VIT®
BHOPAL
www.vitbhopal.ac.in

CSE3009 - Parallel and Distributed Computing

Course Type: LTP Credits: 4

Prepared by
Dr Komarasamy G
Associate Professor (Grade-2)
School of Computing Science and Engineering
VIT Bhopal University

Unit-2

Parallel Algorithm and Design - Preliminaries – Decomposition Techniques – Mapping Techniques for Load balancing.

Synchronous Parallel Processing – Introduction, Example - SIMD Architecture and Programming Principles.

Mapping Techniques for Load balancing

- Once a computation has been **decomposed into tasks**, these tasks are mapped onto processes with the objective that all tasks complete in the shortest amount of elapsed time.
- In order to achieve a small execution time, the ***overheads*** of executing the tasks in parallel must be minimized.
- For a given decomposition, there are two key sources of overhead. **The time spent in inter-process interaction** is one source of overhead.
- Another important source of overhead is the **time that some processes may spend being idle.** Some processes can be idle even before the overall computation is finished for a variety of reasons.
- Uneven load distribution may cause some processes to finish earlier than others.

Mapping Techniques for Load balancing

- At times, all the unfinished tasks mapped onto a process may be waiting for tasks mapped onto other processes to finish in order to satisfy the constraints imposed by the task-dependency graph. Both interaction and idling are often a function of mapping.

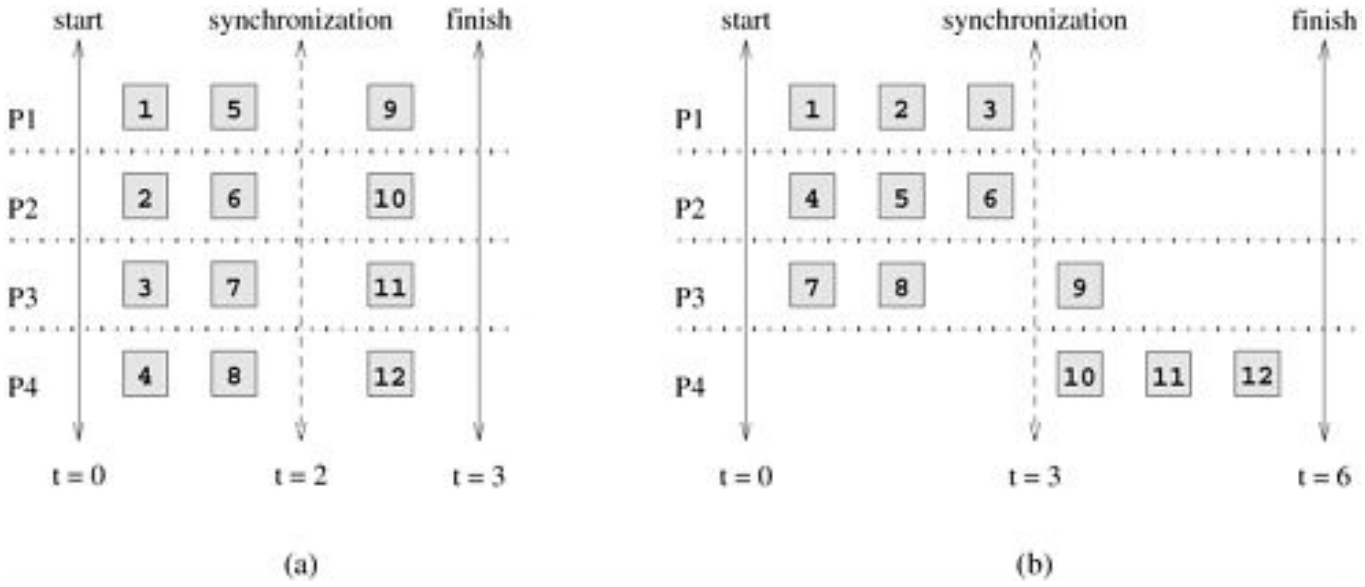
- **Therefore, a good mapping of tasks onto processes must strive to achieve the two objectives.**
- **(1) reducing the amount of time processes spend in interacting with each other.**
- **(2) reducing the total amount of time some processes are idle while the others are engaged in performing some tasks.**
- These two objectives often conflict with each other.

Mapping Techniques for Load balancing

- For example, the objective of minimizing the interactions can be easily achieved by **assigning sets of tasks** that need to interact with each other onto the same process.
- In most cases, such a **mapping will result in a highly unbalanced workload among the processes.**
- In fact, following this strategy to the limit will often map all tasks onto a single process.
- As a result, the processes with a lighter load will be idle when those with a heavier load are trying to finish their tasks.
- Similarly, to balance the load among processes, it may be necessary to assign tasks that interact heavily to different processes.
- Due to the conflicts between these objectives, finding a good mapping is a nontrivial problem.

Mapping Techniques for Load balancing

- Figure, **shows two mappings of 12-task decomposition in which the last four tasks** can be started only after the first eight are finished due to dependencies among tasks. As the figure shows, two mappings, each with an overall balanced workload, can result in different completion times.



Two mappings of a hypothetical decomposition with a synchronization.

Mapping Techniques for Load balancing

- Mapping techniques used in parallel algorithms can be broadly classified into two categories: ***static*** and ***dynamic***.
- The parallel programming paradigm and the characteristics of tasks and the interactions among them determine whether a static or a dynamic mapping is more suitable.
- **Static Mapping:** Static mapping techniques distribute the tasks among processes **prior to the execution of the algorithm**.
- For statically generated tasks, either static or dynamic mapping can be used.
- The choice of a good mapping in this case depends on several factors, including the knowledge of **task sizes, the size of data associated with tasks, the characteristics of inter-task interactions, and even the parallel programming paradigm**.

Mapping Techniques for Load balancing

- **Dynamic Mapping:** Dynamic mapping techniques distribute the work among processes **during the execution of the algorithm**.
- If tasks are generated dynamically, then they must be mapped dynamically too. If **task sizes are unknown**, then a **static mapping can potentially lead to serious load-imbalances and dynamic mappings are usually more effective.**
- If the amount of data associated with tasks is large relative to the computation, then a dynamic mapping may entail moving this data among processes.
- The cost of this data movement may outweigh some other advantages of dynamic mapping and may render a static mapping more suitable. However, in a shared-address-space paradigm, dynamic mapping may work well even with large data associated with tasks if the interaction is read-only.

Schemes for Static Mapping

- Static mapping is often, though not exclusively, used in conjunction with a decomposition **based on data partitioning.**
- Static mapping is also used for mapping certain problems that are expressed naturally by a **static task-dependency graph**.
- In the following subsections, we will discuss mapping schemes based on data partitioning and task partitioning.
- **Mappings Based on Data Partitioning**
- Mappings based on partitioning two of the most common ways of representing data in algorithms, namely, **arrays and graphs**.
- The data-partitioning actually induces a decomposition, but the partitioning or the decomposition is selected with the final mapping in mind.

Mapping Techniques for Load balancing

- **Array Distribution Schemes**
- In a decomposition based on partitioning data, the tasks are closely associated with **portions of data** **by the owner-computes rule.**
- Therefore, mapping the relevant data onto the processes is equivalent to **mapping tasks onto processes.**
- We now study some commonly used techniques of **distributing arrays or matrices among processes.**

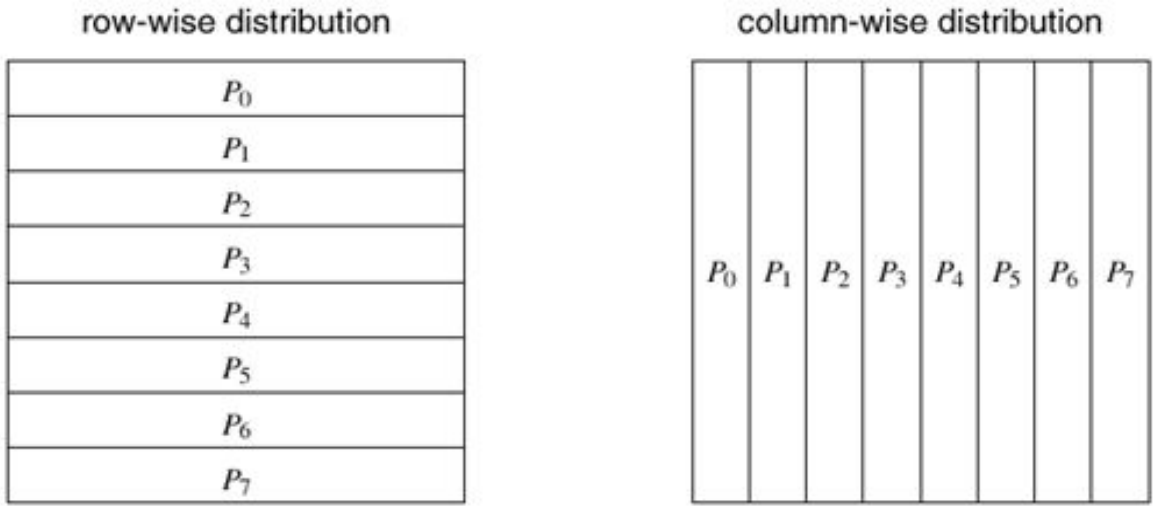
Mapping Techniques for Load balancing

Block Distributions

- ***Block distributions*** are some of the simplest ways to distribute an array and **assign uniform contiguous portions** of the array to different processes.
- In these distributions, a ***d*-dimensional array is distributed among the processes** such that each process receives a **contiguous block of array** entries along a specified subset of array dimensions.
- Block distributions of arrays are particularly suitable when there is a locality of interaction,
- i.e., **computation of an element of an array requires other nearby elements in the array.**

Mapping Techniques for Load balancing

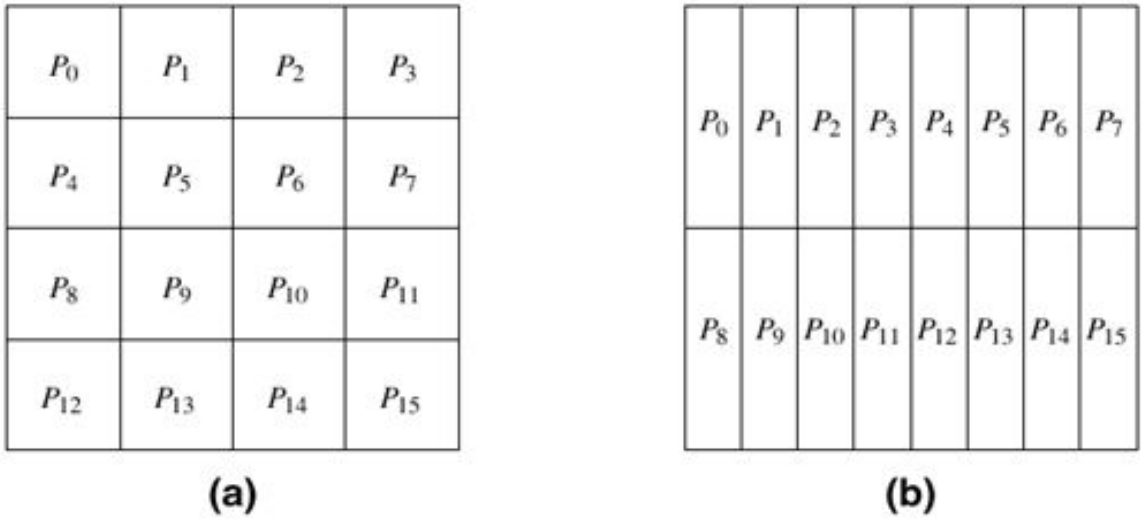
- For example, consider an ***n* x *n* two-dimensional array *A*** with *n* rows and *n* columns. We can now select one of these dimensions, e.g., the first dimension, and partition the array into *p* parts such that the *k*th part contains rows $kn/p \ldots (k+1)n/p - 1$, where $0 \leq k < p$.
- That is, each partition contains a block of n/p consecutive rows of A. Similarly, if we partition A along the second dimension, then each partition contains a block of n/p consecutive columns.
- **These row- and column-wise array distributions are illustrated in Figure**



Examples of one-dimensional partitioning of an array among eight processes.

Mapping Techniques for Load balancing

- instead of selecting a single dimension, we can select **multiple dimensions to partition**. For instance, in the case of array A we can select both dimensions and partition the matrix into blocks such that each block corresponds to a n/p1 x n/p2 section of the matrix, with p = p1 x p2 being the number of processes. **Figure illustrates two different two-dimensional distributions**, on a 4 x 4 and 2x 8 process grid, respectively. In general, given a d-dimensional array, we can distribute it using up to a **d-dimensional block distribution.**



Examples of two-dimensional distributions of an array,
(a) on a 4 x 4 process grid, and (b) on a 2 x 8 process grid.

Mapping Techniques for Load balancing

Cyclic and Block-Cyclic Distributions

- If the amount of work differs for different elements of a matrix, a block distribution can potentially **lead to load imbalances.**
- A classic example of this phenomenon is **LU (lower–upper) factorization of a matrix**, in which the amount of computation increases from the **top left to the bottom right of the matrix.**
- The LU factorization algorithm factors a nonsingular square matrix A into the product of a **lower triangular matrix L with a unit diagonal and an upper triangular matrix U.**

Mapping Techniques for Load balancing

```
1.    procedure COL_LU (A)
2.    begin
3.       for k := 1 to n do
4.           for j := k to n do
5.               A[j, k]:= A[j, k]/A[k, k];
6.           endfor;
7.           for j := k + 1 to n do
8.               for i := k + 1 to n do
9.                   A[i, j] := A[i, j] - A[i, k] x A[k, j];
10.              endfor;
11.          endfor;
     /*
After this iteration, column A[k + 1 : n, k] is logically the kth
column of L and row A[k, k : n] is logically the kth row of U.
     */
12.      endfor;
13.   end COL_LU
```

Algorithm, A serial column-based algorithm to factor a nonsingular matrix A into a lower triangular matrix L and an upper-triangular matrix U.

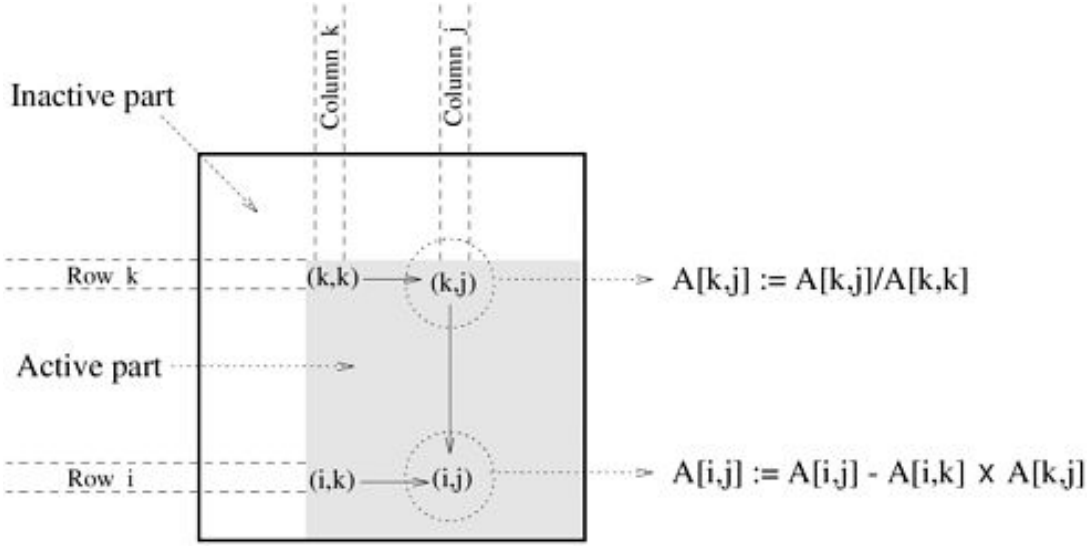
Matrices L and U share space with A. On Line 9, A[i, j] on the left side of the assignment is equivalent to L [i, j] if i > j; otherwise, it is equivalent to U [i, j].

Mapping Techniques for Load balancing

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

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: $A_{3,3} \rightarrow L_{3,3}U_{3,3}$

A decomposition of LU factorization into 14 tasks.



A typical computation in Gaussian elimination and the active part of the coefficient matrix during the *k*th iteration of the outer loop.

Mapping Techniques for Load balancing

P_0 T_1	P_3 T_4	P_6 T_5
P_1 T_2	P_4 T_6 T_{10}	P_7 T_8 T_{12}
P_2 T_3	P_5 T_7 T_{11}	P_8 T_9 T_{13} T_{14}

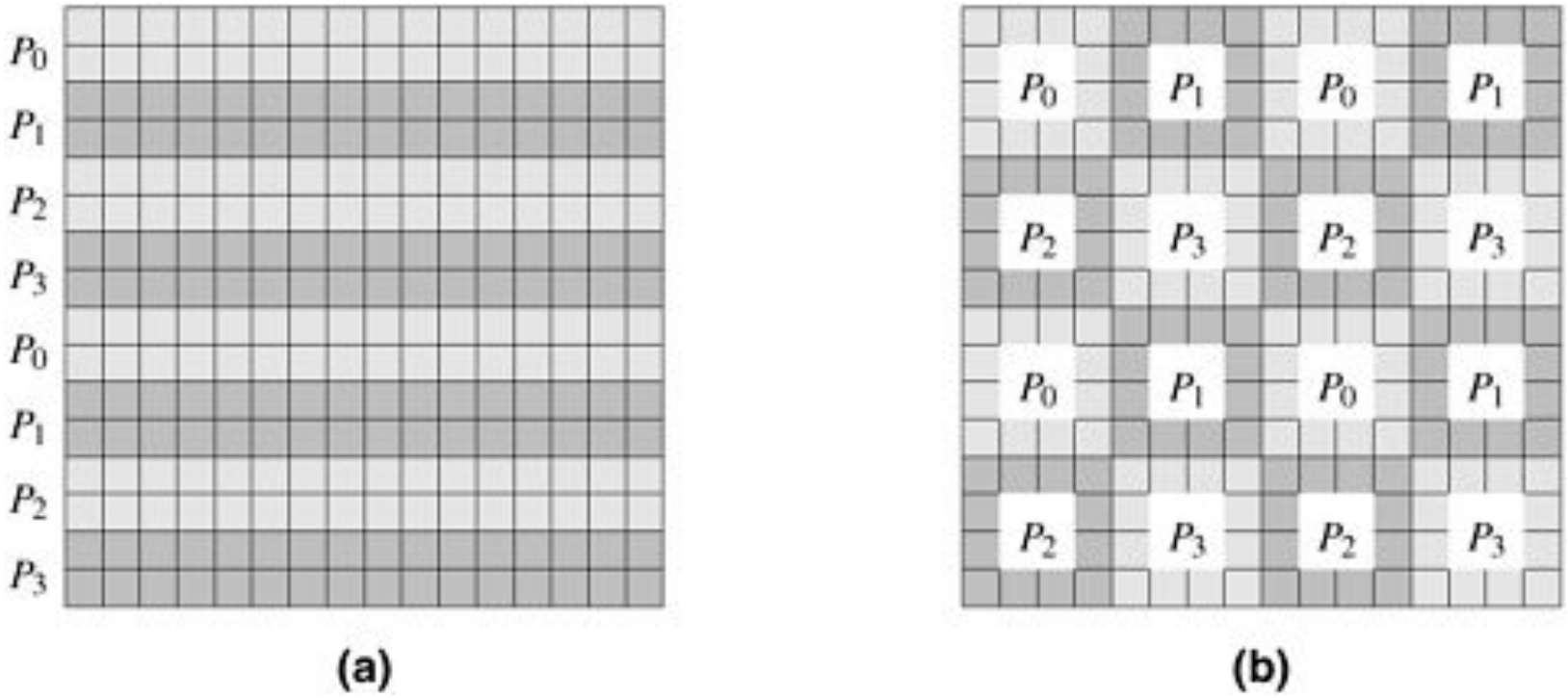
A naive mapping of LU factorization tasks onto processes based on a two-dimensional block distribution.

The ***block-cyclic distribution*** is a variation of the block distribution scheme that can be used to the **load-imbalance and idling problems.**

A detailed description of LU factorization with block-cyclic mapping , where it is shown how a block-cyclic mapping leads to a substantially more balanced work distribution than in above Figure

Mapping Techniques for Load balancing

- Examples of one- and two-dimensional block-cyclic distributions among four processes. (a) The rows of the array are grouped into blocks each consisting of two rows, resulting in eight blocks of rows. These blocks are distributed to four processes in a wraparound fashion. (b) The matrix is blocked into 16 blocks each of size 4 x 4, and it is mapped onto a 2 x 2 grid of processes in a wraparound fashion.



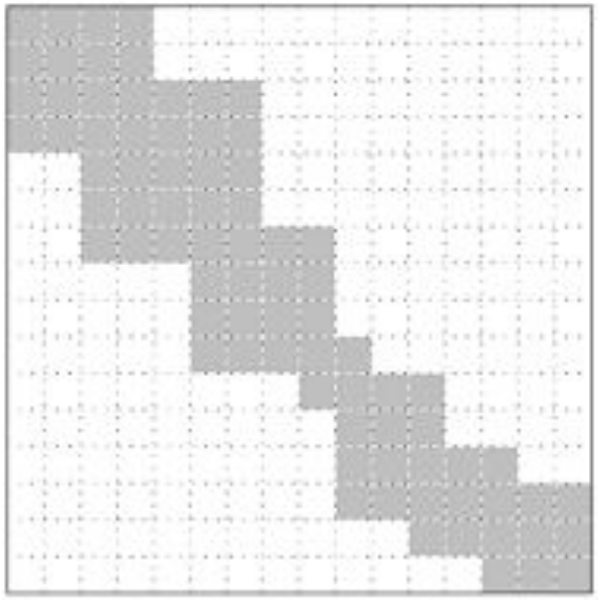
Mapping Techniques for Load balancing

- The reason why a block-cyclic distribution is able to significantly reduce the amount of idling is that **all processes have a sampling of tasks from all parts of the matrix.**
- As a result, even if different parts of the matrix require different amounts of work, the overall work on each process balances out.
- Also, since the tasks assigned to a process belong to different parts of the matrix, there is a good chance that at least some of them are **ready for execution at any given time.**

Mapping Techniques for Load balancing

- **Randomized Block Distributions**
- A block-cyclic distribution may not always be able to balance computations when the distribution of work has some special patterns.
- For example, consider the sparse matrix shown in Figure in which the **shaded areas correspond to regions containing nonzero elements.**
- If this matrix is distributed using a two-dimensional block-cyclic distribution, as illustrated in Figure then we will end up assigning more non-zero blocks to the diagonal processes P0, P5, P10, and P15 than on any other processes.
- In fact some processes, like **P12, will not get any work.**

Mapping Techniques for Load balancing



(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)

Using the block-cyclic distribution shown in (b) to distribute the computations performed in array (a) will lead to load imbalances.

Randomized block distribution, a more general form of the block distribution, can be used in situations illustrated in Figure. Just like a block-cyclic distribution, load balance is sought by partitioning the array into many more blocks than the number of available processes.

Mapping Techniques for Load balancing

- **A one-dimensional randomized block mapping of 12 blocks onto four process (i.e., a = 3).**

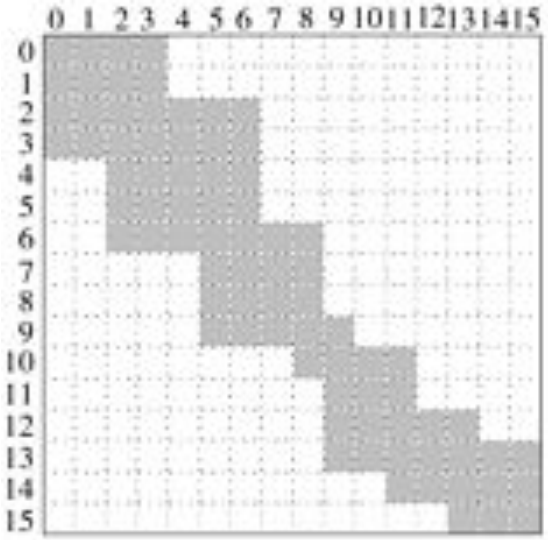
$$V = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]$$

$$\text{random}(V) = [8, 2, 6, 0, 3, 7, 11, 1, 9, 5, 4, 10]$$

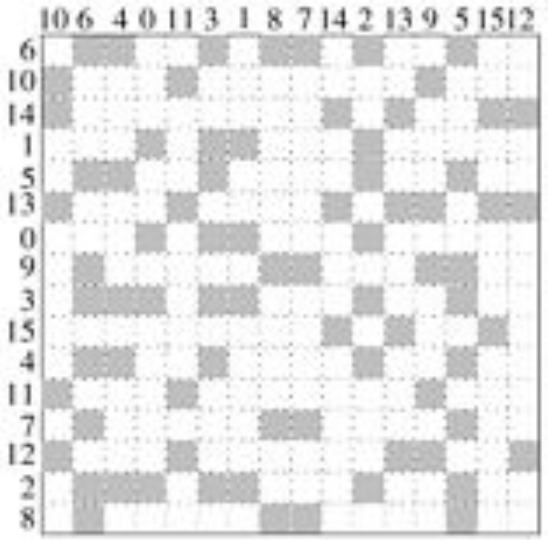
$$\text{mapping} = \underbrace{8\ 2\ 6}_{P_0}\ \underbrace{0\ 3\ 7}_{P_1}\ \underbrace{11\ 1\ 9}_{P_2}\ \underbrace{5\ 4\ 10}_{P_3}$$

Mapping Techniques for Load balancing

- **Using a two-dimensional random block distribution shown in (b) to distribute the computations performed in array (a), as shown in (c).**



(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}

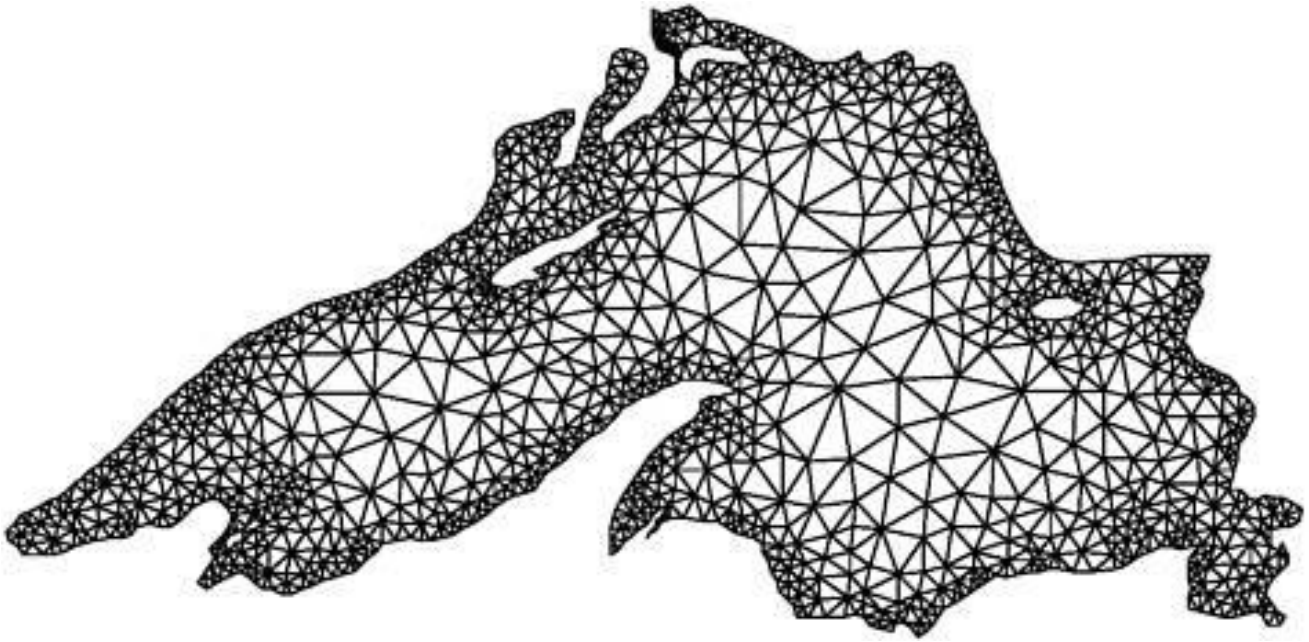
(c)

Mapping Techniques for Load balancing

- **Graph Partitioning**
- The array-based distribution schemes that we described so far are quite effective in balancing the computations and minimizing the interactions for a wide range of algorithms that use dense matrices and have structured and regular interaction patterns. However, there are many algorithms that operate on sparse data structures and for which the **pattern of interaction among data elements is data dependent and highly irregular.**
- Numerical simulations of physical phenomena provide a large source of such type of computations. In these computations, the physical domain is discretized and represented by a mesh of elements.
- The simulation of the physical phenomenon being modeled then involves computing the values of certain physical quantities at each mesh point. The computation at a mesh point usually requires data corresponding to that mesh point and to the points that are adjacent to it in the mesh.

Mapping Techniques for Load balancing

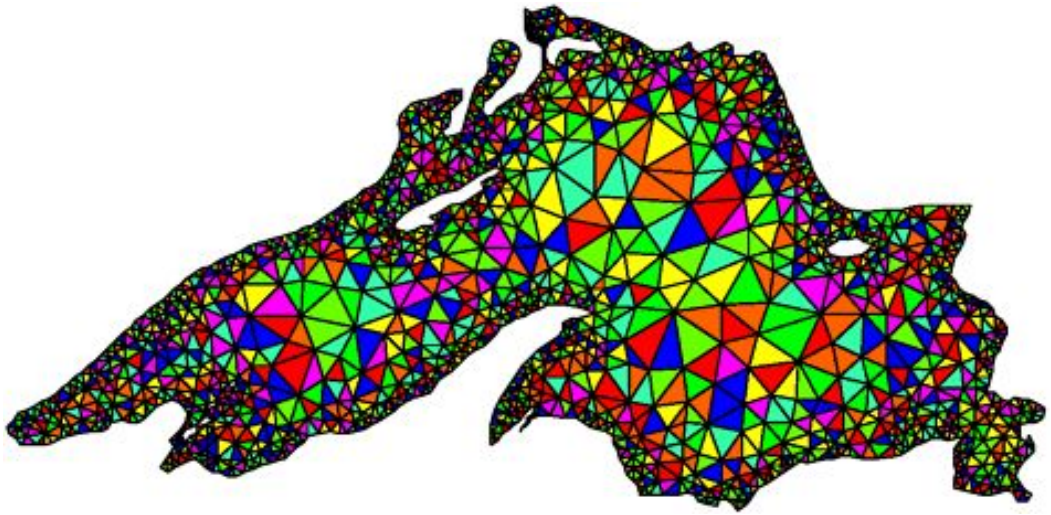
- For example, **Figure shows** a mesh imposed on Lake Superior. The simulation of a physical phenomenon such the dispersion of a water contaminant in the lake would now involve computing the level of contamination at each vertex of this mesh at various intervals of time.



A mesh used to model Lake Superior.

Mapping Techniques for Load balancing

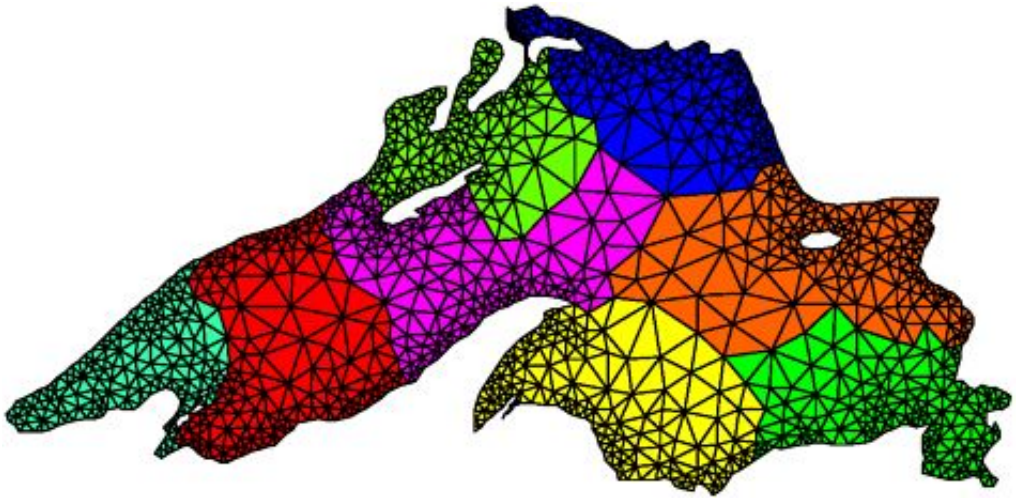
- the amount of computation at each point is the same, the load can be easily balanced by simply assigning the same number of mesh points to each process. However, if a distribution of the mesh points to processes does not strive to keep nearby mesh points together, then it may lead to high interaction overheads due to excessive data sharing.
- For example, if each process receives a random set of points as illustrated in **Figure**, then each process will need to access a large set of points belonging to other processes to complete computations for its assigned portion of the mesh.



A random distribution of the mesh elements to eight processes.

Mapping Techniques for Load balancing

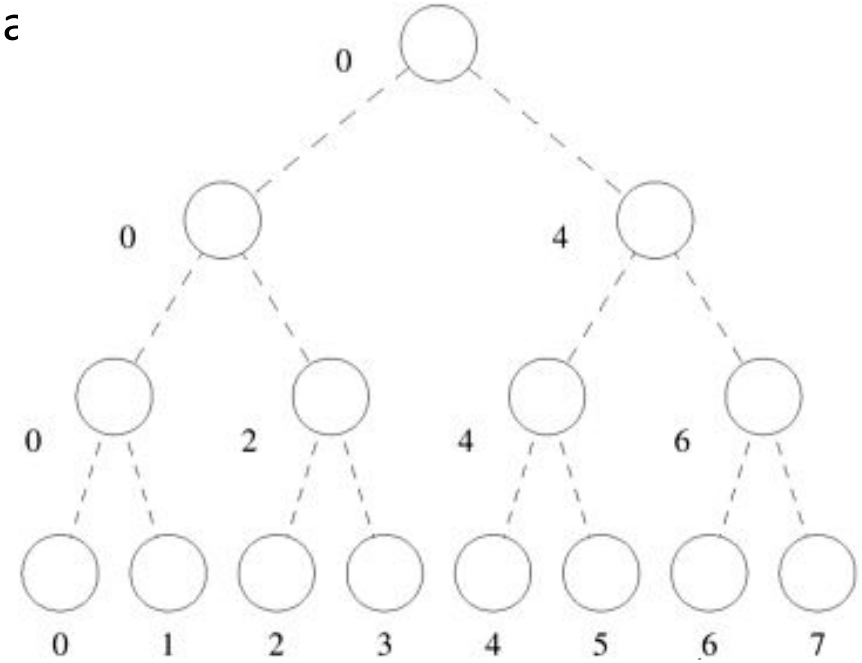
- **Distribute the mesh points in a way that balances the load and at the same time minimizes the amount of data that each process needs to access in order to complete its computations.**
- Partition the mesh into p parts such that each part contains roughly the same number of mesh-points or vertices, and the number of edges that cross partition boundaries is minimized. After partitioning the mesh, each one of these p partitions is assigned to one of the p processes.
- Each process is assigned a contiguous region of the mesh such that the total number of mesh points that needs to be accessed across partition boundaries is minimized.



A distribution of the mesh elements to eight processes, by using a graph-partitioning algorithm.

Mapping Techniques for Load balancing

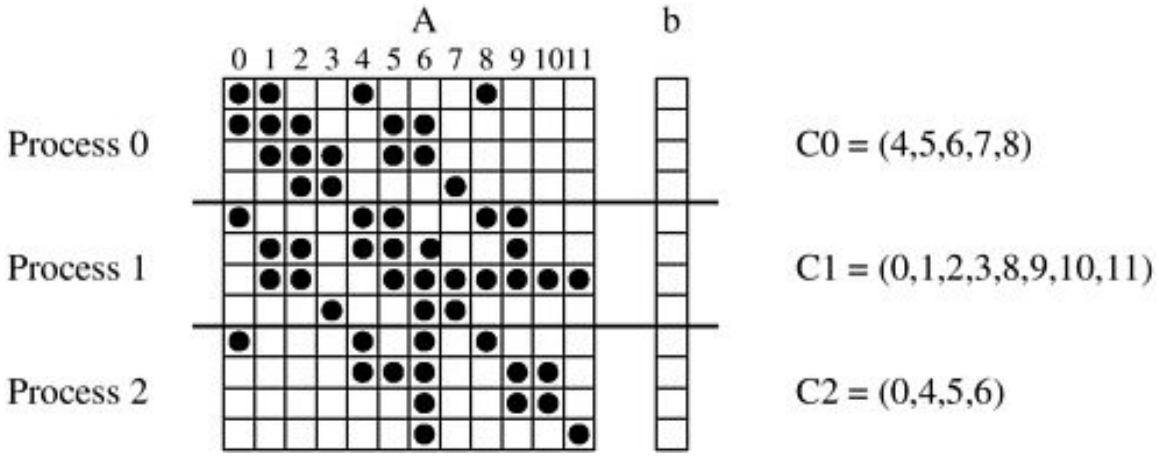
- **Mappings Based on Task Partitioning**
- A mapping based on partitioning a task-dependency graph and mapping its nodes onto processes can be used when the computation is naturally expressible in the **form of a static task-dependency graph with tasks of known sizes.** As usual, this mapping must seek to achieve the often conflicting objectives of minimizing idle time and minimizing the interaction time of the parallel algorithm. **Determining an optimal mapping for a general task-dependency graph is an NP-complete problem**. However, specific situations often lend themselves to a simpler optimal or acceptable approxima



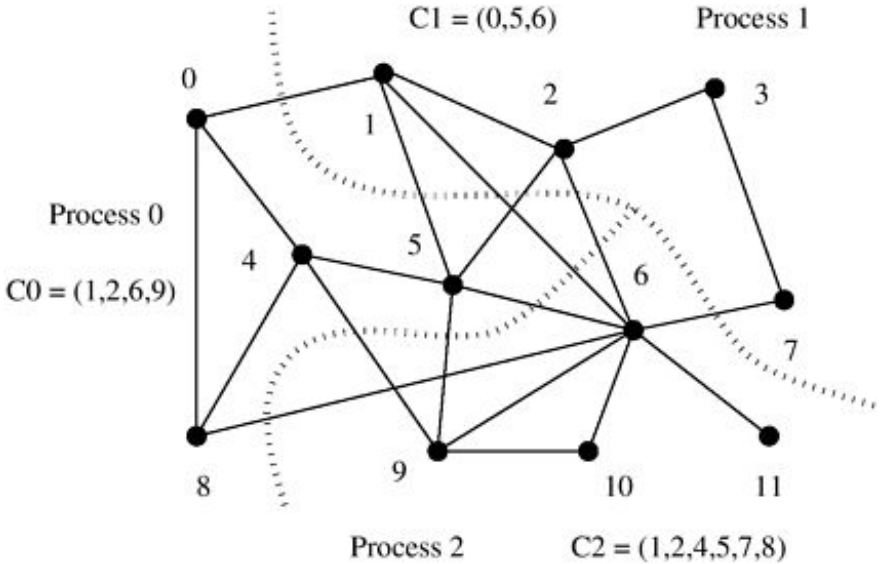
Mapping of a binary tree task-dependency graph onto a hypercube of processes.

Mapping Techniques for Load balancing

- **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.**



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



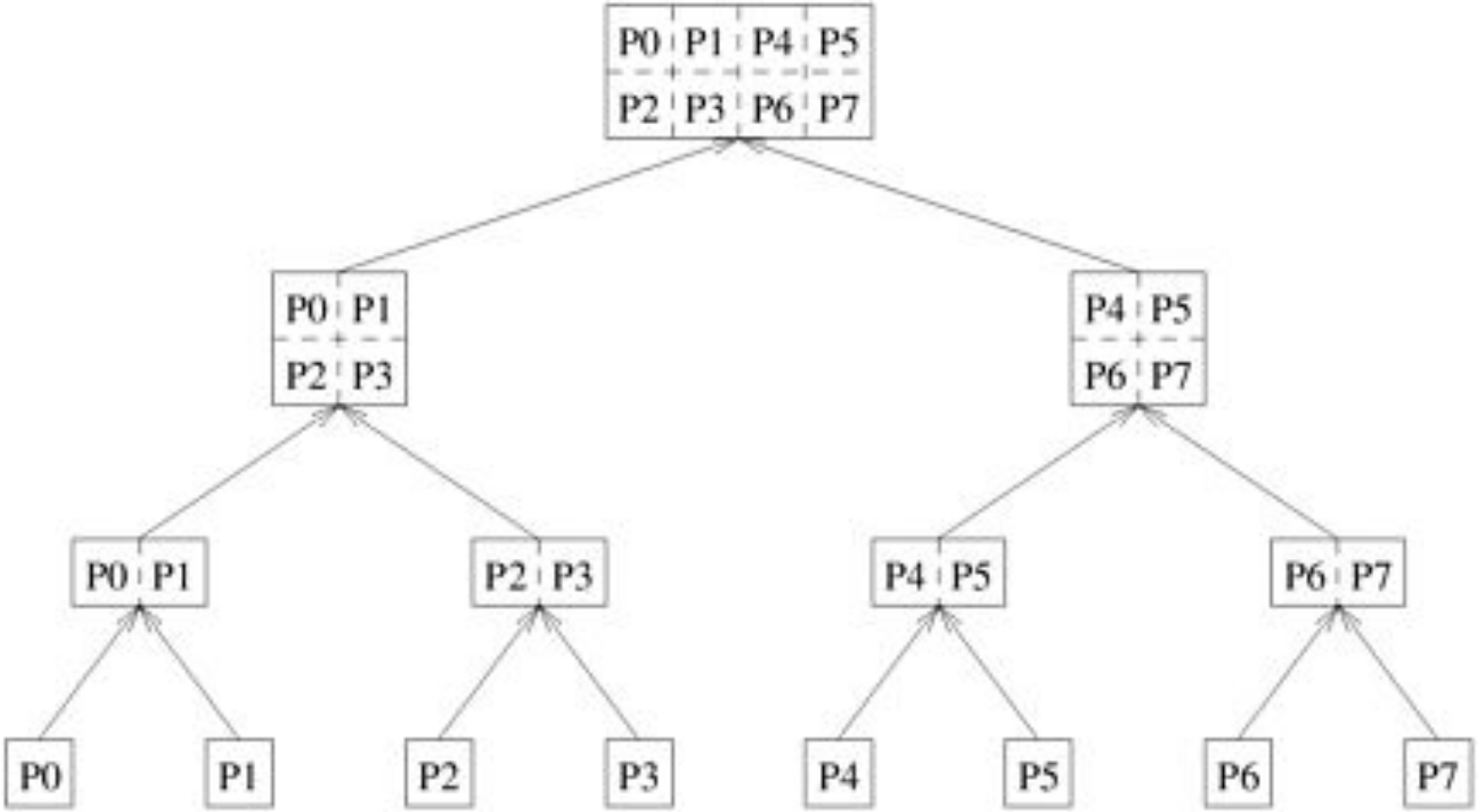
Mapping Techniques for Load balancing

Hierarchical Mappings

- Certain algorithms are naturally expressed as task-dependency graphs;
- However, a mapping based solely on the task-dependency graph may suffer from load-imbalance or inadequate concurrency. For example, in the binary-tree task-dependency graph of Figure, only a few tasks are available for concurrent execution in the top part of the tree. If the tasks are large enough, then a better mapping can be obtained by a further decomposition of the tasks into smaller subtasks.
- In the case where the task-dependency graph is a binary tree with four levels, the root task can be partitioned among eight processes, the tasks at the next level can be partitioned among four processes each, followed by tasks partitioned among two processes each at the next level.
- **The eight leaf tasks can have a one-to-one mapping onto the processes.**
- Figure, illustrates such a hierarchical mapping. Parallel quicksort introduced in Example has a task-dependency graph similar to the one shown in Figure, and hence is an ideal candidate for a hierarchical mapping of the type shown in Figure.

Mapping Techniques for Load balancing

- **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

- Dynamic mapping is necessary in situations where a static mapping may result in a highly imbalanced distribution of work among processes or where the task-dependency graph itself if dynamic, thus precluding a static mapping.
- Since the primary reason for using a dynamic mapping is balancing the workload among processes, dynamic mapping is often referred to as dynamic load-balancing.
- **Dynamic mapping techniques are usually classified as either**

1. **Centralized**
2. **Distributed**

Mapping Techniques for Load balancing

1. Centralized Schemes

- **All executable tasks are maintained in a common central data structure** or they are maintained by a special process or a subset of processes.
- If a special process is designated to manage the pool of available tasks, then it is often referred to as the master and the other processes that depend on the master to obtain work are referred to as slaves.
- Whenever a process has no work, it takes a portion of available work from the central data structure or the master process. Whenever a new task is generated, it is added to this centralized data structure or reported to the master process.
- Centralized load-balancing schemes are usually easier to implement than distributed schemes, but may have limited scalability.
- As more and more processes are used, the large number of accesses to the common data structure or the master process tends to become a bottleneck.

Mapping Techniques for Load balancing

2. Distributed Schemes

- In a distributed dynamic load balancing scheme, the **set of executable tasks are distributed among processes which exchange tasks at run time to balance work.** Each process can send work to or receive work from any other process. These methods do not suffer from the bottleneck associated with the centralized schemes.
- Some of the critical parameters of a distributed load balancing scheme are as follows:
- How are the sending and receiving processes paired together?
- Is the work transfer initiated by the sender or the receiver?
- How much work is transferred in each exchange? If too little work is transferred, then the receiver may not receive enough work and frequent transfers resulting in excessive interaction may be required.
- When is the work transfer performed?
- For example, in receiver initiated load balancing, work may be requested when the process has actually run out of work or when the receiver has too little work left and anticipates being out of work soon.

Mapping Techniques for Load balancing

More details

- http://users.atw.hu/parallelcomp/ch03lev1sec4.html