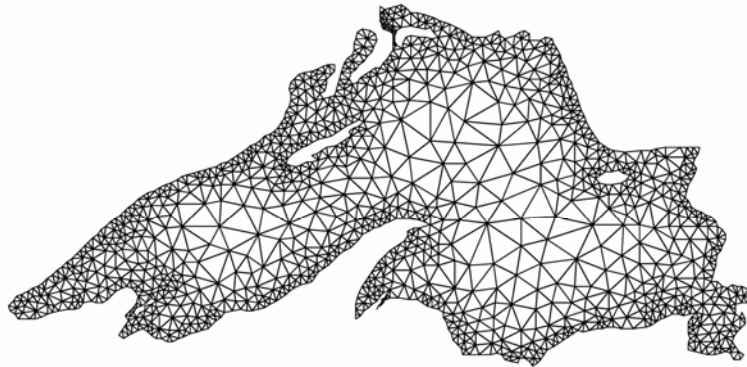


## Graph Partitioning

- For sparse data structures and data dependent interaction patterns.
  - Numerical simulations. Discretize the problem and represent it as a mesh.
- Sparse matrix: assign equal number of nodes to processes & minimize interaction.
- Example: simulation of dispersion of a water contaminant in Lake Superior.

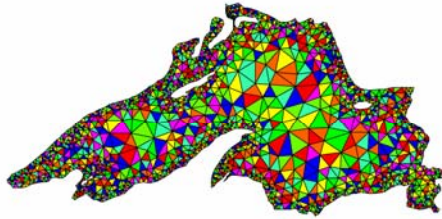


## Discretization

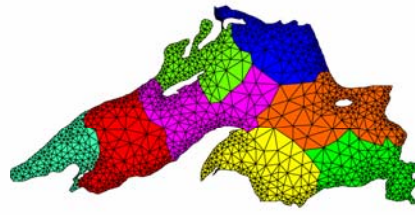


**Figure 3.34** A mesh used to model Lake Superior.

## Partitioning Lake Superior



Random partitioning.



Partitioning with minimum edge cut.

Finding an exact optimal partitioning is an NP-complete problem.

09-03-2007

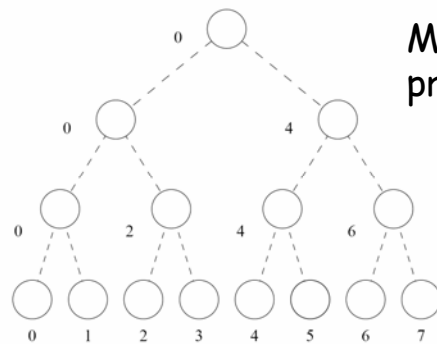
Alexandre David, MVP'07

34

Minimum edge cut from a graph point of view. Keep locality of data with processes to minimize interaction.

## Mappings Based on Task Partitioning

- Partition the task dependency graph.
  - Good when static task dependency graph with known task sizes.



Mapping on 8 processes.

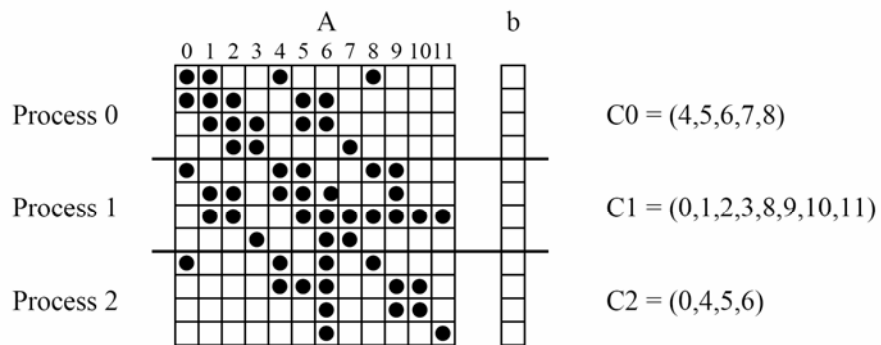
35

Determining an optimal mapping is NP-complete. Good heuristics for structured graphs.

Binary tree task dependency graph: occurs in recursive decompositions as seen before. The mapping minimizes interaction. There is idling but it is inherent to the task dependency graph, we do not add more.

This example good on a hypercube. See why?

# Sparse Matrix \* Vector



**Figure 3.38** A mapping for sparse matrix-vector multiplication onto three processes. The list  $C_i$  contains the indices of  $b$  that Process  $i$  needs to access from other processes.

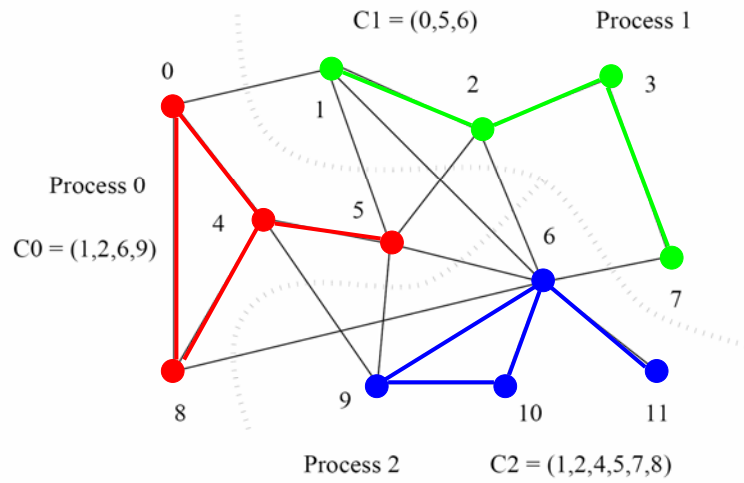
09-03-2007

Alexandre David, MVP'07

36

Example seen before.

# Sparse Matrix\*Vector



**Figure 3.39** Reducing interaction overhead in sparse matrix-vector multiplication by partitioning the task-interaction graph.

09-03-2007

Alexandre David, MVP'07

37

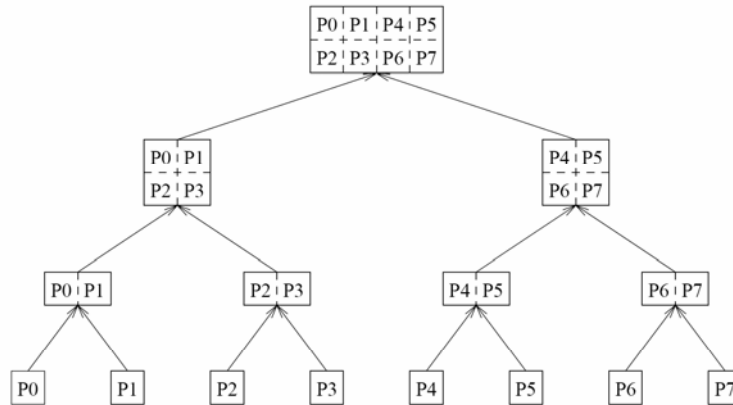


## Hierarchical Mappings

---

- Combine several mapping techniques in a structured (hierarchical) way.
- Task mapping of a binary tree (quicksort) does not use all processors.
  - Mapping based on task dependency graph (hierarchy) & block.

## Binary Tree -> Hierarchical Block Mapping



**Figure 3.40** An example of hierarchical mapping of a task-dependency graph. Each node represented by an array is a supertask. The partitioning of the arrays represents subtasks, which are mapped onto eight processes.



## Schemes for Dynamic Mapping

- Centralized Schemes.
  - Master manages pool of tasks.
  - Slaves obtain work.
  - Limited scalability.
- Distributed Schemes.
  - Processes *exchange tasks* to balance work.
  - Not simple, many issues.

09-03-2007

Alexandre David, MVP'07

40

Centralized schemes are easy to implement but present an obvious bottleneck (the master).

**Self-scheduling:** slaves pick up work to do whenever they are idle.

Bottleneck: tasks of size  $M$ , it takes  $t$  to assign work to a slave  $\rightarrow$  at most  $M/t$  processes can be kept busy.

**Chunk-scheduling:** a way to reduce bottlenecks by getting a group of tasks. Problem for load imbalances.

Distributed schemes more difficult to implement.

How do you choose sender & receiver? i.e. if A is overloaded, which process gets something?

Initiate transfer by sender or receiver? i.e. A overloaded sends work or B idle requests work?

How much work to transfer?

When to transfer?

Answers are application specific.





## Minimizing Interaction Overheads

- Maximize data locality.
  - Minimize volume of data-exchange.
  - Minimize frequency of interactions.
- Minimize contention and hot spots.
  - Share a link, same memory block, etc...
  - Re-design original algorithm to change the interaction pattern.

09-03-2007

Alexandre David, MVP'07

41

Minimize volume of exchange → maximize temporal locality. Use higher dimensional distributions, like in the matrix multiplication example. We can store intermediate results and update global results less often.

Minimize frequency of interactions → maximize spatial locality.

Related to the previously seen cost model for communications.

Changing the interaction pattern: For the matrix multiplication example, the sum is commutative so we can re-order the operations modulo  $\sqrt{p}$  to remove contention.



## Minimizing Interaction Overheads

---

- Overlapping computations with interactions
  - to reduce idling.
    - Initiate interactions in advance.
    - Non-blocking communications.
    - Multi-threading.
- Replicating data or computation.
- Group communication instead of point to point.
- Overlapping interactions.

09-03-2007

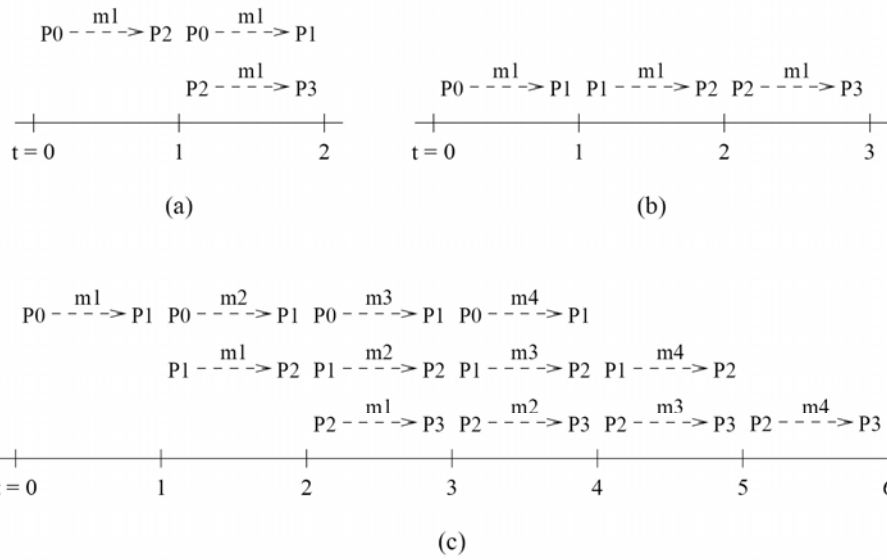
Alexandre David, MVP'07

42

Replication is useful when the cost of interaction is greater than replicating the computation. Replicating data is like caching, good for read-only accesses. Processing power is cheap, memory access is expensive – also apply at larger scale with communicating processes.

Collective communication such as broadcast. However, depending on the communication pattern, a custom collective communication may be better.

# Overlapping Interactions



**Figure 3.41** Illustration of overlapping interactions in broadcasting data from one to four processes.

## Parallel Algorithm Models

*How to select decomposition & mapping techniques to minimize interaction.*

- Data parallel model.
  - Tasks statically mapped.
  - Similar operations on different data.
    - SIMD.
- Task graph model.
  - Start from task dependency graph.
  - Use task interaction graph to promote locality.

An algorithm model is a *way of structuring a parallel algorithm* by selecting a *decomposition and mapping* technique and applying the appropriate strategy to minimize interactions.



## Parallel Algorithm Models

- Work pool (or task pool) model.
  - No pre-mapping – centralized or not.
- Master-slave model.
  - Master generates work for slaves – allocation static or dynamic.
- Pipeline or producer – consumer model.
  - Stream of data traverses processes – stream parallelism.

09-03-2007

Alexandre David, MVP'07

45

Pipeline model heavily used in GPUs. Load balancing is a function of task granularity.

+ hybrid models:

- Multiple models applied hierarchically.
- Multiple models applied sequentially to different phases.