Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Parallel Scalable Algorithms

Vassil Alexandrov, *ICREA* - **Barcelona Supercomputing Center, Spain**

Lecture Plan:

Day 1

Session 1 / 10:00 am – 1:00 pm

1. Introduction to parallel algorithms design and performance parameters (1 h)

2. Tareador: tool to analyze potential parallelism and its inhibitors (2 h)

3. Practical: simple heat diffusion application

Session 2 / 2:00pm – 5:00 pm

1. Introduction to Paraver: tool to analyze and understand performance

2. Practical: hybrid MPI/OpenMP trace analysis

Lecture Plan:

Day 2

Session 1 / 10:00 am - 1:00 pm

1. Introduction to MPI: Overview of MPI

2. Point-to-point communication, collective communication

3. Practical: How to compile and run MPI applications, distributed matrix computations (matrix multiply or/and heat equation)

Session 2 / 2:00 pm - 5:00 pm

1. Blocking and non-blocking communications

2. Communicators, Topologies

3. Practical: Distributed matrix computations (same examples as in session 1)

Lecture Plan:

Day 3

Session 1 / 10:00 am - 1:00 pm

1. MPI I/O issues, Error handling, Parallel libraries

2. Scalability, xSim and Dimemas simulator

3. Practical: Scalability simulations using xSim and Dimemas

Session 2 / 2:00 pm - 5:00 pm

1. Practical: Scalability simulations using xSim and Dimemas (cont.)

2. Outlook - Fault tolerance, FT-MPI, MPI 3.0

Lecture Plan:

Day 4

Session 1 / 10:00am – 1:00 pm

1. Shared-memory programming models, OpenMP fundamentals
2. Parallel regions and work sharing constructs
3. Synchronization mechanisms in OpenMP
4. Practical: heat diffusion

Session 2 / 2:00pm – 5:00 pm

1. Tasking in OpenMP
2. Programming using a hybrid MPI/OpenMP approach
3. Practical: Multisort (tasking) and heat diffusion (hybrid MPI/OpenMP)

Lecture Plan:

Day 5

Session 1 / 10:00 am – 1:00 pm

1. Introduction to the OmpSs programming model

2. Programming using a hybrid MPI/OmpSs approach

3. Practical: TBD

END of COURSE

1st Lecture Outline:

- Introduction
- Parallel Algorithms and Parallelisation Techniques
- Performance Evaluation and Performance Metrics

Top 500 list – trends and performance

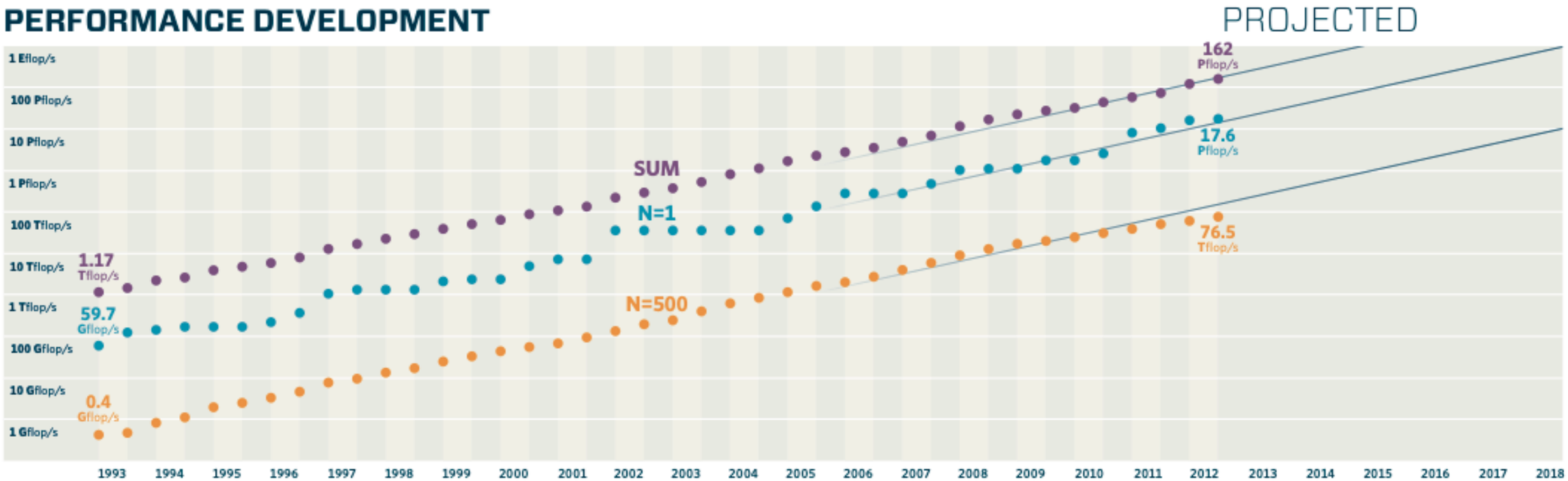
TOP 500® NOVEMBER 2012

PRESENTED BY UNIVERSITY OF MANNHEIM · ICL UT INNOVATIVE COMPUTING LABORATORY THE UNIVERSITY of TENNESSEE · BERKELEY LAB Lawrence Berkeley National Laboratory

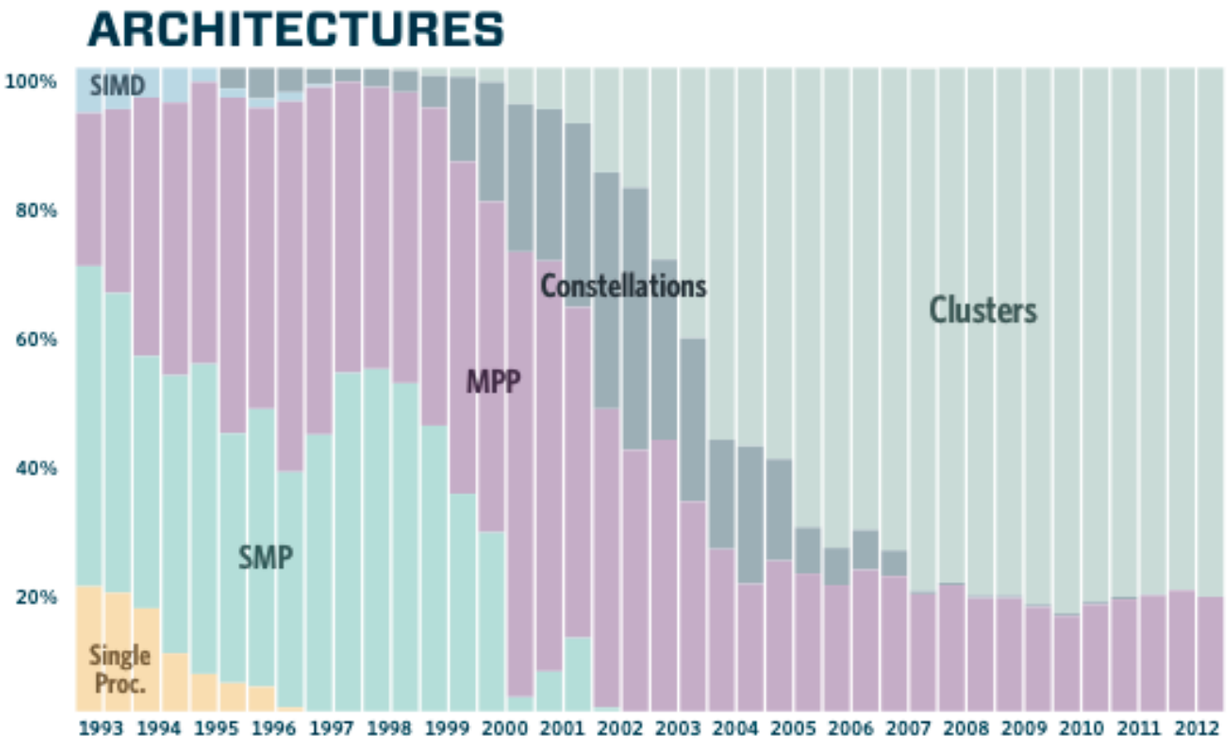
FIND OUT MORE AT www.top500.org

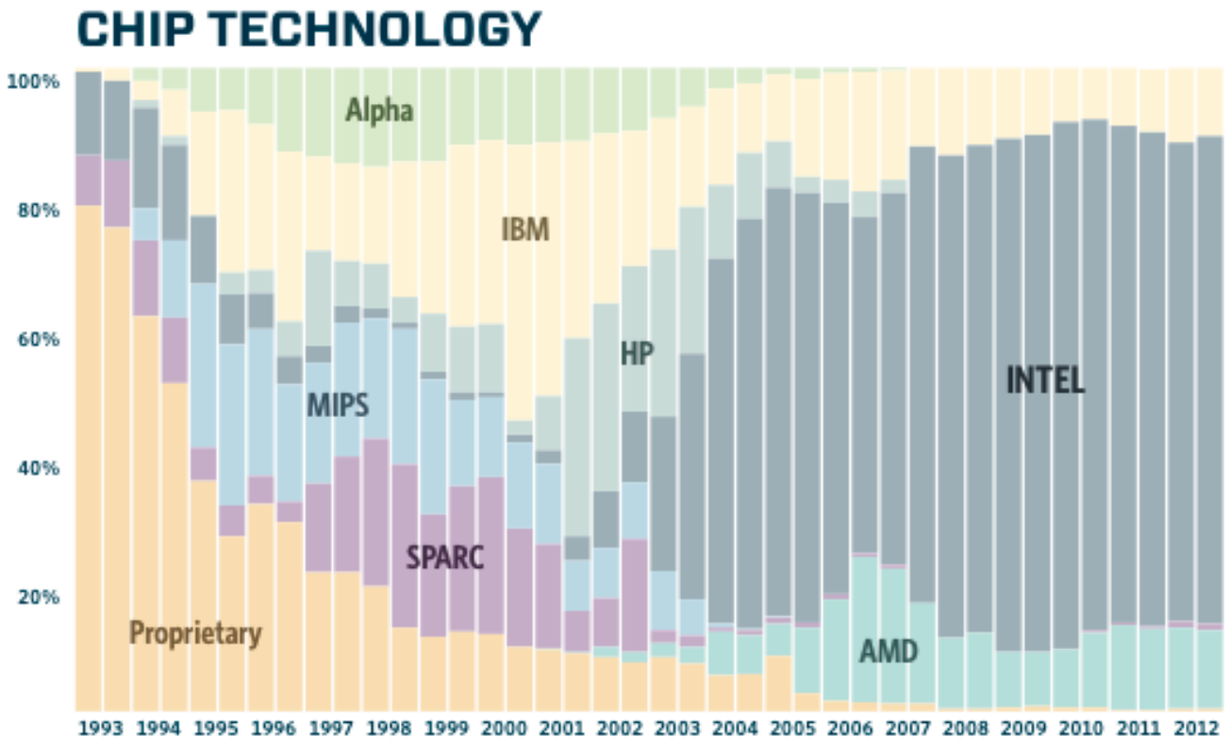
	NAME	SPECS	SITE	COUNTRY	CORES	RMAX PFLOP/S	POWER MW
1	TITAN	Cray XK7, Operon 6274 16C 2.2 GHz + Nvidia Kepler GPU, Custom interconnect	DOE/OS/ORNL	USA	560,640	**17.6**	8.3
2	SEQUOIA	IBM BlueGene/Q, Power BQC 16C 1.60 GHz, Custom interconnect	DOE/NNSA/LLNL	USA	1,572,864	**16.3**	7.9
3	K COMPUTER	Fujitsu SPARC64 VIIIfx 2.0GHz, Custom interconnect	RIKEN AICS	Japan	705,024	**10.5**	12.7
4	MIRA	IBM BlueGene/Q, Power BQC 16C 1.60 GHz, Custom interconnect	DOE/OS/ANL	USA	786,432	**8.16**	3.95
5	JUQUEEN	IBM BlueGene/Q, Power BQC 16C 1.60 GHz, Custom interconnect	Forschungszentrum Jülich	Germany	393,216	**4.14**	1.97

PERFORMANCE DEVELOPMENT

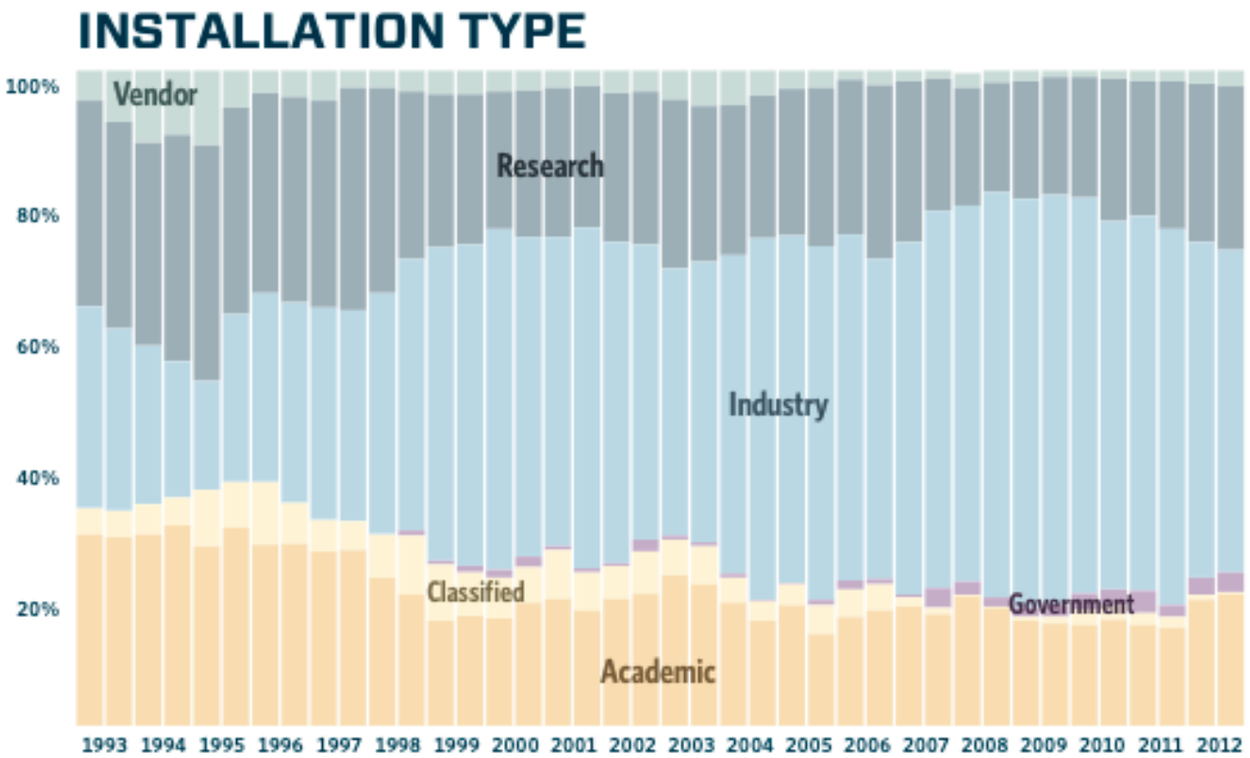


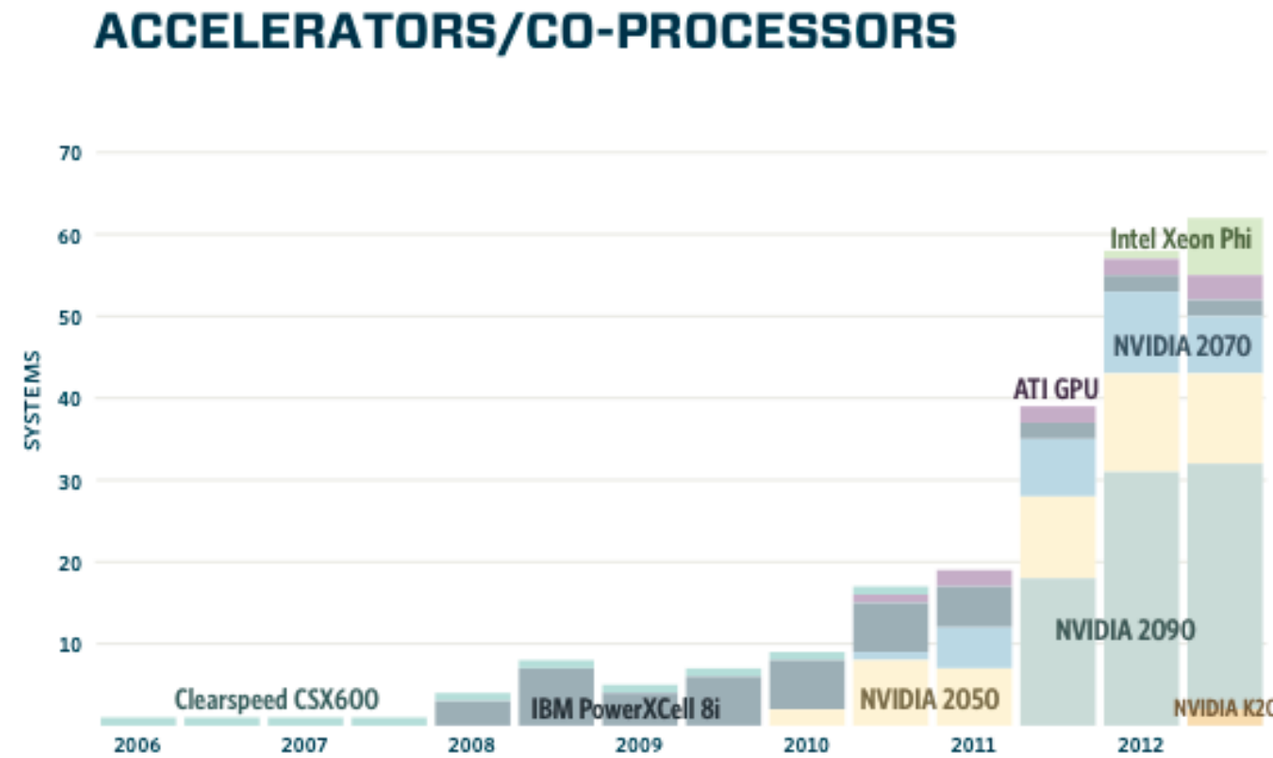
Top 500 - trends





Top 500 - trends





Scalable Algorithms: Motivation/Drivers

- Bridging the Performance Gap while dealing with Hybrid Architectures
- Increased Scalability
- Highly fault-tolerant and fault-resilient algorithms
- Need to calculate with higher precision without restart
- Need to tackle efficiently Grand Challenges problems

Challenges

To achieve excellent results scalability at all levels would be required:

- Mathematical models level
- Algorithmic level
- Systems level

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Parallel Scalable Algorithms- Parallelization Techniques

Vassil Alexandrov, *ICREA* - **Barcelona Supercomputing Center, Spain**

Parallel Algorithms and Concurrency

- Parallel Algorithms
  - Tasks and Decomposition
  - Processes and Mapping
  - Processes Versus Processors
- Decomposition Techniques
  - Recursive Decomposition
  - Recursive Decomposition
  - Exploratory Decomposition
  - Hybrid Decomposition
- Characteristics of Tasks and Interactions
  - Task Generation, Granularity, and Context
  - Characteristics of Task Interactions.

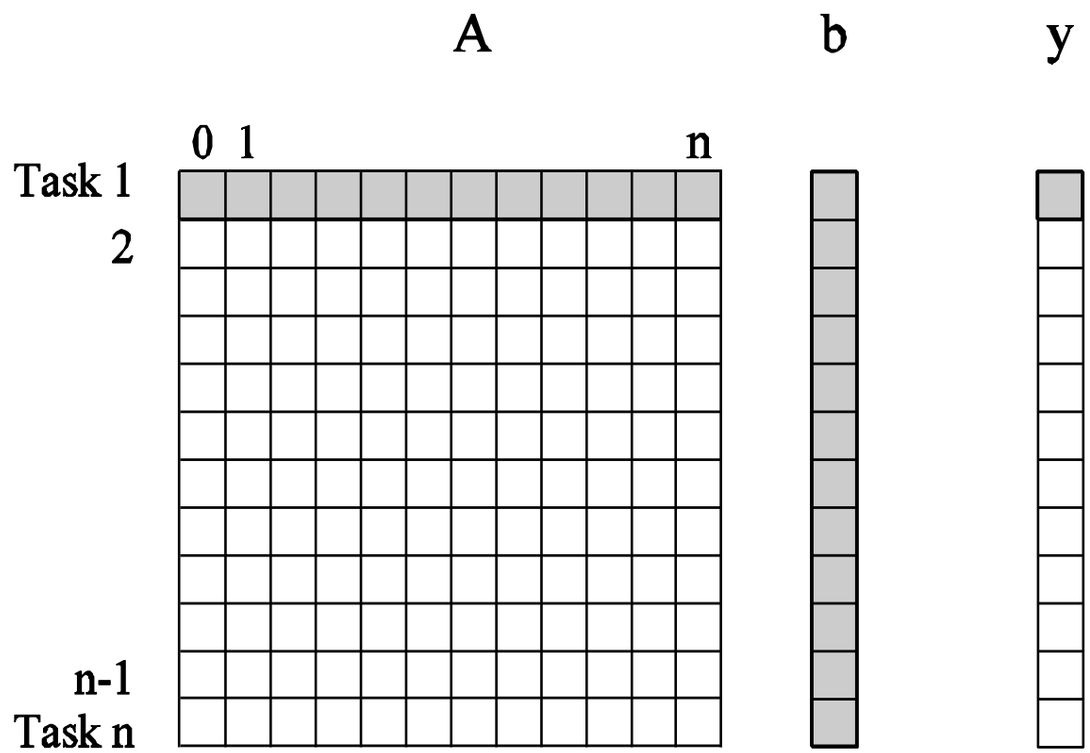
Mapping

- Mapping Techniques for Load Balancing
- Methods for Minimizing Interaction Overheads
- Parallel Algorithm Design Models

Decomposition, Tasks, and Dependency Graphs

- The first step in developing a parallel algorithm is to decompose the problem into tasks that can be executed concurrently
- A given problem may be decomposed into tasks in many different ways.
- Tasks may be of same, or different sizes.
- A decomposition can be illustrated in the form of a directed graph with nodes corresponding to tasks and edges indicating that the result of one task is required for processing the next. Such a graph is called a *task dependency graph*.

Multiplying a Dense Matrix with a Vector

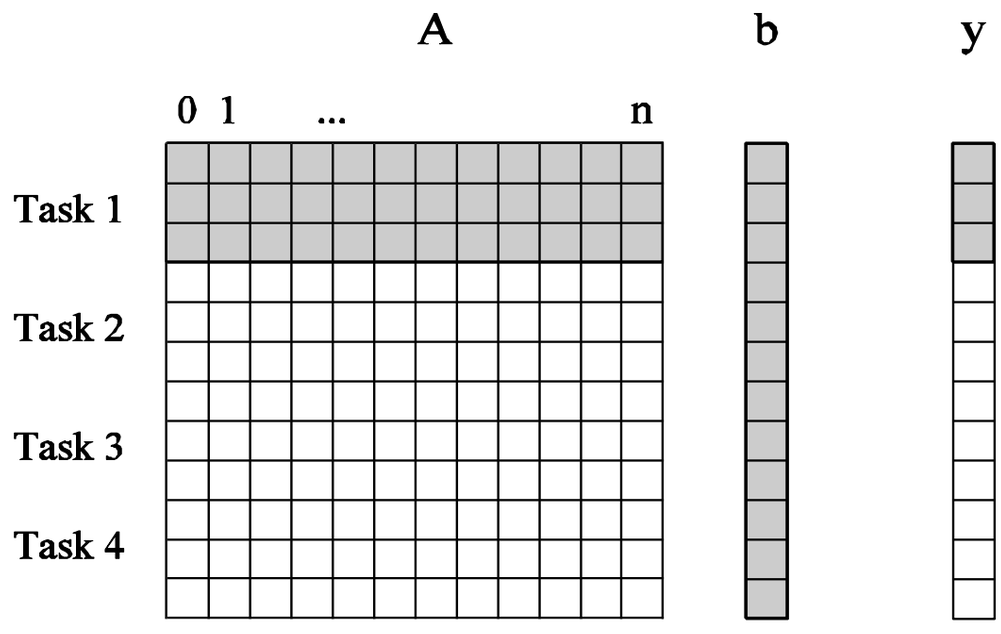


Computation of each element of output vector ***y*** is independent of other elements. Based on this, a dense matrix-vector product can be decomposed into ***n*** tasks. The figure highlights the portion of the matrix and vector accessed by Task 1.

Observations: While tasks share data (namely, the vector ***b***), they do not have any control dependencies - i.e., no task needs to wait for the (partial) completion of any other. All tasks are of the same size in terms of number of operations. *Is this the maximum number of tasks we could decompose this problem into?*

Granularity of Task Decompositions

- The number of tasks into which a problem is decomposed determines its granularity.
- Decomposition into a large number of tasks results in fine-grained decomposition and that into a small number of tasks results in a coarse grained decomposition.



A coarse grained version of the dense matrix-vector product example. Each task in this example corresponds to the computation of p=3 elements of the result vector.

Degree of Concurrency

- The number of tasks that can be executed in parallel is the *degree of concurrency* of a decomposition.
- Since the number of tasks that can be executed in parallel may change over program execution, the *maximum degree of concurrency* is the maximum number of such tasks at any point during execution. *What is the maximum degree of concurrency of the database query examples?*
- The *average degree of concurrency* is the average number of tasks that can be processed in parallel over the execution of the program. *Assuming that each tasks in the database example takes identical processing time, what is the average degree of concurrency in each decomposition?*
- The degree of concurrency increases as the decomposition becomes finer in granularity and vice versa.

Limits on Parallel Performance

- It would appear that the parallel time can be made arbitrarily small by making the decomposition finer in granularity.
- There is an inherent bound on how fine the granularity of a computation can be. *For example, in the case of multiplying a dense matrix with a vector, there can be no more than* (n^2) *concurrent tasks.*
- Concurrent tasks may also have to exchange data with other tasks. This results in communication overhead. The tradeoff between the granularity of a decomposition and associated overheads often determines performance bounds.

Processes and Mapping

- In general, the number of tasks in a decomposition exceeds the number of processing elements available.

- For this reason, a parallel algorithm must also provide a mapping of tasks to processes.

Processes and Mapping

- Appropriate mapping of tasks to processes is critical to the parallel performance of an algorithm.
- Mappings are determined by both the task dependency and task interaction graphs.
- Task dependency graphs can be used to ensure that work is equally spread across all processes at any point (minimum idling and optimal load balance).
- Task interaction graphs can be used to make sure that processes need minimum interaction with other processes (minimum communication).

Processes and Mapping

An appropriate mapping must minimize parallel execution time by:

- Mapping independent tasks to different processes.
- Assigning tasks on critical path to processes as soon as they become available.
- Minimizing interaction between processes by mapping tasks with dense interactions to the same process.

  Note: These criteria often conflict with each other. For example, a decomposition into one task (or no decomposition at all) minimizes interaction but does not result in a speedup at all!

Decomposition Techniques

So how does one decompose a task into various subtasks?

While there is no single recipe that works for all problems, we present a set of commonly used techniques that apply to broad classes of problems. These include:

- recursive decomposition
- data decomposition
- exploratory decomposition
- hybrid decomposition

Data Decomposition: Example

Consider the problem of multiplying two ***n*** x ***n*** matrices ***A*** and ***B*** to yield matrix ***C***. The output matrix ***C*** can be partitioned into four tasks as follows:

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

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

Data Decomposition: Example

A partitioning of output data does not result in a unique decomposition into tasks. For example, for the same problem as in previous slide, with identical output data distribution, we can derive the following two (different) decompositions:

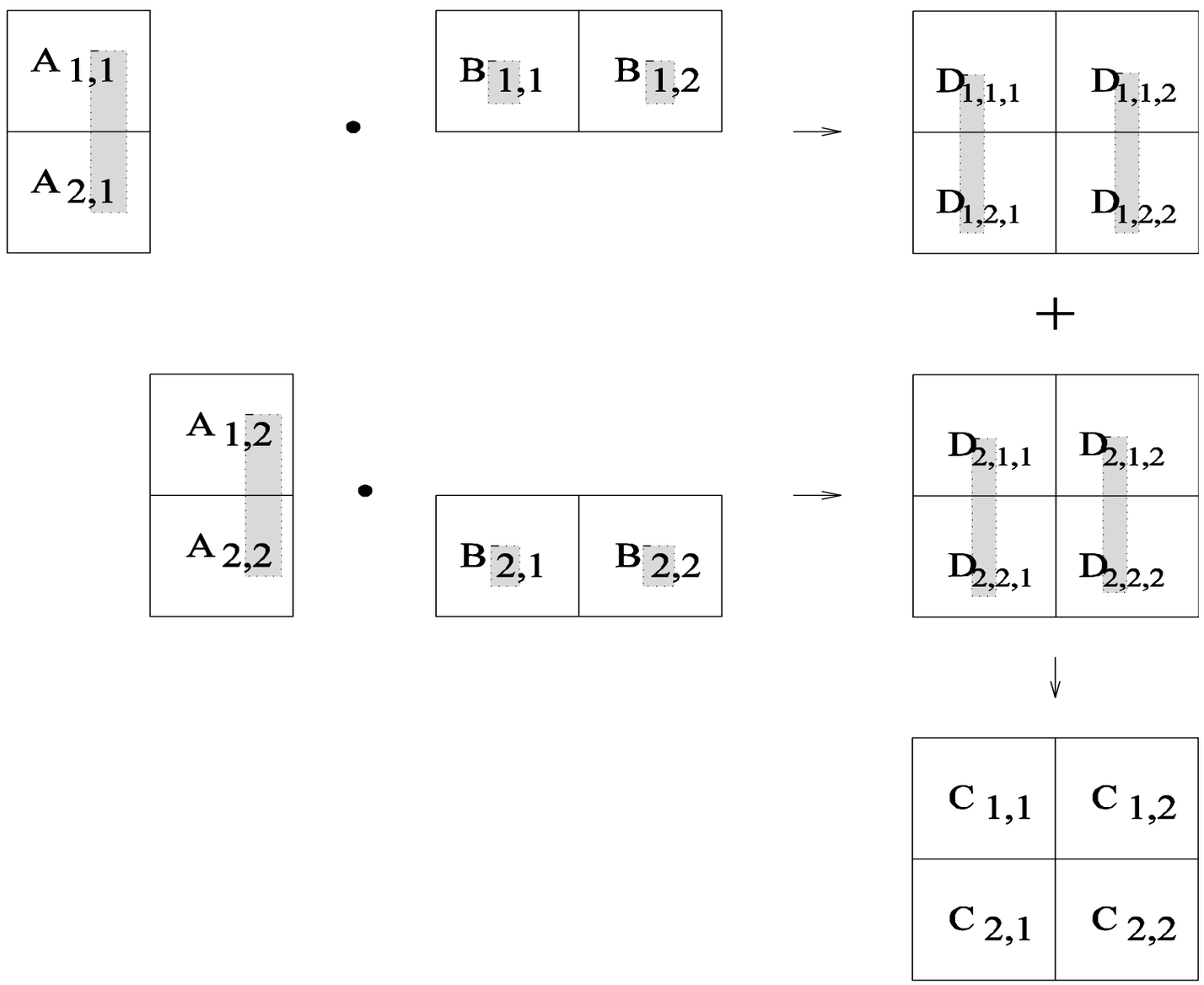
Decomposition I	**Decomposition II**
Task 1: $\boldsymbol{C}_{1,1} = \boldsymbol{A}_{1,1}\,\boldsymbol{B}_{1,1}$	Task 1: $\boldsymbol{C}_{1,1} = \boldsymbol{A}_{1,1}\,\boldsymbol{B}_{1,1}$
Task 2: $\boldsymbol{C}_{1,1} = \boldsymbol{C}_{1,1} + \boldsymbol{A}_{1,2}\,\boldsymbol{B}_{2,1}$	Task 2: $\boldsymbol{C}_{1,1} = \boldsymbol{C}_{1,1} + \boldsymbol{A}_{1,2}\,\boldsymbol{B}_{2,1}$
Task 3: $\boldsymbol{C}_{1,2} = \boldsymbol{A}_{1,1}\,\boldsymbol{B}_{1,2}$	Task 3: $\boldsymbol{C}_{1,2} = \boldsymbol{A}_{1,2}\,\boldsymbol{B}_{2,2}$
Task 4: $\boldsymbol{C}_{1,2} = \boldsymbol{C}_{1,2} + \boldsymbol{A}_{1,2}\,\boldsymbol{B}_{2,2}$	Task 4: $\boldsymbol{C}_{1,2} = \boldsymbol{C}_{1,2} + \boldsymbol{A}_{1,1}\,\boldsymbol{B}_{1,2}$
Task 5: $\boldsymbol{C}_{2,1} = \boldsymbol{A}_{2,1}\,\boldsymbol{B}_{1,1}$	Task 5: $\boldsymbol{C}_{2,1} = \boldsymbol{A}_{2,2}\,\boldsymbol{B}_{2,1}$
Task 6: $\boldsymbol{C}_{2,1} = \boldsymbol{C}_{2,1} + \boldsymbol{A}_{2,2}\,\boldsymbol{B}_{2,1}$	Task 6: $\boldsymbol{C}_{2,1} = \boldsymbol{C}_{2,1} + \boldsymbol{A}_{2,1}\,\boldsymbol{B}_{1,1}$
Task 7: $\boldsymbol{C}_{2,2} = \boldsymbol{A}_{2,1}\,\boldsymbol{B}_{1,2}$	Task 7: $\boldsymbol{C}_{2,2} = \boldsymbol{A}_{2,1}\,\boldsymbol{B}_{1,2}$
Task 8: $\boldsymbol{C}_{2,2} = \boldsymbol{C}_{2,2} + \boldsymbol{A}_{2,2}\,\boldsymbol{B}_{2,2}$	Task 8: $\boldsymbol{C}_{2,2} = \boldsymbol{C}_{2,2} + \boldsymbol{A}_{2,2}\,\boldsymbol{B}_{2,2}$

Intermediate Data Partitioning

- Computation can often be viewed as a sequence of transformation from the input to the output data.
- In these cases, it is often beneficial to use one of the intermediate stages as a basis for decomposition.

Intermediate Data Partitioning: Example

Let us revisit the example of dense matrix multiplication. We first show how we can visualize this computation in terms of intermediate matrices ***D***.



Intermediate Data Partitioning

A decomposition of intermediate data structure leads to the following decomposition into 8 + 4 tasks:

Stage I

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

Stage II

$$\begin{pmatrix} D_{1,1,1} & D_{1,1,2} \\ D_{1,2,2} & D_{1,2,2} \end{pmatrix} + \begin{pmatrix} D_{2,1,1} & D_{2,1,2} \\ D_{2,2,2} & D_{2,2,2} \end{pmatrix} \rightarrow \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}$$

Task 01: $\boldsymbol{D}_{1,1,1} = \boldsymbol{A}_{1,1}\,\boldsymbol{B}_{1,1}$	Task 02: $\boldsymbol{D}_{2,1,1} = \boldsymbol{A}_{1,2}\,\boldsymbol{B}_{2,1}$
Task 03: $\boldsymbol{D}_{1,1,2} = \boldsymbol{A}_{1,1}\,\boldsymbol{B}_{1,2}$	Task 04: $\boldsymbol{D}_{2,1,2} = \boldsymbol{A}_{1,2}\,\boldsymbol{B}_{2,2}$
Task 05: $\boldsymbol{D}_{1,2,1} = \boldsymbol{A}_{2,1}\,\boldsymbol{B}_{1,1}$	Task 06: $\boldsymbol{D}_{2,2,1} = \boldsymbol{A}_{2,2}\,\boldsymbol{B}_{2,1}$
Task 07: $\boldsymbol{D}_{1,2,2} = \boldsymbol{A}_{2,1}\,\boldsymbol{B}_{1,2}$	Task 08: $\boldsymbol{D}_{2,2,2} = \boldsymbol{A}_{2,2}\,\boldsymbol{B}_{2,2}$
Task 09: $\boldsymbol{C}_{1,1} = \boldsymbol{D}_{1,1,1} + \boldsymbol{D}_{2,1,1}$	Task 10: $\boldsymbol{C}_{1,2} = \boldsymbol{D}_{1,1,2} + \boldsymbol{D}_{2,1,2}$
Task 11: $\boldsymbol{C}_{2,1} = \boldsymbol{D}_{1,2,1} + \boldsymbol{D}_{2,2,1}$	Task 12: $\boldsymbol{C}_{2,,2} = \boldsymbol{D}_{1,2,2} + \boldsymbol{D}_{2,2,2}$

Exploratory Decomposition

- In many cases, the decomposition of the problem goes hand-in-hand with its execution.
- These problems typically involve the exploration (search) of a state space of solutions.
- Problems in this class include a variety of discrete optimization problems (0/1 integer programmin, etc.), theorem proving, game playing, etc.

Speculative Decomposition

- In some applications, dependencies between tasks are not known a-priori.
- For such applications, it is impossible to identify independent tasks.
- There are generally two approaches to dealing with such applications: conservative approaches, which identify independent tasks only when they are guaranteed to not have dependencies, and, optimistic approaches, which schedule tasks even when they may potentially be erroneous.
- Conservative approaches may yield little concurrency and optimistic approaches may require roll-back mechanism in the case of an error.

Characteristics of Tasks

Once a problem has been decomposed into independent tasks, the characteristics of these tasks critically impact choice and performance of parallel algorithms. Relevant task characteristics include:

- Task generation.
- Task sizes.
- Size of data associated with tasks.

Task Generation

- Static task generation: Concurrent tasks can be identified a-priori (matrix operations).
- Dynamic task generation (generated during computation)

Task Sizes

- Task sizes may be uniform (i.e., all tasks are the same size) or non-uniform.
- Non-uniform task sizes may be such that they can be determined (or estimated) a-priori or not.
- Examples in this class include discrete optimization problems, in which it is difficult to estimate the effective size of a state space.

Size of Data Associated with Tasks

- The size of data associated with a task may be small or large when viewed in the context of the size of the task.
- A small context of a task implies that an algorithm can easily communicate this task to other processes dynamically.
- A large context ties the task to a process, or alternately, an algorithm may attempt to reconstruct the context at another processes as opposed to communicating the context of the task (e.g., 0/1 integer programming).

Characteristics of Task Interactions

- Tasks may communicate with each other in various ways. The associated dichotomy is:
- Static interactions: The tasks and their interactions are known a-priori. These are relatively simpler to code into programs.
- Dynamic interactions: The timing or interacting tasks cannot be determined a-priori. These interactions are harder to code, especially, as we shall see, using message passing APIs.

Characteristics of Task Interactions

- Regular interactions: There is a definite pattern (in the graph sense) to the interactions. These patterns can be exploited for efficient implementation.
- Irregular interactions: Interactions lack well-defined topologies.

Characteristics of Task Interactions

- Interactions may be read-only or read-write.
- In read-only interactions, tasks just read data items associated with other tasks.
- In read-write interactions tasks read, as well as modify data items associated with other tasks.
- In general, read-write interactions are harder to code, since they require additional synchronization primitives.

Characteristics of Task Interactions

- Interactions may be one-way or two-way.
- A one-way interaction can be initiated and accomplished by one of the two interacting tasks.
- A two-way interaction requires participation from both tasks involved in an interaction.
- One way interactions are somewhat harder to code in message passing APIs.

Mapping Techniques

- Once a problem has been decomposed into concurrent tasks, these must be mapped to processes (that can be executed on a parallel platform).
- Mappings must minimize overheads.
- Primary overheads are communication and idling.
- Minimizing these overheads often represents contradicting objectives.
- Assigning all work to one processor trivially minimizes communication at the expense of significant idling.

Mapping Techniques for Minimum Idling

Mapping techniques can be static or dynamic.

- Static Mapping: Tasks are mapped to processes a-priori. For this to work, we must have a good estimate of the size of each task. Even in these cases, the problem may be NP complete.
- Dynamic Mapping: Tasks are mapped to processes at runtime. This may be because the tasks are generated at runtime, or that their sizes are not known.

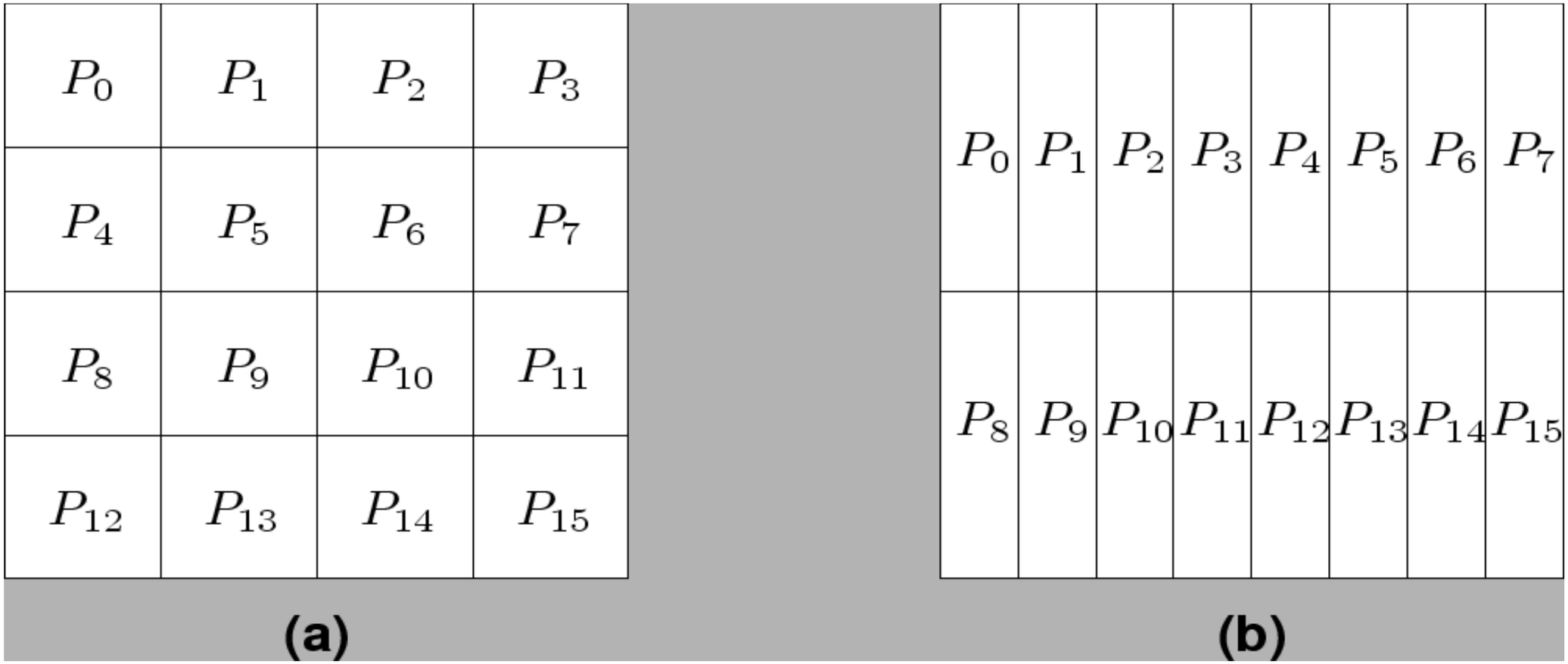
Other factors that determine the choice of techniques include the size of data associated with a task and the nature of underlying domain.

Schemes for Static Mapping

- Mappings based on data partitioning.
- Mappings based on task graph partitioning – functional decomposition
- Hybrid mappings.

Block Array Distribution Schemes

Block distribution schemes can be generalized to higher dimensions as well.

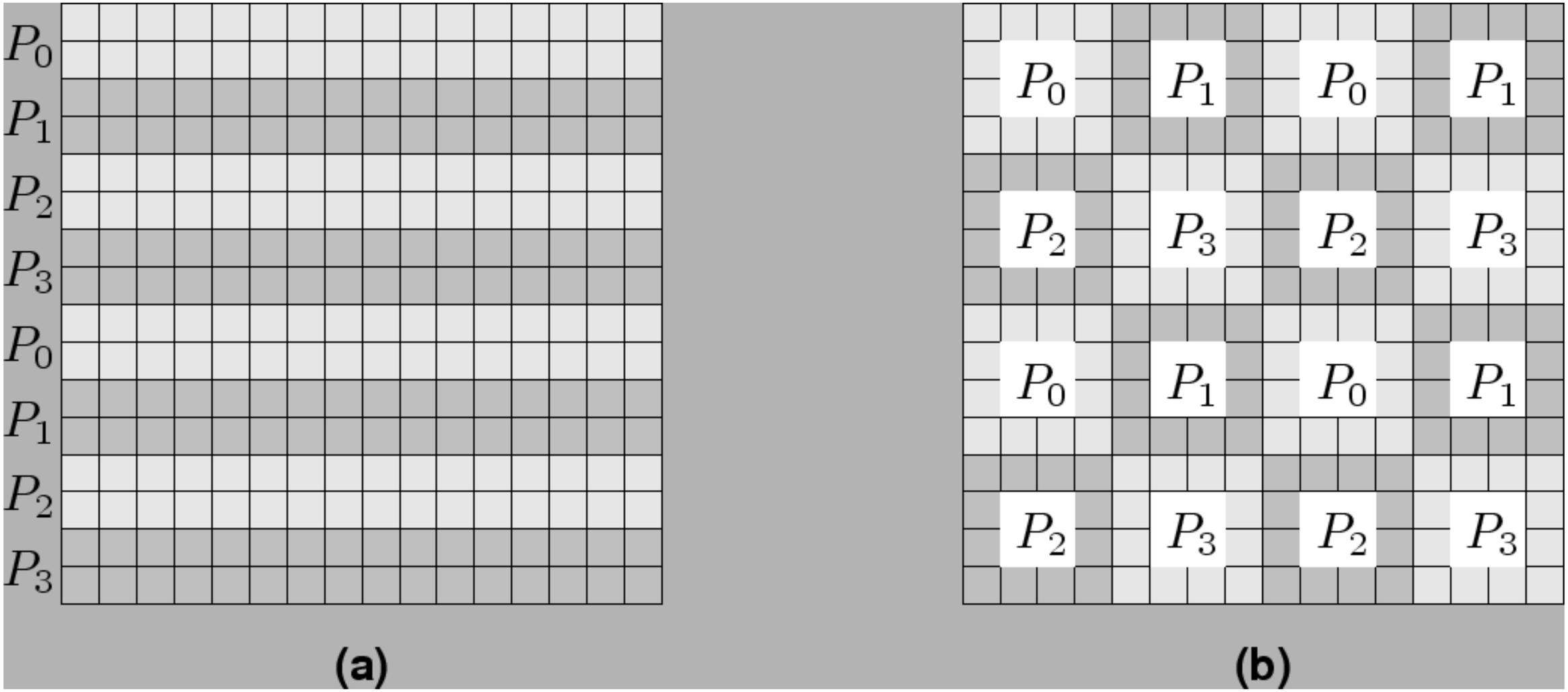


Cyclic and Block Cyclic Distributions

- If the amount of computation associated with data items varies, a block decomposition may lead to significant load imbalances.
- A simple example of this is in LU decomposition (or Gaussian Elimination) of dense matrices.

Block-Cyclic Distribution

- A cyclic distribution is a special case in which block size is one.
- A block distribution is a special case in which block size is ***n/p*** , where ***n*** is the dimension of the matrix and ***p*** is the number of processes.



Mappings Based on Task Partitioning

- Partitioning a given task-dependency graph across processes.
- Determining an optimal mapping for a general task-dependency graph is an NP-complete problem.
- Excellent heuristics exist for structured graphs.

Hierarchical Mappings

- Sometimes a single mapping technique is inadequate.
- For example, the task mapping of the binary tree (quicksort) cannot use a large number of processors.
- For this reason, task mapping can be used at the top level and data partitioning within each level.

Minimizing Interaction Overheads

- Maximize data locality: Where possible, reuse intermediate data. Restructure computation so that data can be reused in smaller time windows.
- Minimize volume of data exchange: There is a cost associated with each word that is communicated. For this reason, we must minimize the volume of data communicated.
- Minimize frequency of interactions: There is a startup cost associated with each interaction. Therefore, try to merge multiple interactions to one, where possible.
- Minimize contention and hot-spots: Use decentralized techniques, replicate data where necessary.

Minimizing Interaction Overheads (continued)

- Overlapping computations with interactions: Use non-blocking communications, multithreading, and prefetching to hide latencies.
- Replicating data or computations.
- Using group communications instead of point-to-point primitives.
- Overlap interactions with other interactions.

Parallel Algorithm Models

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.

- Data Parallel Model (Data Decomposition): Tasks are statically (or semi-statically) mapped to processes and each task performs similar operations on different data.
- Task Graph Model (Functional Decomposition): Starting from a task dependency graph, the interrelationships among the tasks are utilized to promote locality or to reduce interaction costs.

Parallel Algorithm Models (cont.)

- SPMD – Single Program Multiple Data model
- MPMD – Multiple Programs Multiple Data model
- Master-Slave Model: One or more processes generate work and allocate it to worker processes. This allocation may be static or dynamic.
- Pipeline / Producer-Consumer Model: A stream of data is passed through a succession of processes, each of which perform some task on it.
- Hybrid Models: A hybrid model may be composed either of multiple models applied hierarchically or multiple models applied sequentially to different phases of a parallel algorithm.

Parallel Algorithms Design

- Start from an existing sequential algorithm and design new parallel one.
- Start from existing parallel algorithms and improve it.
- Design completely new parallel algorithm.

Parallel Algorithms Design - Summary

- Choose the initial decomposition technique depending on the given problem.
- Replicate the data to minimize the communication if necessary
- Define initial task sizes.
- Map initial parallel algorithm onto parallel architecture.
- Calibrate the algorithm by optimizing the task size and minimizing the communication.
- Arrive iteratively into the refined parallel algorithm.

Remember, the parallel algorithm has to be much faster than the sequential one!