C-DAC Four Days Technology Workshop

ON

Hybrid Computing – Co-Processors/Accelerators
Power-aware Computing – Performance of Applications Kernels

hyPACK-2013
(Mode-1: Multi-Core)

Lecture Topic :

Multi-Core Processors :
Algorithms & Applications Overview - Part-I

Venue : CMSD, UoHYD ; Date : October 15-18, 2013

Parallel Algorithmic Design

Lecture Outline

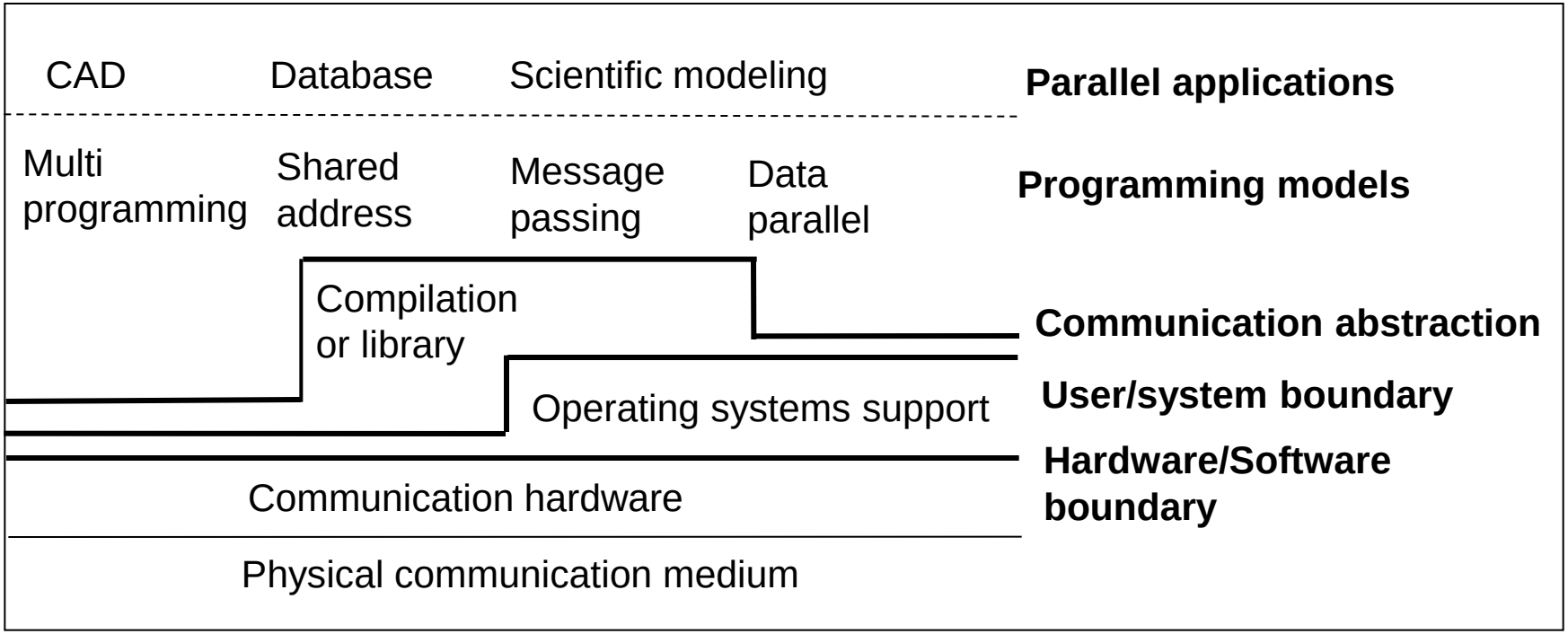
Following Topics will be discussed

- Data parallelism; Task parallelism; Combination of Data and Task parallelism
- Decomposition Techniques
- Static and Load Balancing
  - Mapping for load balancing
  - Minimizing Interaction
  - Overheads in parallel algorithms design
- Data Sharing Overheads

Source : Reference :[1], [4]

Layers of Abstraction in Parallel Computers

- Critical layers of abstraction lie between the application program and actual hardware



Source : Reference :[1], [4]

Parallel Algorithms and Design

Questions to be answered

- How to partition the data?
- Which data is going to be partitioned?
- How many types of concurrency?
- What are the key principles of designing parallel algorithms?
- What are the overheads in the algorithm design?
- How the mapping for balancing the load is done effectively?

Parallel Algorithmic Design

(Contd…)

Principles of Design of parallel algorithms

Two key steps

- Discuss methods for mapping the tasks to processors so that the processors are efficiently utilized.
- Different decompositions and mapping may yield good performance on different computers for a given problem.

It is therefore crucial for programmers to *understand the relationship between the underlying machine model* and *the parallel program to develop efficient programs*

Programming Aspects Examples

Implementation of Streaming Media Player on Multi-Core

- One decomposition of work using Multi-threads
- It consists of
  - A thread Monitoring a network port for arriving data,
  - A decompressor thread for decompressing packets
  - Generating frames in a video sequence
  - A rendering thread that displays frame at programmed intervals

Programming Aspects Example

Implementation of Streaming Media Player on Multi-Core

- The thread must communicate via shared buffers –
  - an in-buffer between the network and decompressor,
  - an out-buffer between the decompressor and renderer
- It consists of
  - Listen to port ……..Gather data from the network
  - Thread generates frames with random bytes (Random string of specific bytes)
  - Render threads pick-up frames & from the out-buffer and calls the display function
  - Implement using the Thread Condition Variables

Basic steps in Designing Parallel Application

- Partitioning
- Communication
- Agglomeration
- Mapping

Source : Reference :[1], [4]

Types of Parallelism

Types of Parallelism

- Data parallelism
- Task parallelism
- Combination of Data and Task parallelism
- Stream parallelism

Types of Parallelism : Task Parallelism

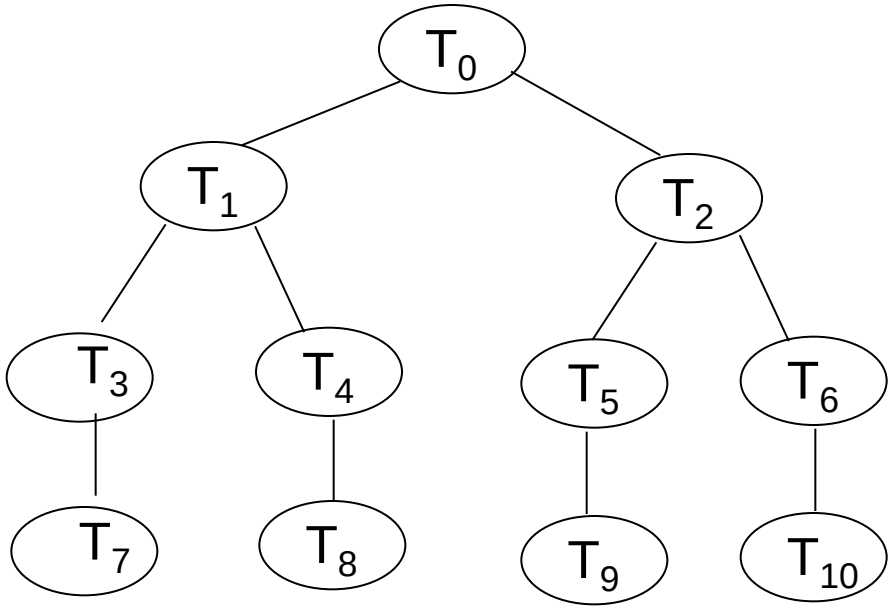
(Contd…)

Task parallelism

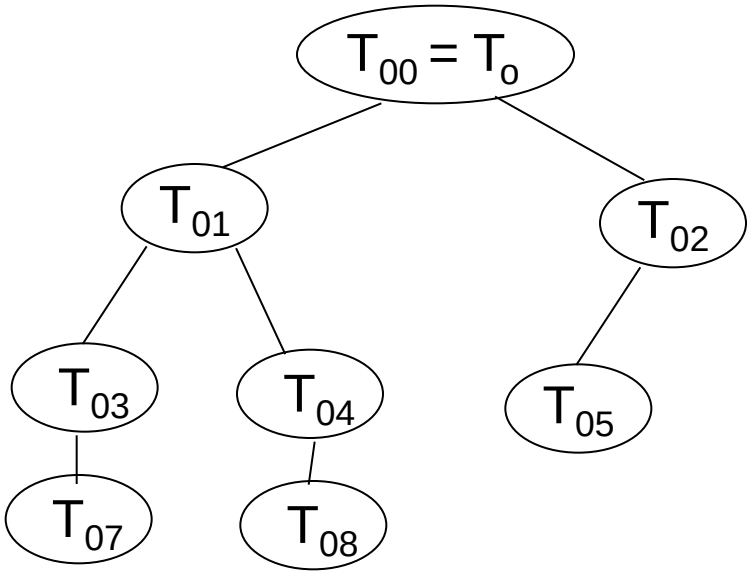
- Many tasks are executed concurrently is called task parallelism.
- This can be done (visualized) by a task graph. In this graph, the node represent a task to be executed. Edges represent the dependencies between the tasks.
- Sometimes, a task in the task graph can be executed as long as all preceding tasks have been completed.
- Let the programmer define different types of processes. These processes communicate and synchronize with each other through MPI or other mechanisms.

Types of Parallelism : Task Parallelism

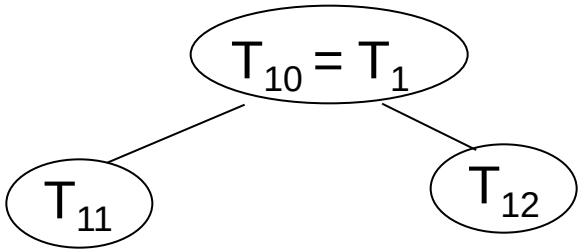
Task Graph



Sub Task Graph for T_o



Sub Task Graph for T_1



Types of Parallelism : Task Parallelism (Contd…)

Programmer's responsibility

Programmer must deal explicitly with process creation, communication and synchronization.

Task parallelism

- Example

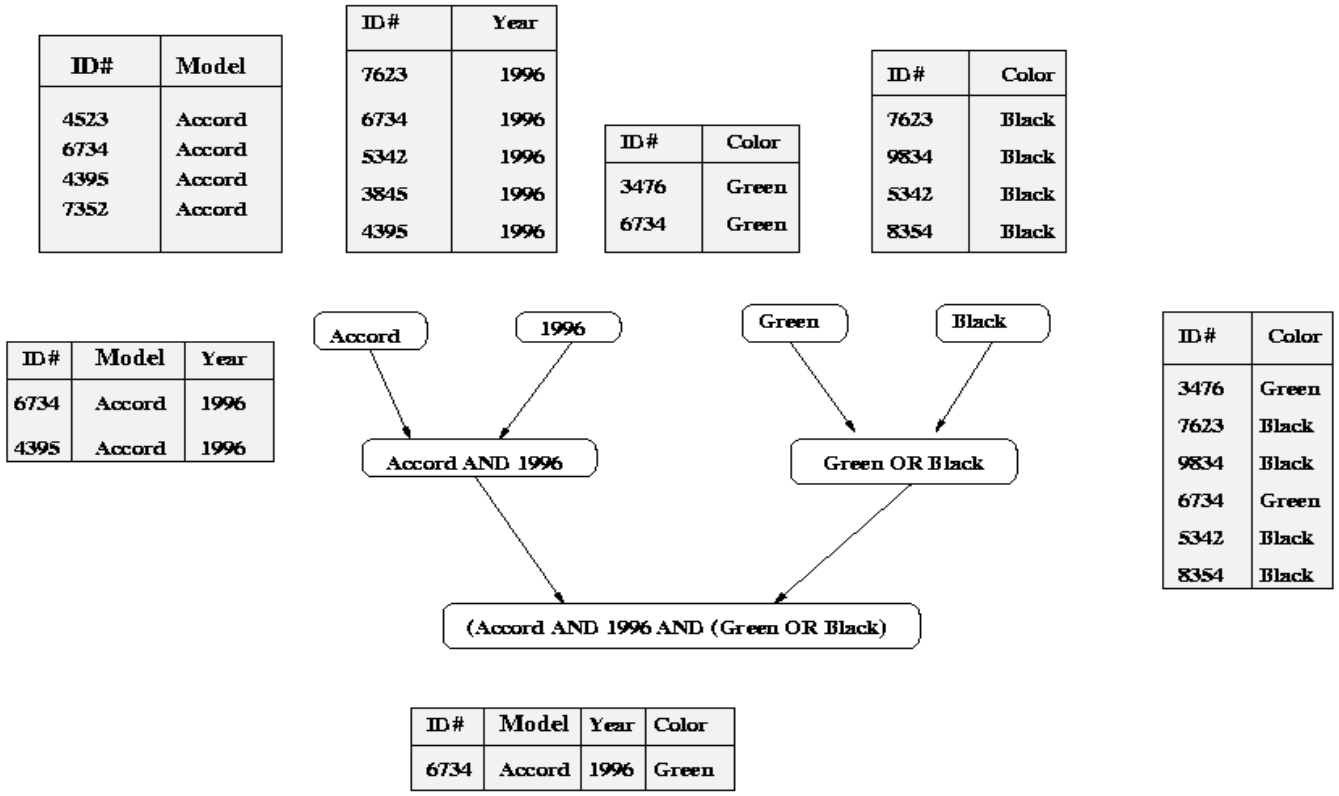
  Vehicle relational database to process the following query

  (MODEL = "-------" AND YEAR = "-------")
  AND (COLOR = "Green" OR COLOR = "Black")

Types of Parallelism : Task Parallelism

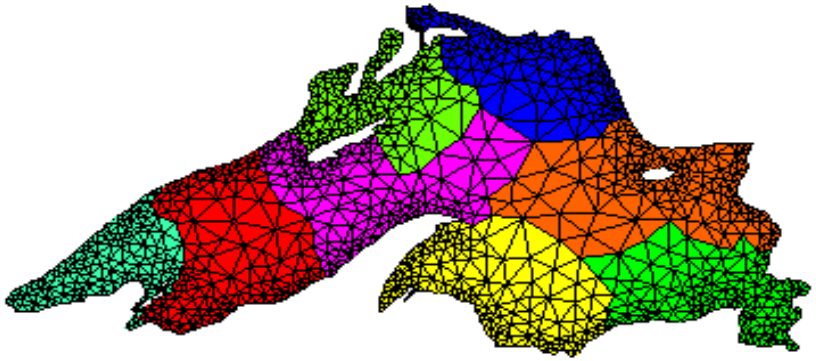
(Contd…)

The different tables generated by processing the query and their dependencies.



Types of Parallelism : Task Parallelism (Contd…)

- Parallel Unstructured Adaptive Mesh/Mesh Repartitioning methods



Finite Element Unstructured Mesh for Lake Superior Region

- HPF / Automatic compiler techniques may not yield good performance for unstructured mesh computations.

- Choosing right algorithm and message passing is a right candidate for partition (decomposition) of unstructured mesh (graph) onto processors. Task parallelism is right to obtain concurrency.

Types of Parallelism : Data and Task Parallelism

(Contd…)

Integration of Task and Data Parallelism

- Two Approaches
  - Add task parallel constructs to data parallel constructs.
  - Add data parallel constructs to task parallel construct
- Approach to Integration
  - Language based approaches.
  - Library based approaches.

Types of Parallelism : Data and Task Parallelism

(Contd…)

Advantages

- Generality
- Ability to increase scalability by exploiting both forms of parallelism in a application.
- Ability to co-ordinate multidisciplinary applications.

Problems

- Differences in parallel program structure
- Address space organization
- Language implementation

Types of Parallelism : Stream Parallelism

Stream Parallelism

- Stream parallelism refers to the simultaneous execution of different programs on a data stream. It is also referred to as *pipelining*.
- The computation is parallelized by executing a different program at each processor and sending intermediate results to the next processor.
- The result is a pipeline of data flow between processors.

Decomposition Techniques

(Contd…)

Decomposition techniques

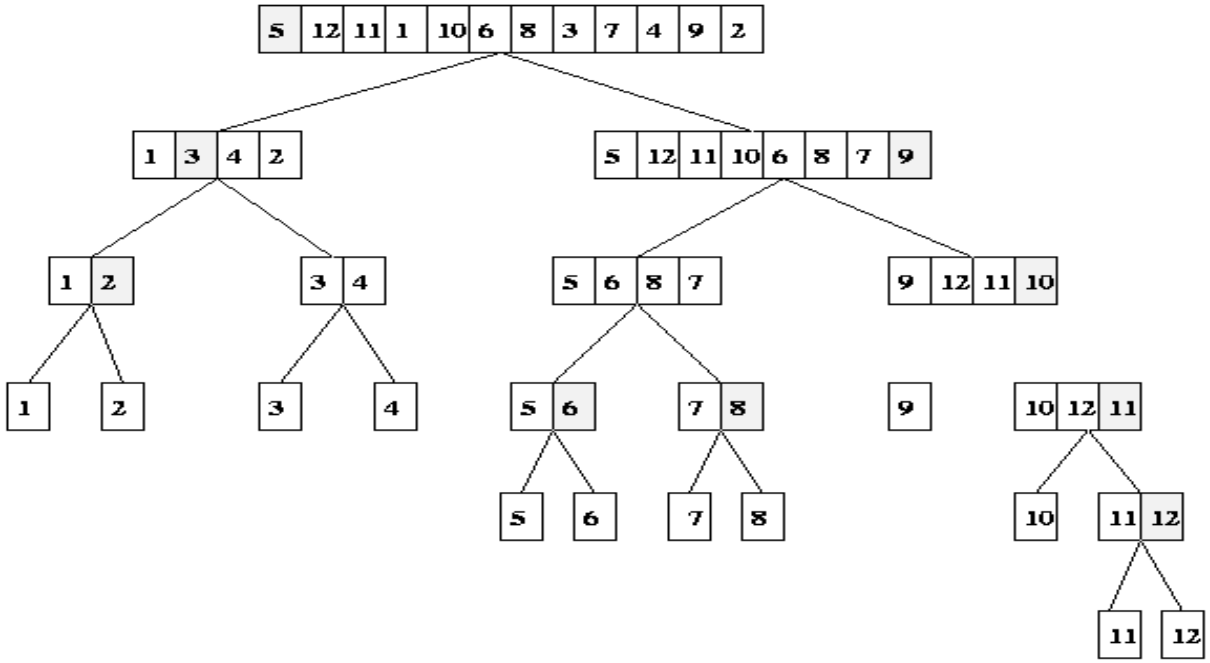
- Recursive decomposition
- Data decomposition
- Exploratory decomposition
- Hybrid decomposition

Source : Reference :[1], [4]

Recursive Decomposition

(Contd…)

The quick sort recursion tree for sorting the sequence of 12 members. Each shaded box corresponds to the pivot element used to split each particular list.



Data Decomposition

Data decomposition

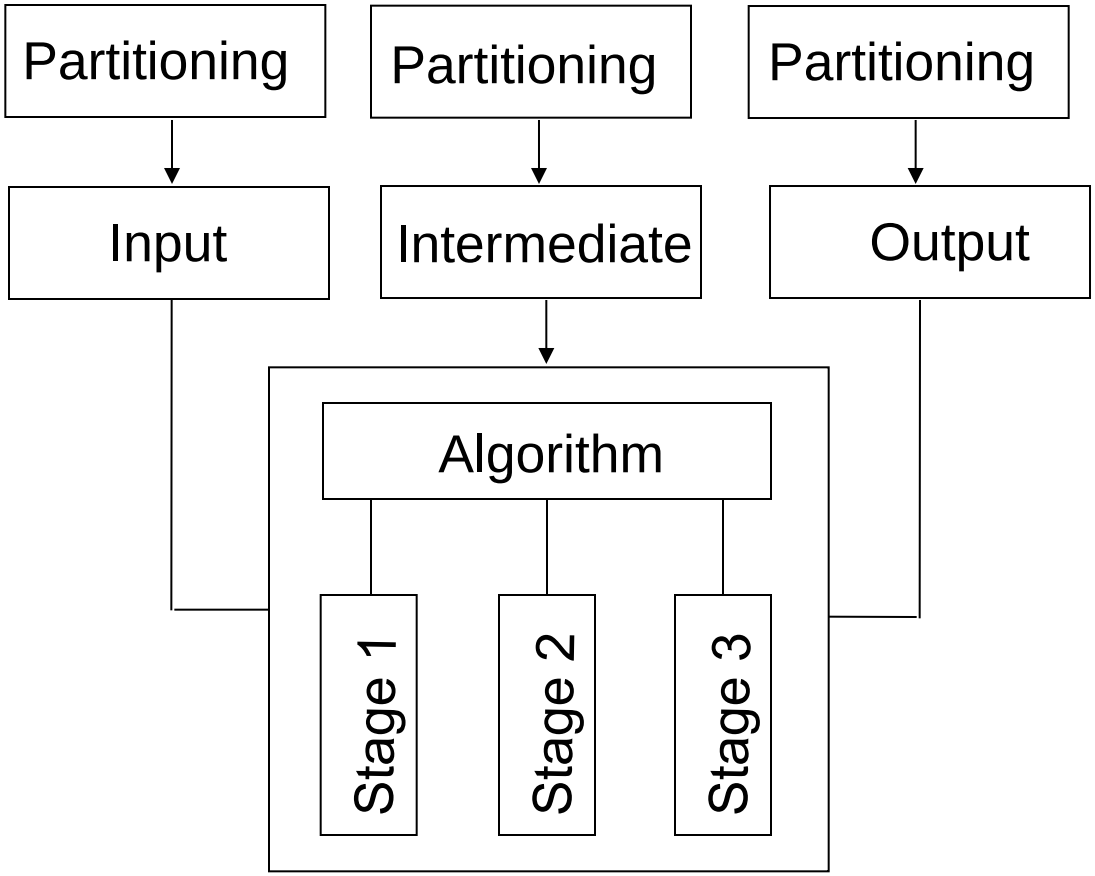
- Data decomposition is a powerful method for deriving concurrency in algorithms that operate on large data structures.

- Decomposition of computations is done in two steps –
  - **First step** : The data (or the domain) on which the computations are performed is partitioned
  - **Second step** : This data partitioning is used to induce a partitioning of the computations into task which leads to data level parallelism

Data Decomposition

(Contd…)

Decomposition of Input, Intermediate and output data



Data Decomposition : Partitioning output data

- Any partitioning of the output suggests a decomposition of the overall computation performed by the algorithm
- The degree of concurrency induced by this decomposition is a powerful method to find concurrency
- This decomposition leads to parallel algorithms that require little or no interaction among various concurrent tasks

Example:

- Adding of *square* matrices A and B to give matrix C = A + B (Partition the computations by partitioning the output matrix into groups each containing an equal number of matrix elements).
- Multiplication of *square* matrices A and B to give matrix C = A * B
- Problem of rendering a scene using the ray-tracing algorithm

Data Decomposition : Partitioning intermediate data

- In many algorithms, computations of some elements of output depends upon the computation of other output elements.
- In such cases, computation of different output elements can not be performed concurrently
- Sometimes these computations can be restructured as multi-stage computation such that each output elements of any of the stages can be computed *independently* of other output elements
  - Parallel Gauss elimination method to solve linear system of matrix equations AX=B
  - LU factorization to solve linear system of matrix equations AX=B
  - Adaptive repartitioning of finite element meshes - dynamic load balancing methods

Data Decomposition : Partitioning Input data

Remark

- Partitioning the output data can be performed if each output can be naturally computed as a function of the input. In many algorithms, this is not possible or desirable to partition the output data.
- It is possible to decompose this computations by partitioning the input array and assigning each input partition to a different task.

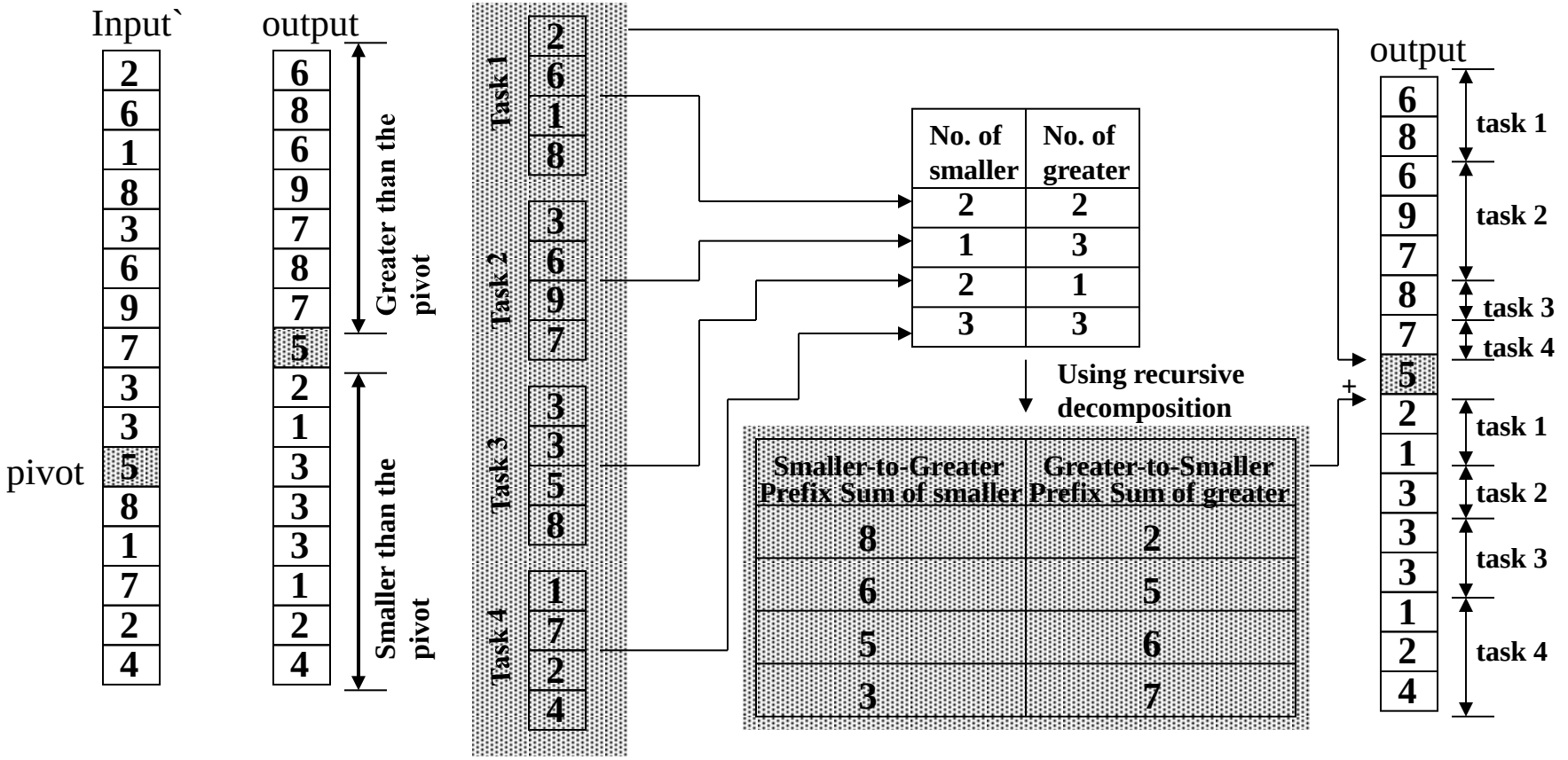
Example : Partitioning Input data: Partitioning a list around a pivot element

- For sorting algorithm, the individual elements of the output cannot be efficiently determined in isolation.
- In such cases, it is sometimes possible to partition the input data and then use this partitioning to induce concurrency

Data Decomposition : Partitioning Input data

(Contd…)

Example : Sorting algorithm



❖Partitioning the input data around a pivot element and obtain concurrency

Exploratory Decomposition

(Contd…)

Exploratory Decomposition

- Used to decompose problems whose underlying computations correspond to a search of a space for solutions.

- Partition search space into smaller parts, and search each one of these parts concurrently, until the desired solutions are found.

Example : A large company wants to award ten prizes to ten of its randomly selected highly valued customers (a highly valued customer is one that buys over $100,000 worth of products annually)

Exploratory Decomposition (Contd…)

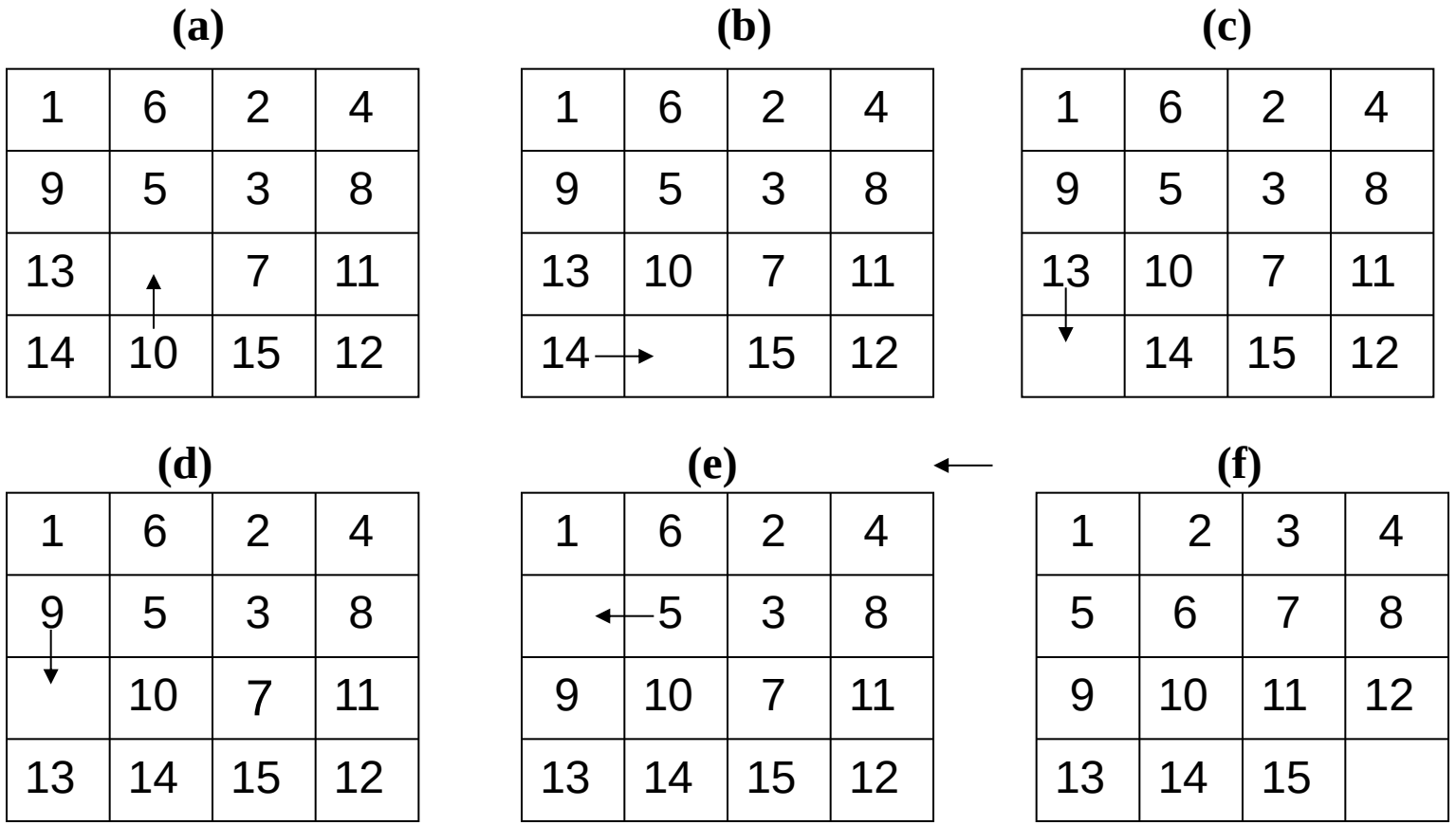
Example : Consider the problem of searching a state space (such as finding a solution to a puzzle problem)

Steps :

- One way of decomposing this computation is to split it into tasks, each one searching a different portion of the search space.
- The task of finding the shortest path from initial to final configuration now translates to finding a path from one of these newly generated nodes to final configuration.
- The 15 puzzle problem is typically solved using tree search techniques. Starting from initial configuration, all possible successor one generated.

Exploratory Decomposition

(Contd…)



A 15-puzzle problem instance : (a) initial configuration; (b)-(e) a sequence of moves leading form the initial to the final configuration. (f) final configuration.

Load Balancing Techniques

Objectives :

- The amount of computation assigned to each processors is balanced so that some processor do not idle while others are executing tasks.
- The interactions among the different processors is minimized, so that the processors spend most of the time in doing work.
- To balance the load among processors, it may be necessary to assign tasks, that interact heavily, to different processors.

Remark : These objectives conflict with each other. The problem of finding a good mapping becomes harder

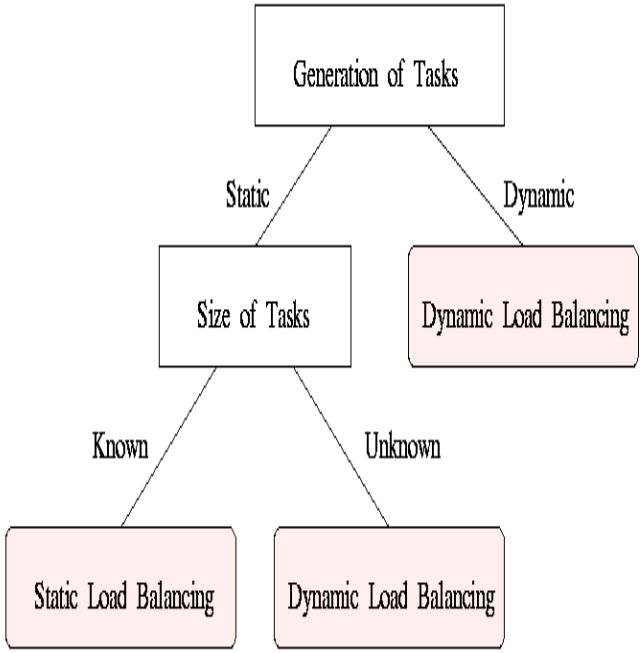
Source : Reference :[1], [4]

Load Balancing Techniques

(Contd…)

❖ **Static load-balancing**

- Distribute the work among processors prior to the execution of the algorithm
- Matrix-Matrix Computation
- Easy to design and implement



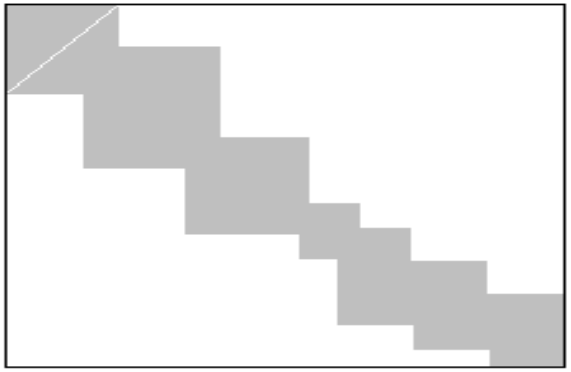
❖ **Dynamic load-balancing**

- Distribute the work among processors during the execution of the algorithm
- Algorithms that require dynamic load-balancing are somewhat more complicated (Parallel Graph Partitioning and Adaptive Finite Element Computations)

Schemes for Static Load Balancing

(Contd…)

Array Distribution schemes : Block Distributions of matrix; One dimensional (strip); Two dimensional (checkerboard);Block cyclic Distributions ;Randomized block 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}

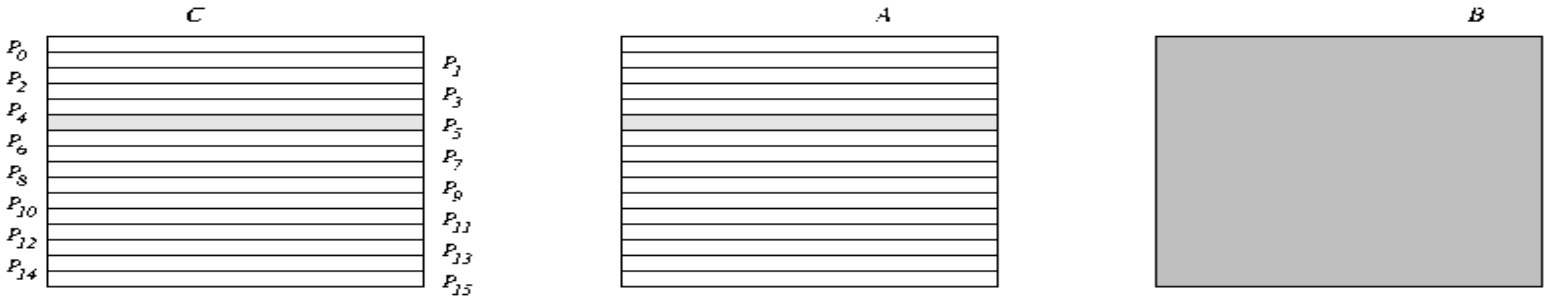
(b)

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

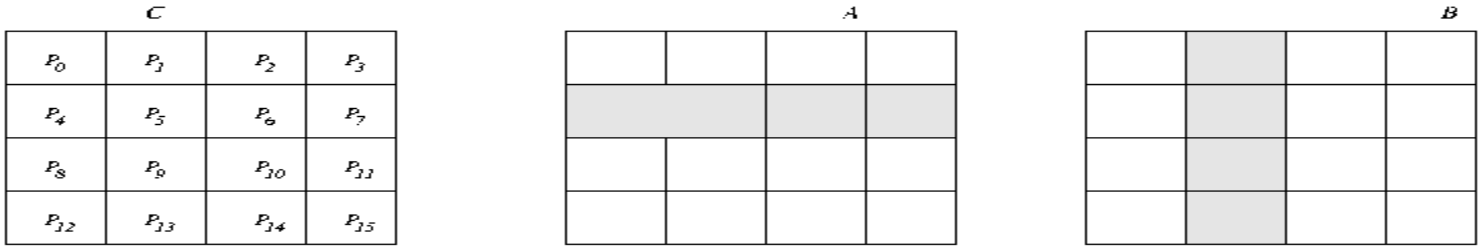
Schemes for Static Load Balancing

Data decomposition : When data decomposition is used to derive concurrency a suitable decomposition of data can itself be used to balance the load and minimize interactions.

Matrix-Matrix addition



Dense matrix-matrix multiplication



Remark : Different decomposition require different amount of data

Schemes for Dynamic Load Balancing

(Contd…)

Dynamic partition

- There are problems in which we cannot statistically partition the work among the processors
- In these problems, a static work partitioning is either impossible (e.g. first class) or can potentially lead to serious load imbalance problems (e.g., second and third classes)
- The only way to develop efficient message passing programs for these classes of problem is if we allow dynamic load balancing.
- Thus during the execution of the program, work is dynamically transferred among the processors that have a lot of work to the processors that have little or no work

Schemes for Dynamic Load Balancing

(Contd…)

- Three general classes of problems that fall into the category
  - The **first** class consists of problems which all the tasks are available at the beginning of the computation but the amount of time required by each task is different and cannot be determined
  - The **second** class consists of problems in which tasks are available at the beginning but as the computation progresses, the amount of time required by each task changes.
  - The **third** class consists of problems in which tasks are generated dynamically

Schemes for Dynamic Load Balancing

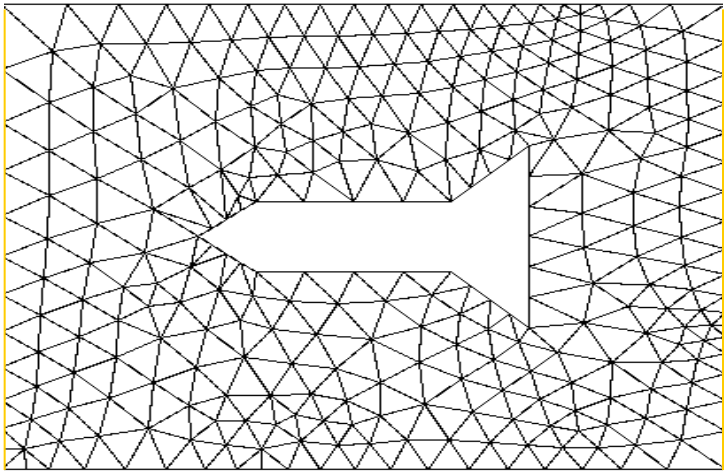
(Contd…)

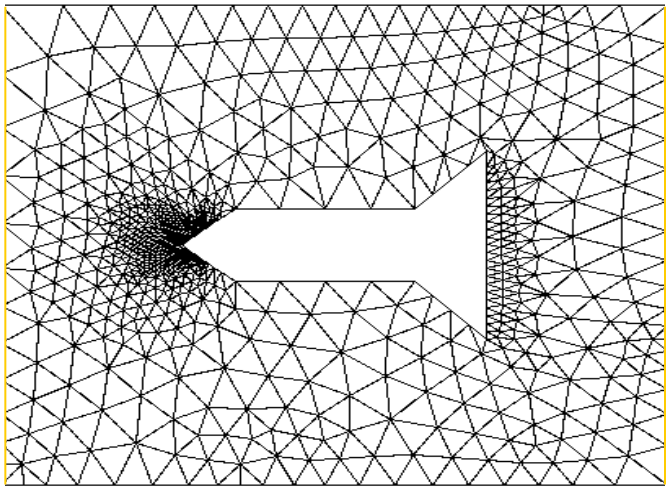
- Self-Scheduling methods
- Chunk Scheduling methods
- Master-Slave paradigm
- Randomization techniques
- Round robin methods

Schemes for Dynamic Load Balancing

(Contd…)

Examples : Adaptive load balancing of unstructured meshes

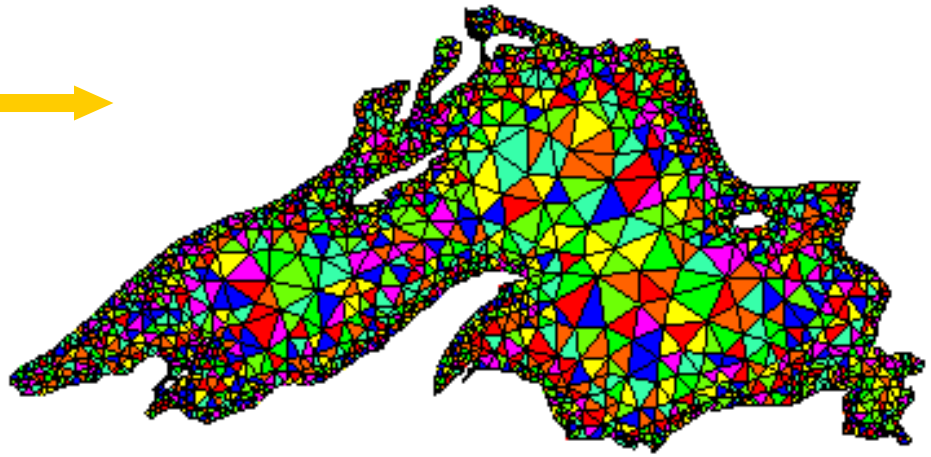


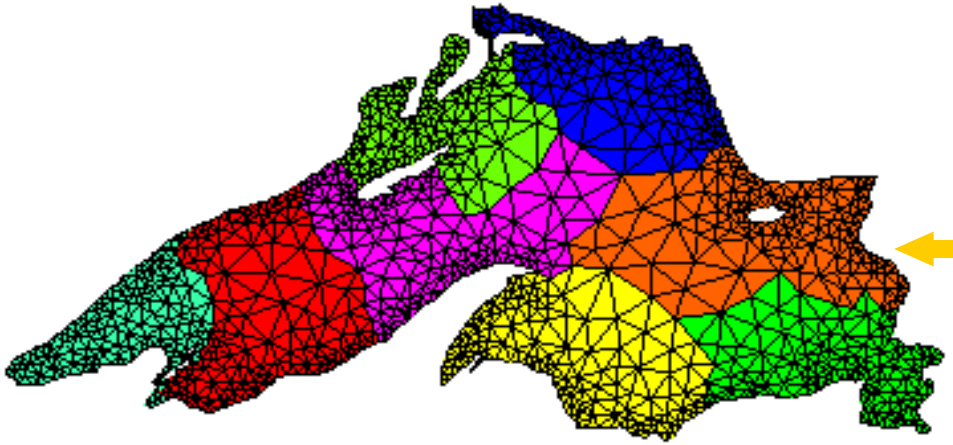


- To refine mesh in parallel often require dynamic load balancing algorithms
  - Use diffusive and non-diffusive algorithms
  - Adapted graph is partitioned from scratch using state of art multilevel graph partitioning.
  - Incremental modification of re-partitioned graphs.
  - Cut and paste re-partitioning algorithm.

Graph Partitioning Algorithms

Random distribution of elements in a two dimensional region

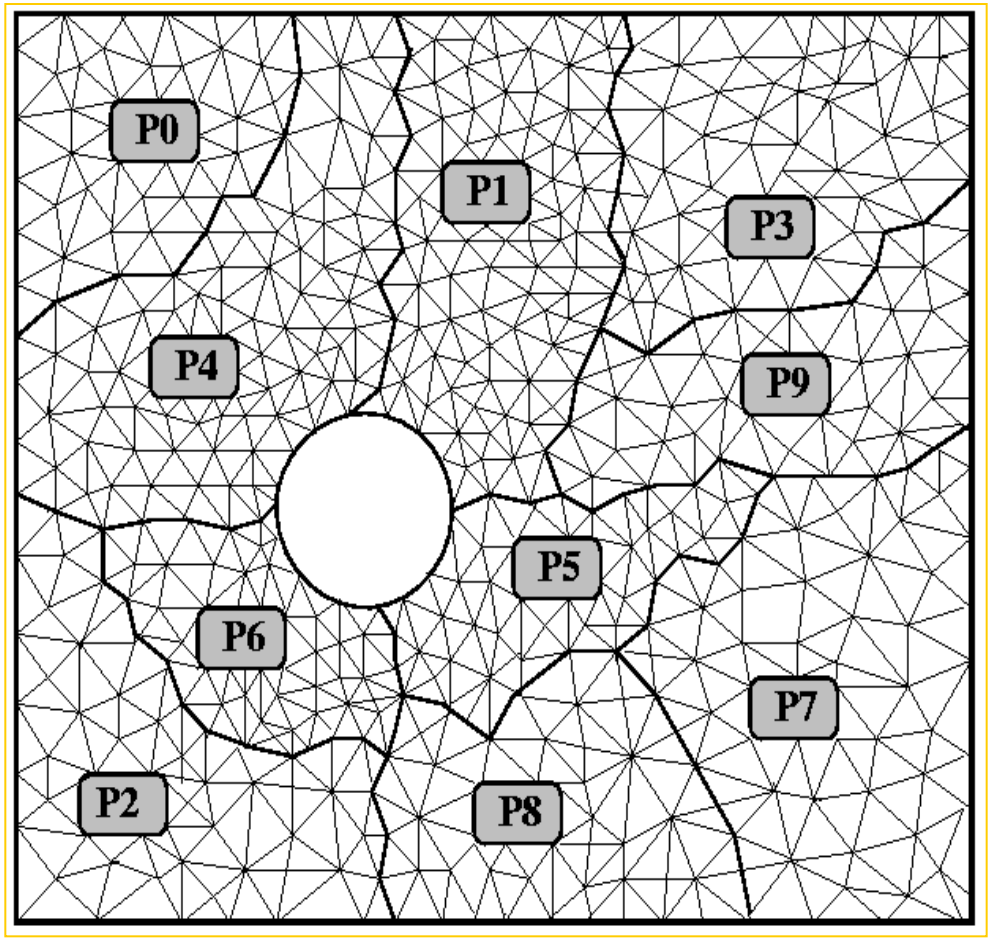




Graph partitioning of a two dimensional region

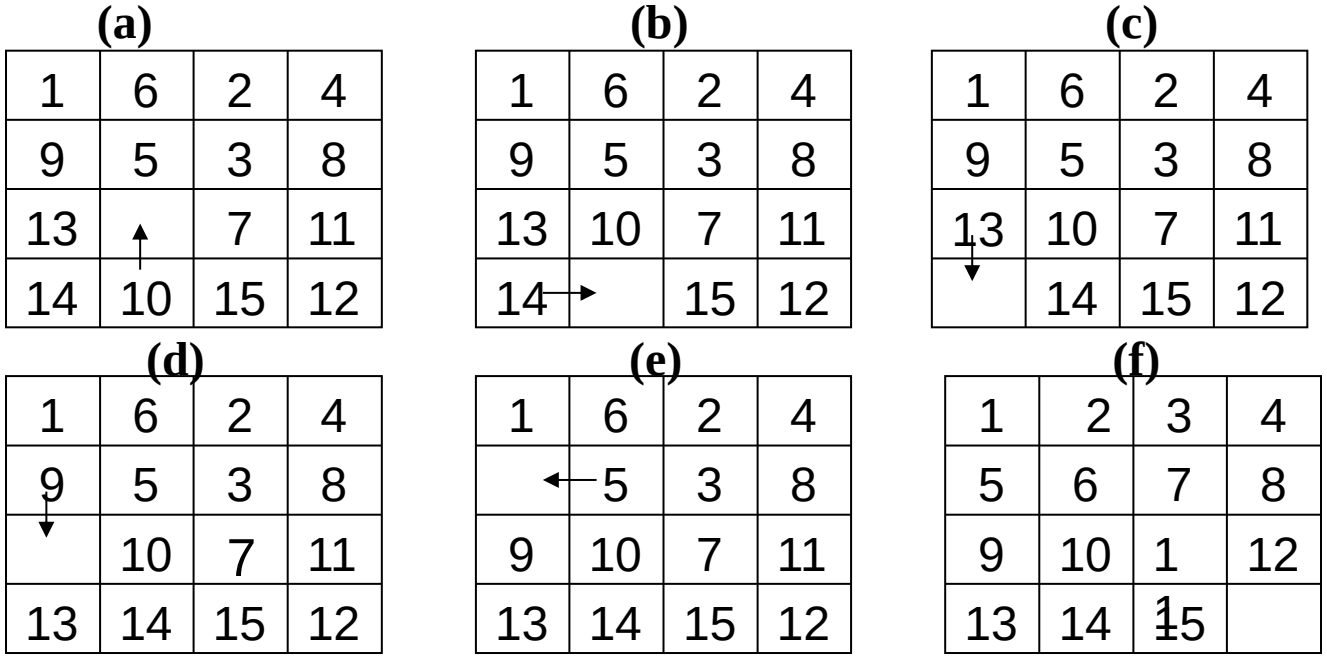
Mesh Distribution for Finite Element Computations

- Elements (evenly distributed)
- Processor level nodes (determined by elements on each processor)
- Global nodes (aligned with processor-level nodes as much as possible)



Dynamic Load Balancing

Example : 15 Puzzle problem



A 15-puzzle problem instance : (a) initial configuration; (b)-(e) a sequence of moves leading form the initial to the final configuration.

In this problem, it is not possible to get an accurate estimate of the work associated with each task, as the size of the tree rooted at any node can vary significantly.

Overheads in Algorithm Design

(Contd…)

Idling overheads :

Processors can become idle for three main reasons.

- The **first** reason is due to load imbalances
- The **second** is due to computational dependencies.
- The **third** is due to inter-processor communication information exchange and synchronization.

Overheads in Algorithm Design

(Contd…)

Data sharing overheads

- In most non-trivial parallel programs, the tasks executed by different processors require access to the same data.
- For example, in matrix-vector multiplication $Y = Ax$, in which tasks correspond to computing contain access to the entire vector x. (Here A is dense and sparse matrix).
- Some tasks require data that are generated dynamically by earlier tasks.

Overheads in Algorithm Design

(Contd…)

Data sharing overheads

- Data sharing can take different forms which depend on the underlying architecture.
  - On distributed memory machines, data is initially partitioned among processors.
  - On shared address space machines, data sharing is often performed implicitly.

Overheads in Algorithm Design

(Contd…)

Data sharing overheads

- Developing algorithms that minimize data sharing overheads often involves intelligent task decomposition,
  - Task distribution
  - Initial data distribution
  - Proper scheduling of the data sharing operations
  - Collective data sharing operations

Overheads in Algorithm Design

(Contd…)

Point-to-Point data sharing operations

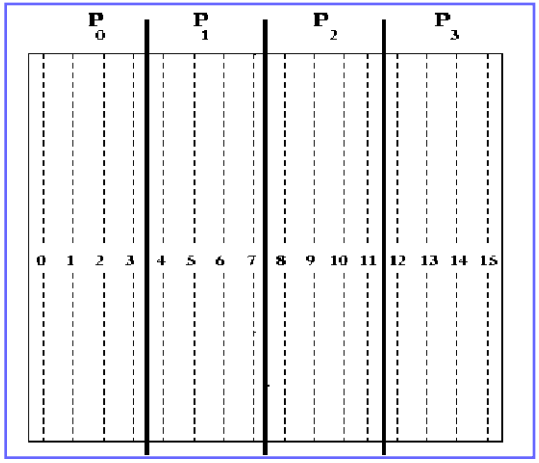
- Collective Data Sharing Operations
- Collective Data Transfers
  - Broadcast
  - Gather
  - Scatter
  - All-to-All
- Collective Computations
  - Reduction
- Collective Synchronization
  - Barrier

Overheads in Algorithm Design

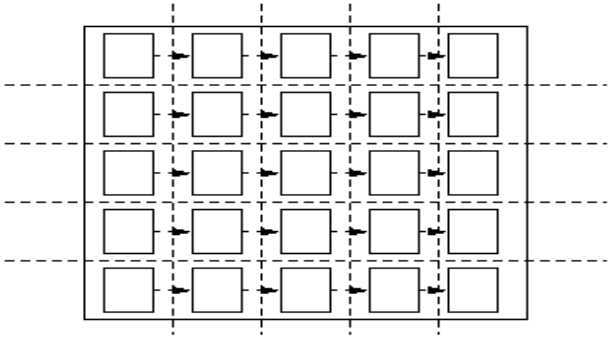
(Contd…)

Cost of Data Sharing Operations

- Structured Data Sharing
  - Dense Matrix-Matrix Multiplication

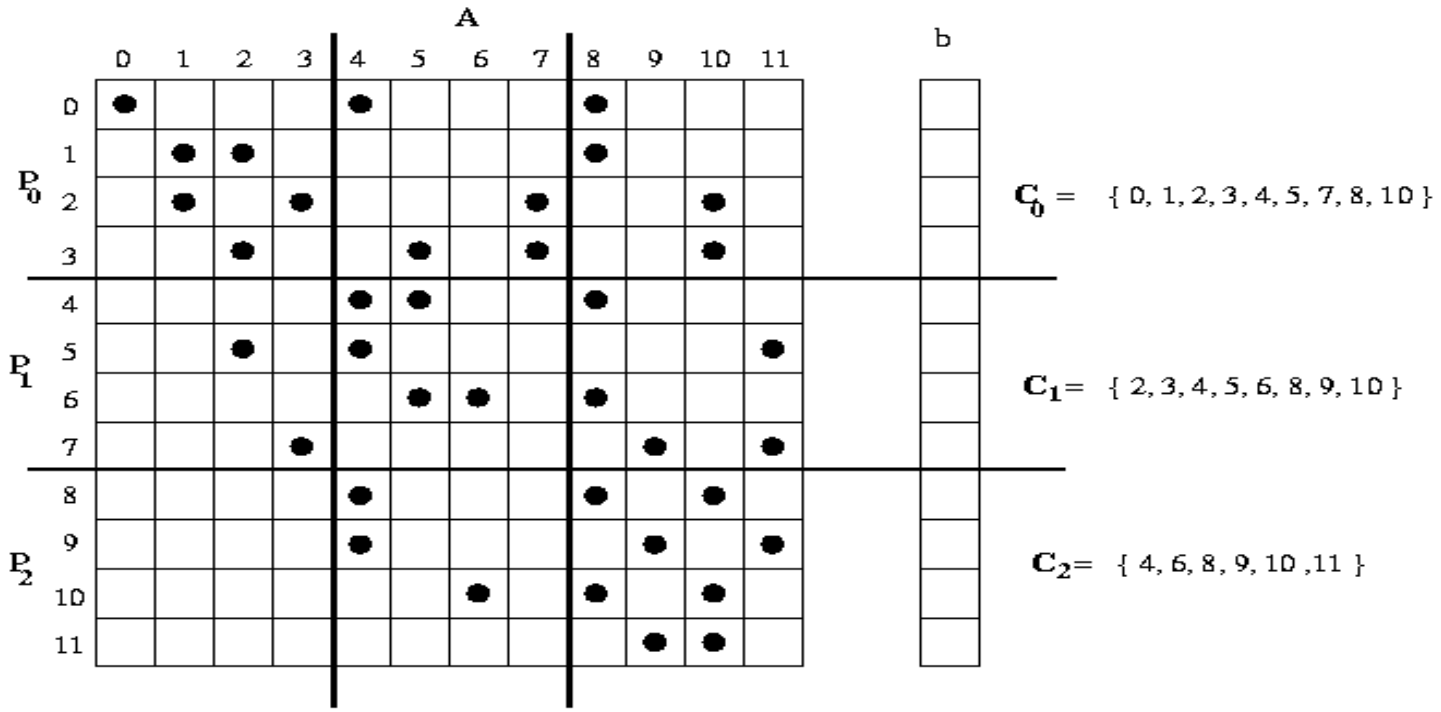


- Unstructured Data Sharing
  - Sparse Matrix-Vector Multiplication



Sparse Matrix A and Vector Multiplication

Sparse Matrix A and the vector b Multiplication



P_1 needs to receive elements {8,9,10} from P_2.
P_2 needs to receive only elements {4,6} from P_1.

Overheads in Algorithm Design

(Contd…)

Extraneous Computations

- Parallel algorithms can potentially perform two types of extraneous computations.
  - The first type consists of non-essential computations that are performed by the parallel algorithms but they are not required by the serial one.
  - In parallel algorithms, non-essential computation often arise either because of speculative execution one.
- The second type of extraneous computation happens when more than one processor ends up performing the same computations.

Example : Parallel search of a sparse graph

Overheads in Algorithm Design (Contd…)

Methods for containing interactions overheads

- Data sharing overheads
  - Static *vs* Dynamic
  - Data sharing interactions

Example : Sparse matrix-vector multiplication

- Regular vs irregular interactions

Remark : One must strive to minimize the interaction overhead in the same way as the user to minimize the interaction overhead due to idling and extraneous computations.

- Two general schemes for dealing with interaction overheads.
  - Perform useful work while the interaction takes place
  - Try to reduce the interactions parallel programs

Disadvantages using Multi Core

(Contd…)

- Maximum Utilization of the computing resources provides by Multi-Core processors
- Ability of multi-core processors to increase application performance depends on the use of multiple threads within applications.
- More difficult to manage thermally.
- Two processing cores sharing he same system bus and memory bandwidth limits
- Improve energy efficiency - focusing on performance –per-watt

Overheads in Algorithm Design

(Contd…)

Conclusions

Five step process for the design of parallel programs. These steps are described as follows

- Identification of type of parallelism
- Decomposition to expose concurrency
- Mapping and interaction cost
- Extraneous computations
- Handling overheads

References

1. Andrews, Grogory R. **(2000),** Foundations of Multithreaded, Parallel, and Distributed Programming, Boston, MA : Addison-Wesley

2. Butenhof, David R **(1997),** Programming with POSIX Threads , Boston, MA : Addison Wesley Professional

3. Culler, David E., Jaswinder Pal Singh **(1999),** Parallel Computer Architecture - A Hardware/Software Approach , San Francsico, CA : Morgan Kaufmann

4. Grama Ananth, Anshul Gupts, George Karypis and Vipin Kumar **(2003),** Introduction to Parallel computing, Boston, MA : Addison-Wesley

5. Intel Corporation, **(2003),** Intel Hyper-Threading Technology, Technical User's Guide, Santa Clara CA : Intel Corporation Available at : http://www.intel.com

6. Shameem Akhter, Jason Roberts **(April 2006),** Multi-Core Programming - Increasing Performance through Software Multi-threading , Intel PRESS, Intel Corporation,

7. Bradford Nichols, Dick Buttlar and Jacqueline Proulx Farrell **(1996),** Pthread Programming O'Reilly and Associates, Newton, MA 02164,

8. James Reinders, Intel Threading Building Blocks – (**2007**) , O'REILLY series

9. Laurence T Yang & Minyi Guo (Editors), (**2006**) *High Performance Computing - Paradigm and Infrastructure* Wiley Series on Parallel and Distributed computing, Albert Y. Zomaya, Series Editor

10. Intel Threading Methodology ; Principles and Practices Version 2.0 copy right **(March 2003),** Intel Corporation

References

11. William Gropp, Ewing Lusk, Rajeev Thakur **(1999),** Using MPI-2, Advanced Features of the Message-Passing Interface, The MIT Press..

12. Pacheco S. Peter, **(1992),** Parallel Programming with MPI, , University of Sanfrancisco, Morgan Kaufman Publishers, Inc., Sanfrancisco, California

13. Kai Hwang, Zhiwei Xu, (**1998**), Scalable Parallel Computing (Technology Architecture Programming), McGraw Hill New York.

14. Michael J. Quinn (**2004**), Parallel Programming in C with MPI and OpenMP McGraw-Hill International Editions, Computer Science Series, McGraw-Hill, Inc. Newyork

15. Andrews, Grogory R. **(2000),** Foundations of Multithreaded, Parallel, and Distributed Progrmaming, Boston, MA : Addison-Wesley

16. SunSoft. Solaris multithreaded programming guide. SunSoft Press, Mountainview, CA, **(1996),** Zomaya, editor. Parallel and Distributed Computing Handbook. McGraw-Hill,

17. Chandra, Rohit, Leonardo Dagum, Dave Kohr, Dror Maydan, Jeff McDonald, and Ramesh Menon, **(2001)**,Parallel Programming in OpenMP San Fracncisco Moraan Kaufmann

18. S.Kieriman, D.Shah, and B.Smaalders **(1995),** Programming with Threads, SunSoft Press, Mountainview, CA. 1995

19. Mattson Tim, **(2002),** Nuts and Bolts of multi-threaded Programming Santa Clara, CA : Intel Corporation, Available at : http://www.intel.com

20. I. Foster **(1995,** Designing and Building Parallel Programs ; Concepts and tools for Parallel Software Engineering, Addison-Wesley (1995)

21. J.Dongarra, I.S. Duff, D. Sorensen, and H.V.Vorst **(1999),** Numerical Linear Algebra for High Performance Computers (Software, Environments, Tools) SIAM, 1999

References

22. OpenMP C and C++ Application Program Interface, Version 1.0". **(1998),** OpenMP Architecture Review Board. October 1998

23. D. A. Lewine. *Posix Programmer's Guide:* **(1991),** Writing Portable Unix Programs with the Posix. 1 Standard. O'Reilly & Associates, 1991

24. Emery D. Berger, Kathryn S McKinley, Robert D Blumofe, Paul R.Wilson, *Hoard : A Scalable Memory Allocator for Multi-threaded Applications* ; The Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX). Cambridge, MA, November (**2000)**. Web site URL : http://www.hoard.org/

25. Marc Snir, Steve Otto, Steyen Huss-Lederman, David Walker and Jack Dongarra, (**1998**) *MPI-The Complete Reference: Volume 1, The MPI Core, second edition* [MCMPI-07].

26. William Gropp, Steven Huss-Lederman, Andrew Lumsdaine, Ewing Lusk, Bill Nitzberg, William Saphir, and Marc Snir (**1998**) *MPI-The Complete Reference: Volume 2, The MPI-2 Extensions*

27. A. Zomaya, editor. Parallel and Distributed Computing Handbook. McGraw-Hill, **(1996)**

28. OpenMP C and C++ Application Program Interface, Version 2.5 (**May 2005**)", From the OpenMP web site, URL **: http://www.openmp.org/**

29. Stokes, Jon 2002 Introduction to Multithreading, Super-threading and Hyper threading *Ars Technica*, October **(2002)**

30. Andrews Gregory R. 2000, Foundations of Multi-threaded, Parallel and Distributed Programming, Boston MA : Addison – Wesley (**2000)**

31. Deborah T. Marr , Frank Binns, David L. Hill, Glenn Hinton, David A Koufaty, J . Alan Miller, Michael Upton, "Hyperthreading, Technology Architecture and Microarchitecture", Intel (**2000-01)**

Thank You

Any questions ?