Introduction to Parallel and High-Performance Computing

(with Machine-Learning applications)

Anshul Gupta, IBM Research

Prabhanjan (Anju) Kambadur, Bloomberg L.P.

Introduction to Parallel and High-Performance Computing (with Machine-Learning applications)

Part 1

Parallel computing basics and parallel algorithm analysis

Part 2

Parallel algorithms for Building a Classifier

Part 1: Parallel and High-Performance Computing

- Why parallel computing?

Part 1: Parallel and High-Performance Computing

- Why parallel computing?
- Parallel computing platforms

Part 1: Parallel and High-Performance Computing

- Why parallel computing?
- Parallel computing platforms
- Parallel algorithm basics

Part 1: Parallel and High-Performance Computing

- Why parallel computing?
- Parallel computing platforms
- Parallel algorithm basics
- Decomposition for parallelism

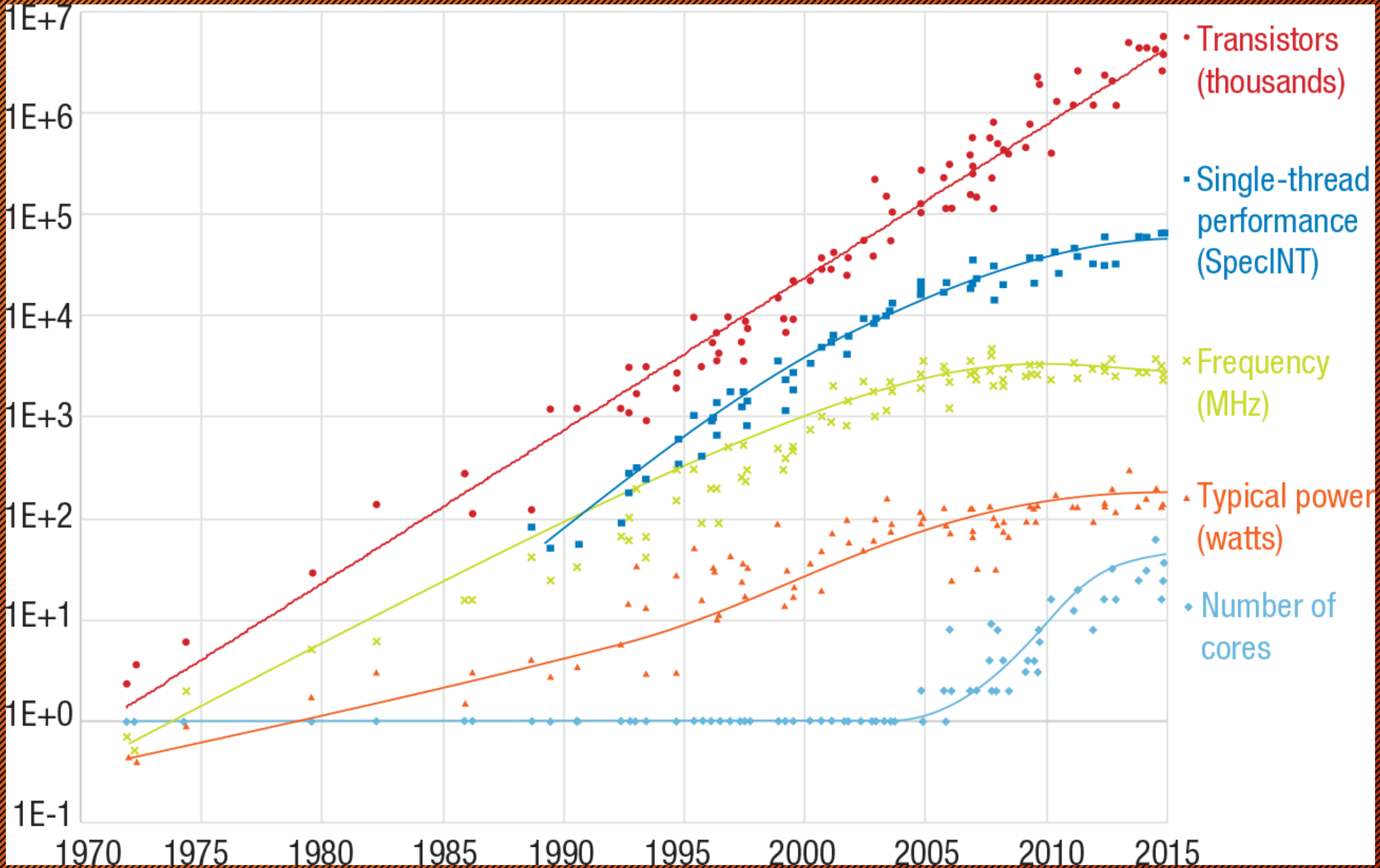
Part 1: Parallel and High-Performance Computing

- Why parallel computing?
- Parallel computing platforms
- Parallel algorithm basics
- Decomposition for parallelism
- Parallel programming models

Part 1: Parallel and High-Performance Computing

- Why parallel computing?
- Parallel computing platforms
- Parallel algorithm basics
- Decomposition for parallelism
- Parallel programming models
- Parallel algorithm analysis

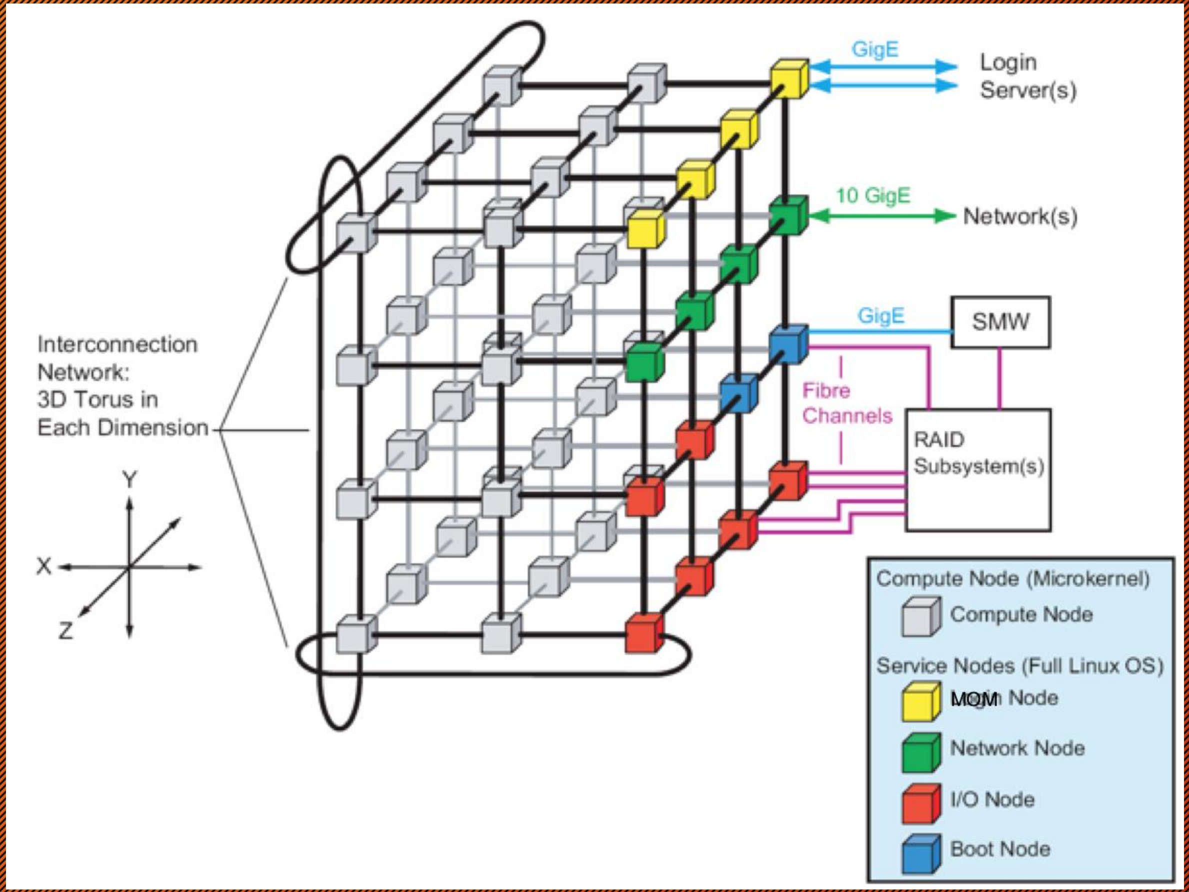
Why parallel computing?



Microprocessor trends: 1972--2015

Kirk M. Bresniker, Sharad Singhal, R. Stanley Williams, "Adapting to Thrive in a New Economy of Memory Abundance",*Computer*, vol.48, no. 12, pp. 44-53, Dec. 2015, doi:10.1109/MC.2015.368

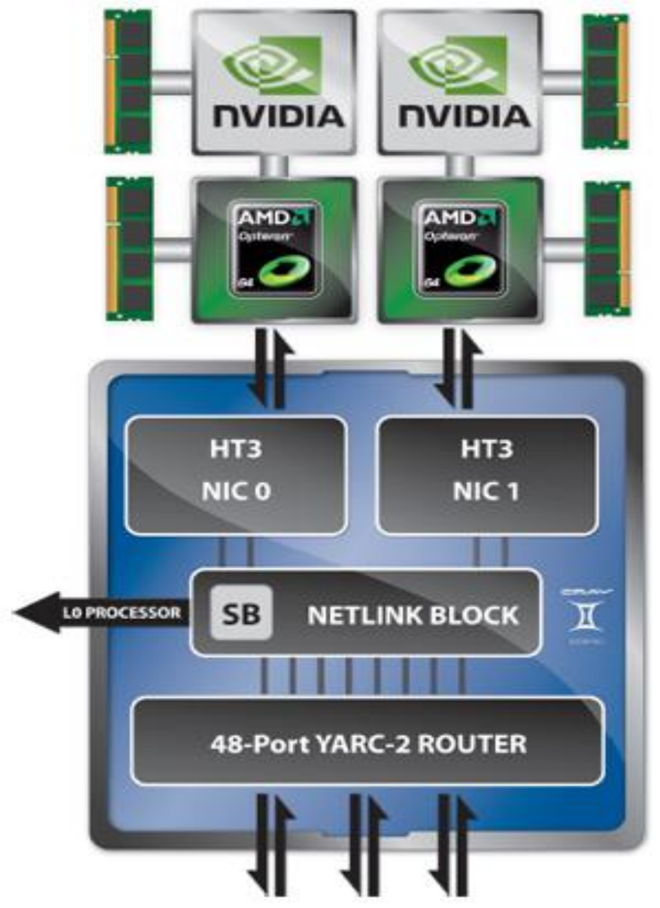
Parallel computing platforms



Highest level of parallelism:

- Compute nodes on an interconnection network
- Possibly, thousands of nodes
- Distributed memory
- Distributed or shared address space
- Scalability analysis crucial

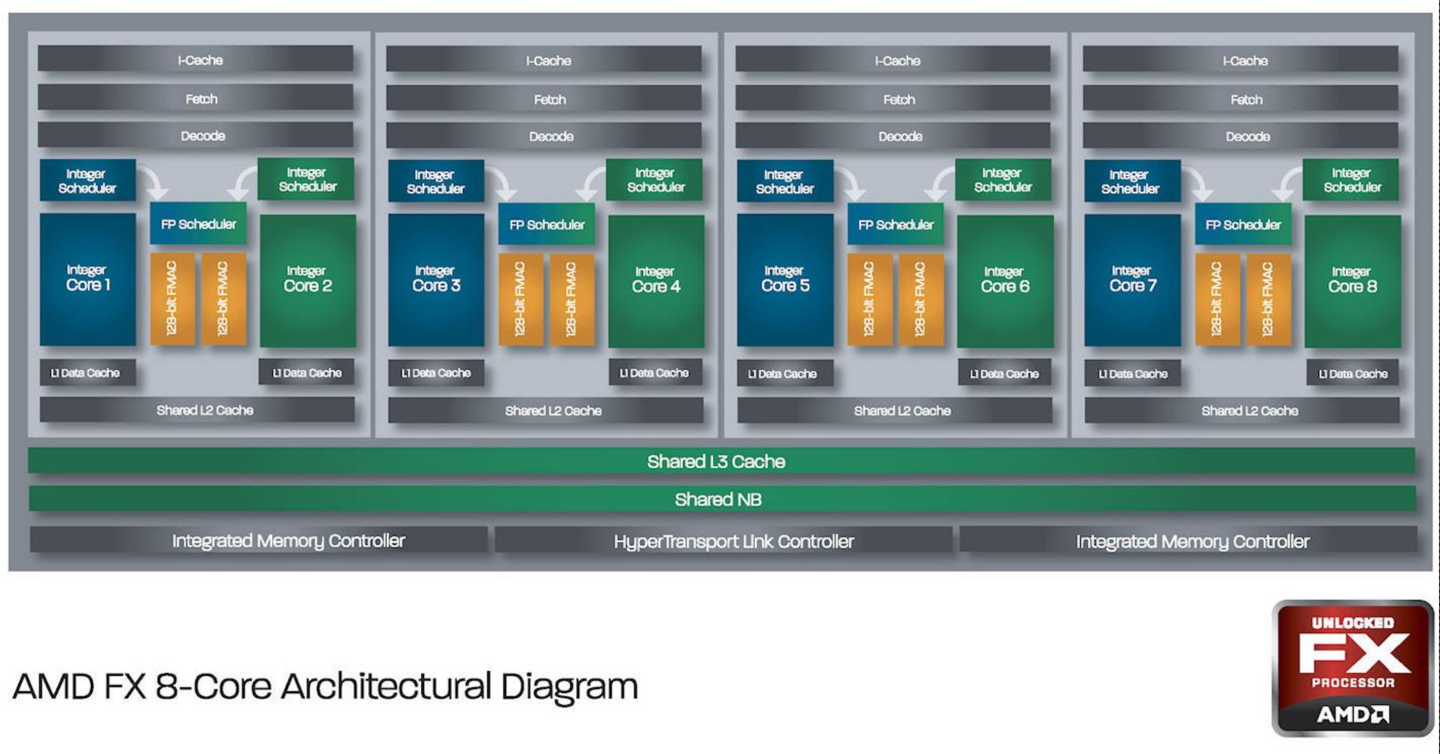
Parallel computing platforms (cont.)



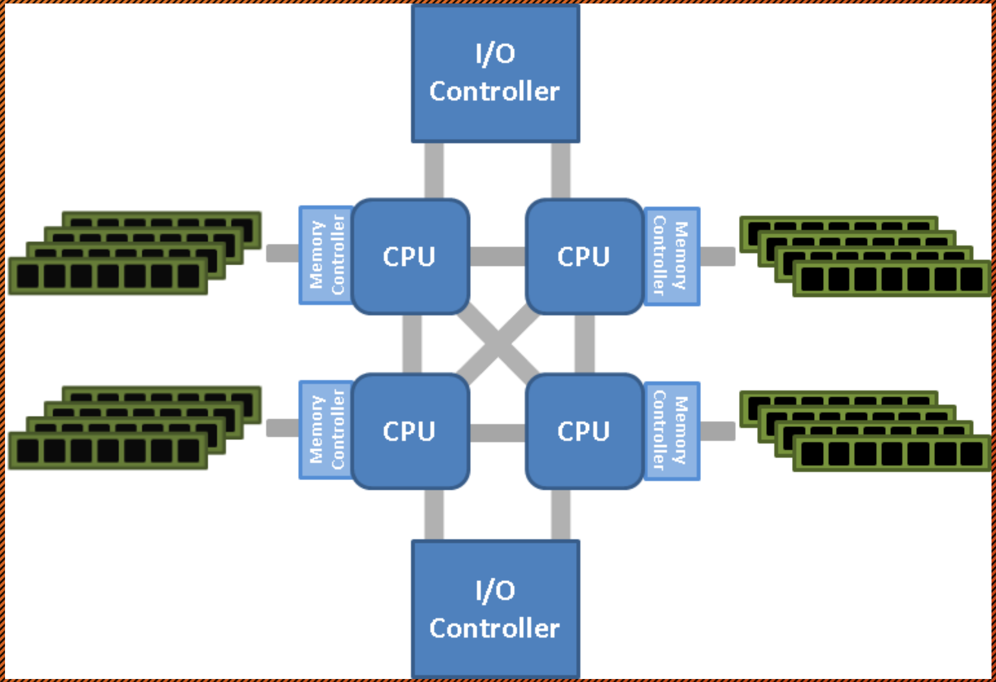
A generalized compute node

- (Possibly) multiple CPUs
- (Possibly) multiple GPUs

Parallel computing platforms (cont.)

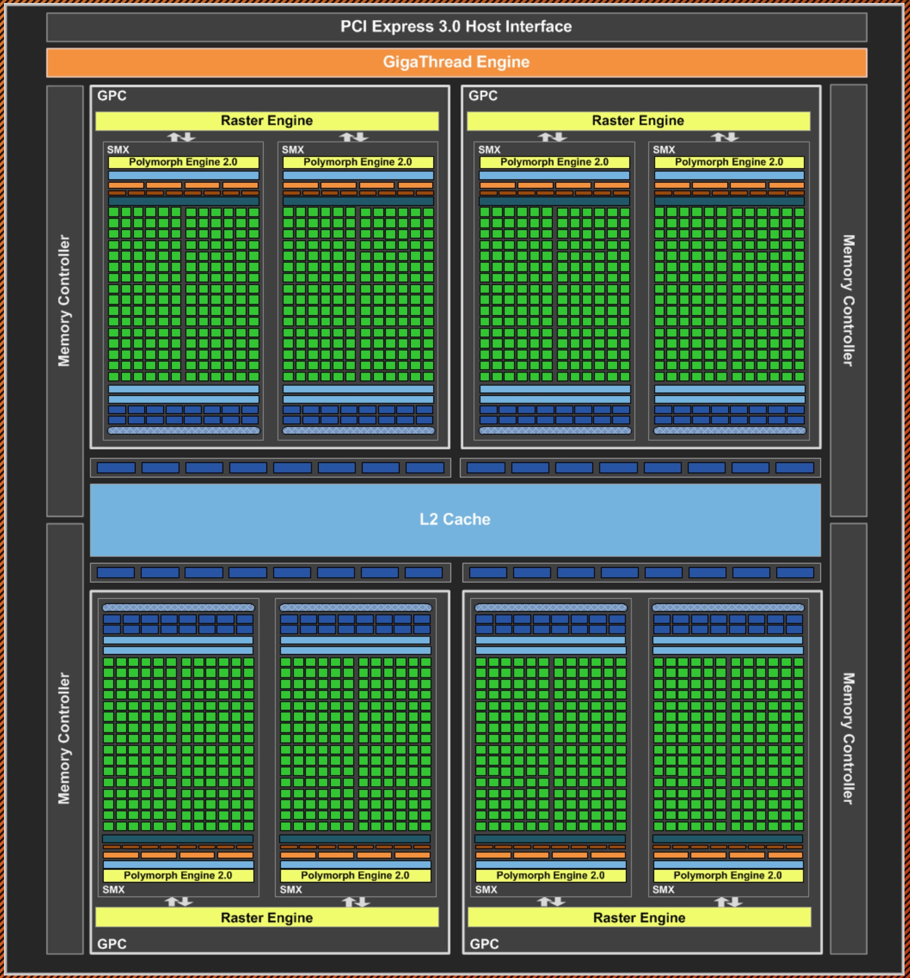


Parallel computing platforms (cont.)



Shared memory on a node, but Non-Uniform Memory Access (NUMA).

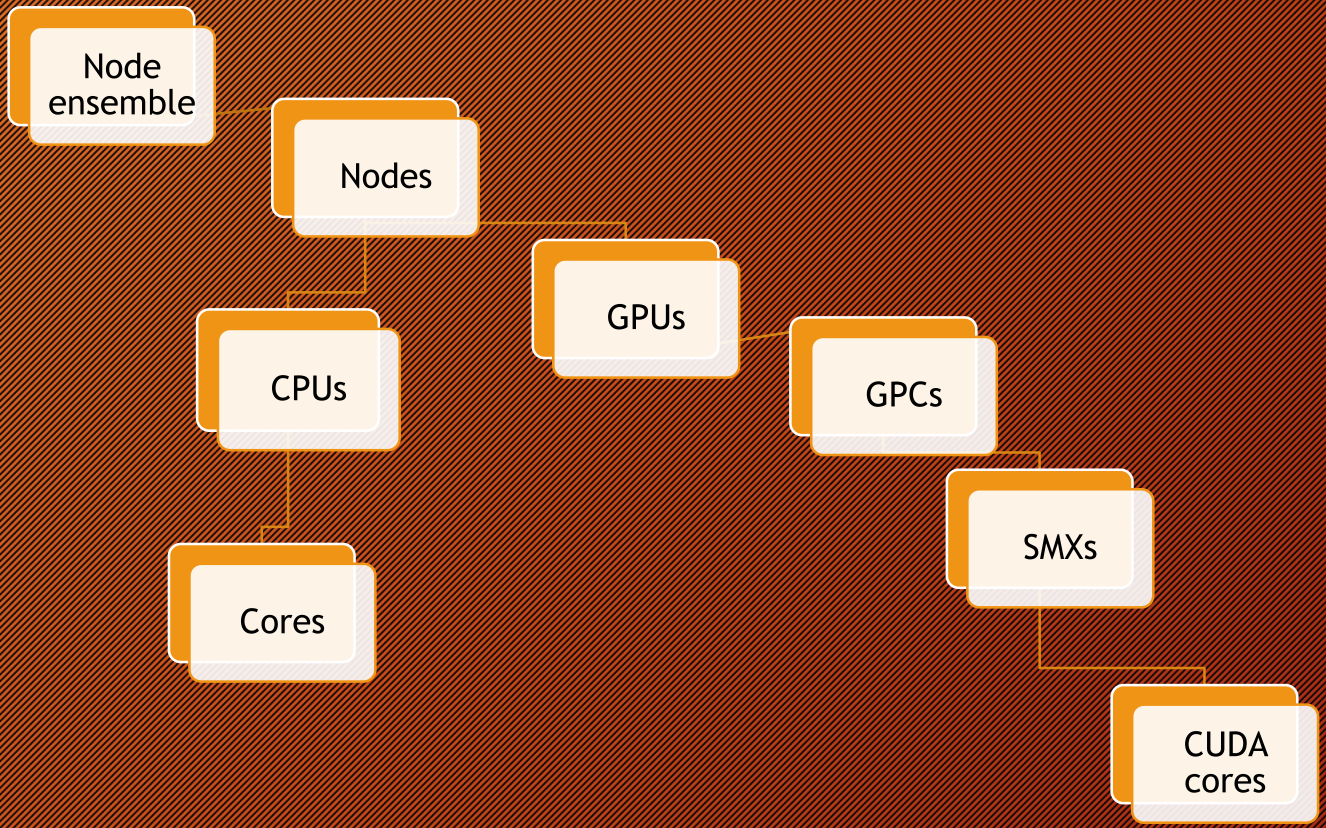
Parallel computing platforms (cont.)



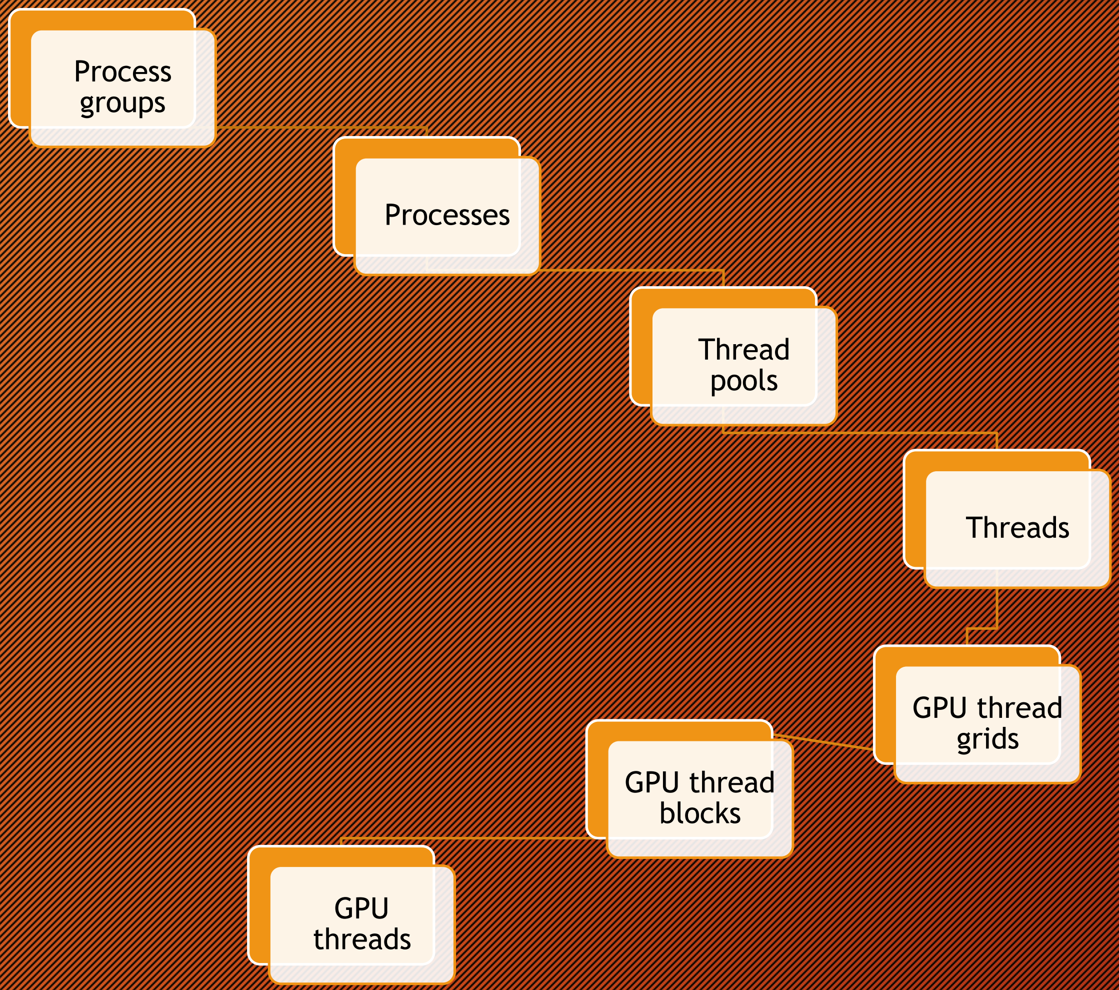
Nvidia GeForce GTX 680 (Kepler)

- 4 GPCs (graphics processing clusters)
- 8 SMXs (streaming multiprocessors)
- 192 X 8 = 1536 CUDA cores

Parallel hardware hierarchy



Parallel program hierarchy



Parallel programming paradigms

<u>Distributed Memory</u> <u>Shared Memory</u> <u>Accelerator</u>

Parallel programming paradigms

<u>Distributed Memory</u>

- Multiple processes
- Distributed address space
- Explicit data movement
- Locality!

Parallel programming paradigms

Distributed Memory

- Multiple processes
- Distributed address space
- Explicit data movement
- Locality!

Shared Memory

- Multiple threads
- Shared address space
- Implicit data movement
- Locality!

Parallel programming paradigms

Distributed Memory

- Multiple processes
- Distributed address space
- Explicit data movement
- Locality!

Shared Memory

- Multiple threads
- Shared address space
- Implicit data movement
- Locality!

Accelerator

- Host memory ← PCIe → Device memory
- Explicit data movement
- Locality!

Algorithm design

- Algorithm design is critical to devising a computing solution

Algorithm design

- Algorithm design is critical to devising a computation solution
- Serial algorithm is a recipe or sequence of basic steps or operations

Algorithm design

- Algorithm design is critical to devising a computation solution
- Serial algorithm is a recipe or sequence of basic steps or operations
- Parallel algorithm is a recipe for solving the given problem using an ensemble of hardware resources

Algorithm design

- Algorithm design is critical to devising a computation solution
- Serial algorithm is a recipe or sequence of basic steps or operations
- Parallel algorithm is a recipe for solving the given problem using an ensemble of hardware resources
- Specifying parallel algorithm involves a lot more than specifying a sequence of basic steps

Algorithm design

- Algorithm design is critical to devising a computation solution
- Serial algorithm is a recipe or sequence of basic steps or operations
- Parallel algorithm is a recipe for solving the given problem using an ensemble of hardware resources
- Specifying parallel algorithm involves a lot more than specifying a sequence of basic steps

Parallel algorithm design steps

- Identifying portions of work that can be performed concurrently

Parallel algorithm design steps

- Identifying portions of work that can be performed concurrently
- Mapping concurrent pieces of work onto computing agents running in parallel

Parallel algorithm design steps

- Identifying portions of work that can be performed concurrently
- Mapping concurrent pieces of work onto computing agents running in parallel
- Making the input, output, and intermediate data available to the right computing agent at the right time

Parallel algorithm design steps

- Identifying portions of work that can be performed concurrently
- Mapping concurrent pieces of work onto computing agents running in parallel
- Making the input, output, and intermediate data available to the right computing agent at the right time
- Managing simultaneous requests for shared data

Parallel algorithm design steps

- Identifying portions of work that can be performed concurrently
- Mapping concurrent pieces of work onto computing agents running in parallel
- Making the input, output, and intermediate data available to the right computing agent at the right time
- Managing simultaneous requests for shared data
- Synchronizing computing agents for correct program execution

Parallel algorithm design steps

- Identifying portions of work that can be performed concurrently - Decomposition
- Mapping concurrent pieces of work onto computing agents running in parallel
- Making the input, output, and intermediate data available to the right computing agent at the right time – Data Dependencies
- Managing simultaneous requests for shared data
- Synchronizing computing agents for correct program execution – Task Dependencies

Decomposition for concurrency

Task Decomposition

Data Decomposition

Decomposition for concurrency

Task Decomposition

- Concurrent tasks are identified and mapped onto to threads or processes

Decomposition for concurrency

Task Decomposition

- Concurrent tasks are identified and mapped onto to threads or processes
- Tasks share or exchange data as needed

Decomposition for concurrency

Task Decomposition

- Concurrent tasks are identified and mapped onto to threads or processes
- Tasks share or exchange data as needed
- May be static or dynamic

Decomposition for concurrency

Task Decomposition

- Concurrent tasks are identified and mapped onto to threads or processes
- Tasks share or exchange data as needed
- May be static or dynamic

Data Decomposition

- Data is partitioned (input, output, or intermediate)

Decomposition for concurrency

Task Decomposition

- Concurrent tasks are identified and mapped onto to threads or processes
- Tasks share or exchange data as needed
- May be static or dynamic

Data Decomposition

- Data is partitioned
- Partitions are assigned to computing agents
- “Owner computes” rule

Decomposition for concurrency

Task Decomposition

- Concurrent tasks are identified and mapped onto to threads or processes
- Tasks share or exchange data as needed
- May be static or dynamic

Data Decomposition

- Data is partitioned
- Partitions are assigned to computing agents
- “Owner computes” rule
- Usually static

Decomposition for concurrency

Data Decomposition + Task Decomposition = Hybrid Decompositions

(Example: sparse matrix factorization)

Task decomposition example

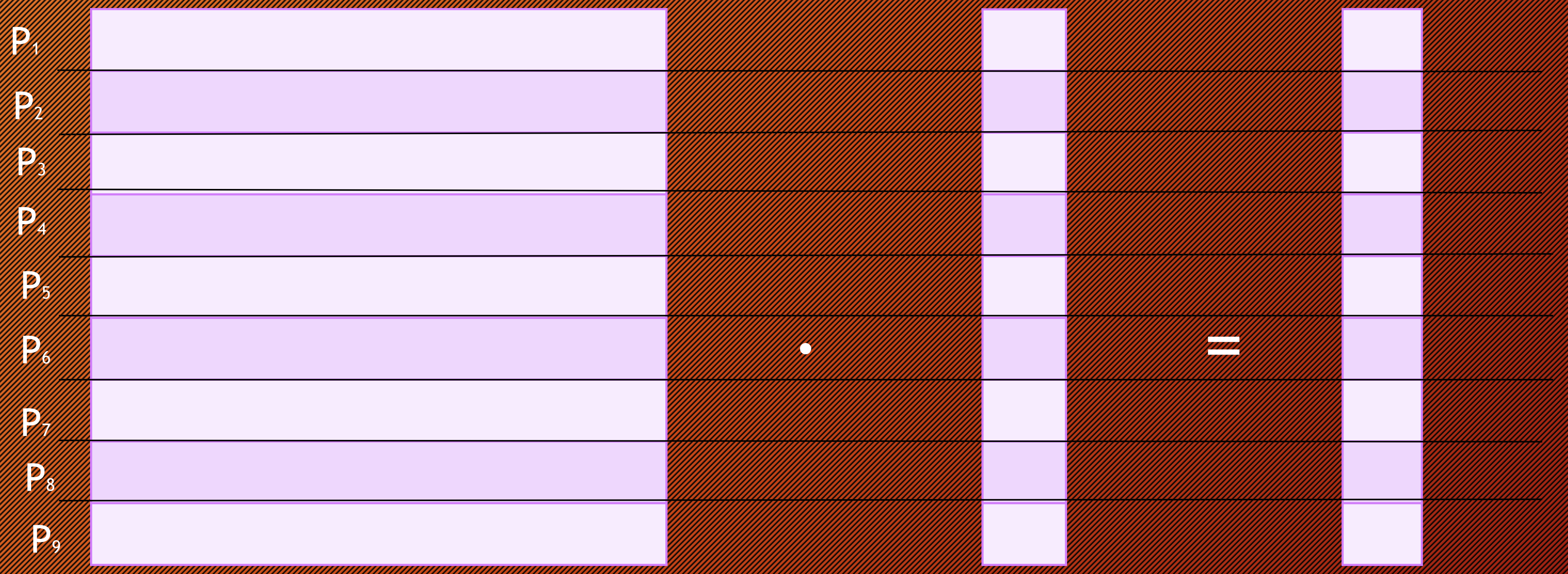
Chess Program

R1 K1 B1 Q . . . P1 P2

- Each task evaluates all moves of a single piece (branch-and-bound)
- Small data (board position) can be replicated
- Dynamic load balancing required

Data decomposition example

Dense Matrix-Vector Multiplication



Parallel application design guidelines

- Focus on one level of hierarchy at a time – from top to bottom.

Parallel application design guidelines

- Focus on one level of hierarchy at a time – from top to bottom.
- Devise the best decomposition strategy at the given level

Parallel application design guidelines

- Focus on one level of hierarchy at a time – from top to bottom.
- Devise the best decomposition strategy at the given level
- Computing agents are likely to be parallel themselves

Parallel application design guidelines

- Focus on one level of hierarchy at a time – from top to bottom.
- Devise the best decomposition strategy at the given level
- Computing agents are likely to be parallel themselves
- Minimize interactions, synchronization, and data movement among computing agents

Parallel application design guidelines

- Focus on one level of hierarchy at a time – from top to bottom.
- Devise the best decomposition strategy at the given level
- Computing agents are likely to be parallel themselves
- Minimize interactions, synchronization, and data movement among computing agents
- Minimize load imbalance and idling among computing agents

Parallel application design guidelines

- Focus on one level of hierarchy at a time – from top to bottom.
- Devise the best decomposition strategy at the given level
- Computing agents are likely to be parallel themselves
- Minimize interactions, synchronization, and data movement among computing agents
- Minimize load imbalance and idling among computing agents

Parallel algorithm analysis

- Serial run time, T_S: time required by best known method on a single computing agent

Parallel algorithm analysis

- Serial run time, T_S: time required by best known method on a single computing agent
- Problem size, W = total amount of work: $T_S = kW$

Parallel algorithm analysis

- Serial run time, T_S: time required by best known method on a single computing agent
- Problem size, W = total amount of work: $T_S = kW$
- Parallel run time, T_P: time elapsed between start of computation until the last of the p computing agents finishes

Parallel algorithm analysis

- Serial run time, T_S: time required by best known method on a single computing agent
- Problem size, W = total amount of work: $T_S = kW$
- Parallel run time, T_P: time elapsed between start of computation until the last of the p computing agents finishes
- Overhead, sum of all wasted compute resources: $T_O = pT_P - T_S$

Parallel algorithm analysis

- Serial run time, T_S: time required by best known method on a single computing agent
- Problem size, W = total amount of work: $T_S = kW$
- Parallel run time, T_P: time elapsed between start of computation until the last of the p computing agents finishes
- Overhead, sum of all wasted compute resources: $T_O = pT_P - T_S$
- Speedup, ratio of serial to parallel time: $S = T_S / T_P = pT_S / (T_S + T_O)$

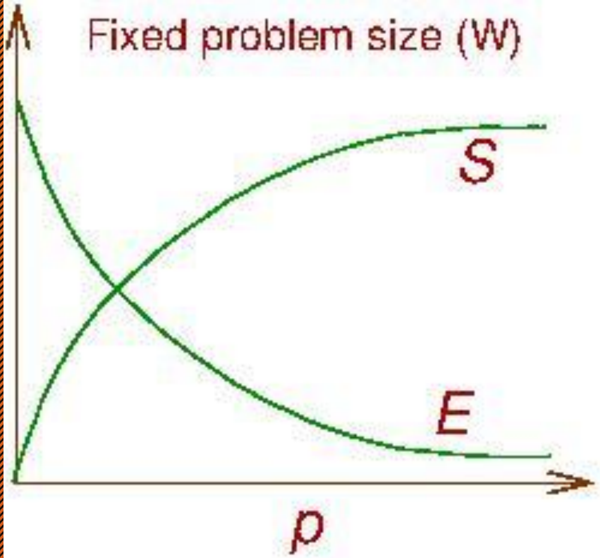
Parallel algorithm analysis

- Serial run time, T_S: time required by best known method on a single computing agent
- Problem size, W = total amount of work: $T_S = kW$
- Parallel run time, T_P: time elapsed between start of computation until the last of the p computing agents finishes
- Overhead, sum of all wasted compute resources: $T_O = pT_P - T_S$
- Speedup, ratio of serial to parallel time: $S = T_S / T_P = pT_S / (T_S + T_O)$
- Efficiency, fraction of overall time spent doing useful work:
  $E = S/p = T_S / pT_P = T_S / (T_S + T_O)$

Parallel algorithm analysis

- Serial run time, T_S: time required by best known method on a single computing agent
- Problem size, W = total amount of work: $T_S = kW$
- Parallel run time, T_P: time elapsed between start of computation until the last of the p computing agents finishes
- Overhead, sum of all wasted compute resources: $T_O = pT_P - T_S$
- Speedup, ratio of serial to parallel time: $S = T_S / T_P = pT_S / (T_S + T_O)$
- Efficiency, fraction of overall time spent doing useful work: $E = S/p = T_S / pT_P = T_S / (T_S + T_O)$

Parallel algorithm analysis



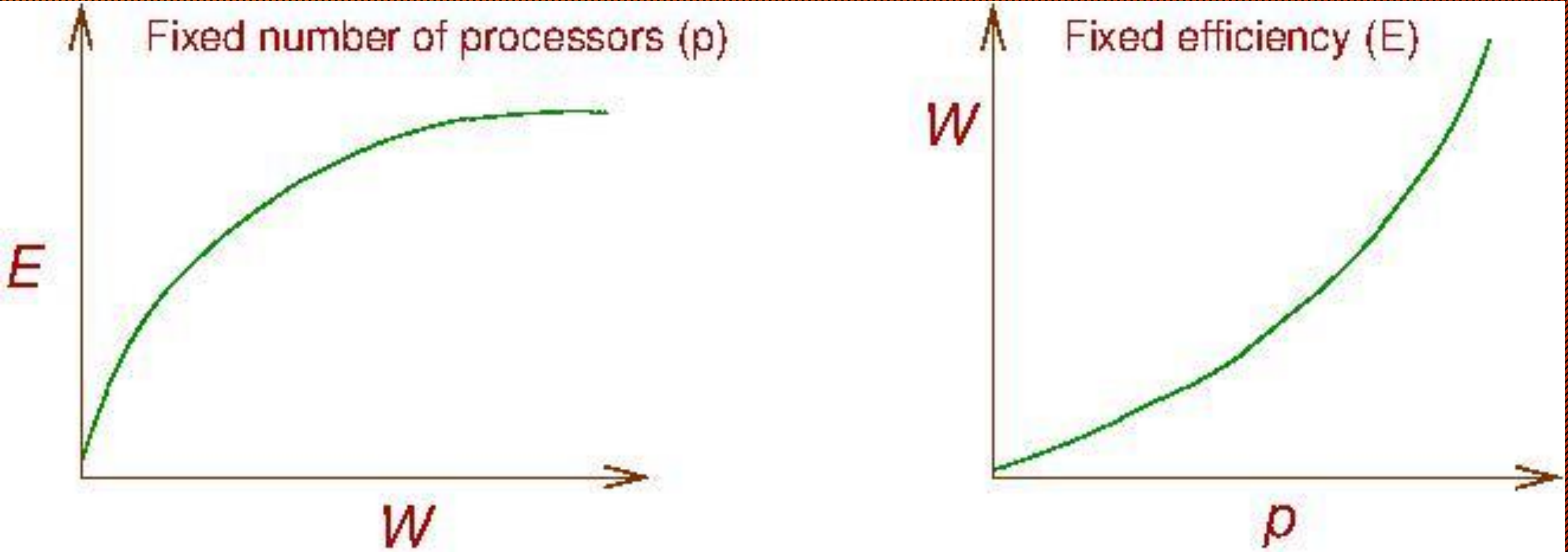
W = problem size (opcount)

p = number of computing agents

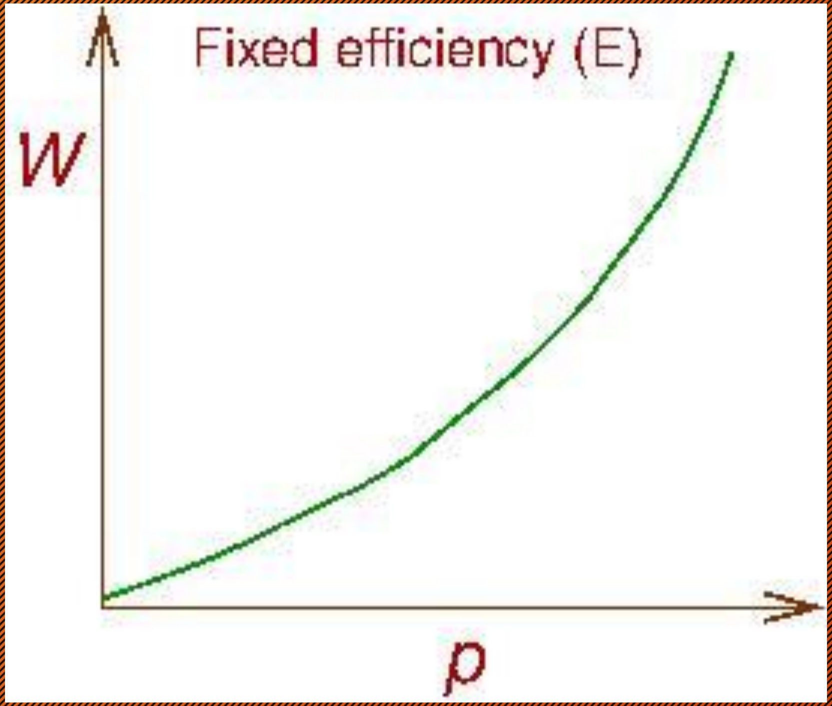
S = speedup

E = efficiency = S/p

Parallel algorithm analysis

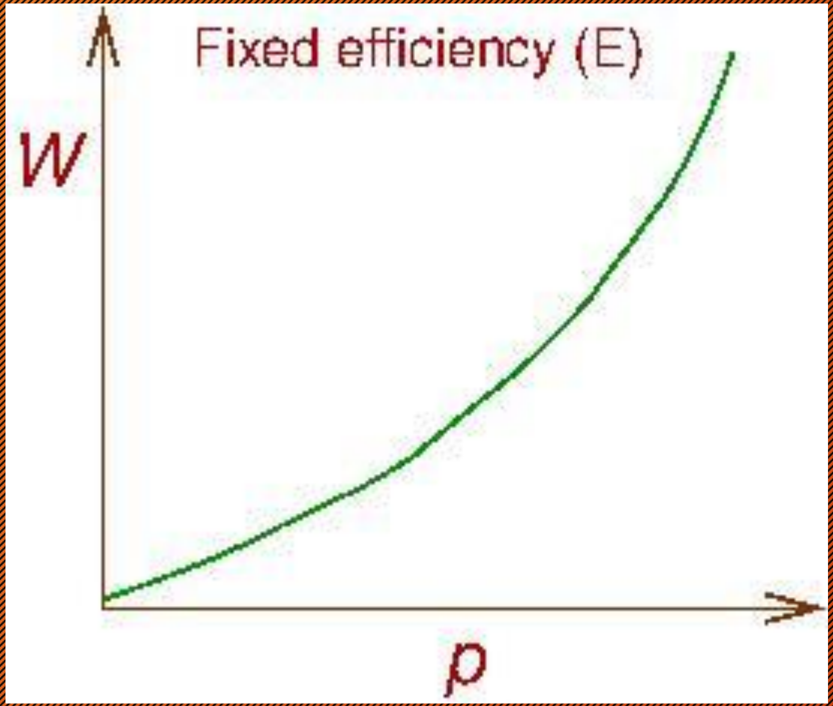


Isoefficiency function



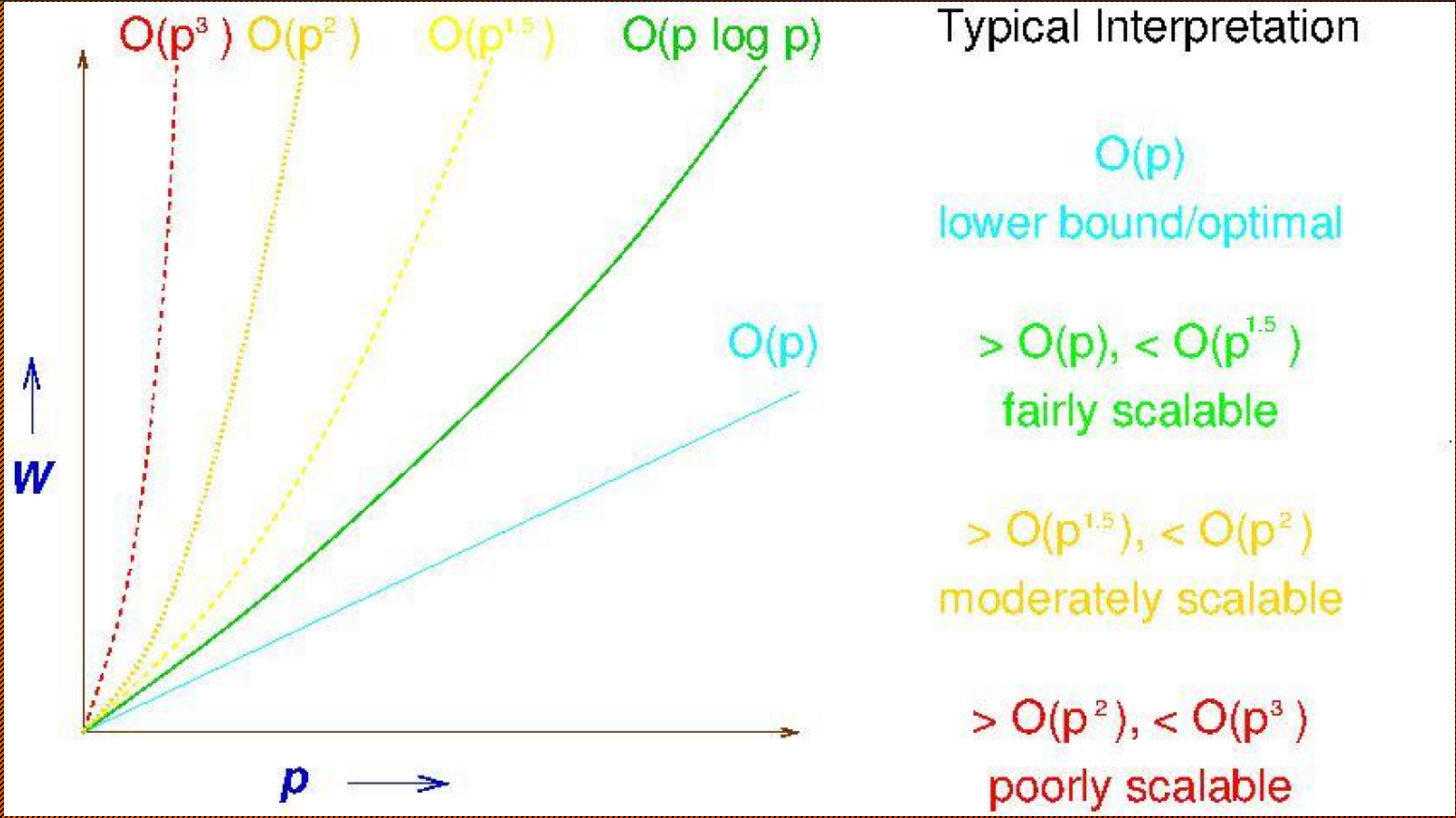
- Function $f_E(p)$ of the number of computing agents p by which the problem size W must grow in order to maintain a given efficiency E.

Isoefficiency function



- Function $f_E(p)$ of the number of computing agents p by which the problem size W must grow with p in order to maintain a given efficiency E.
- Captures the effect of communication, load-imbalance, contention, serial-bottlenecks, etc.

Isoefficiency function



Scalability analysis

$E = S/p = T_S/pT_P$

Scalability analysis

$E = S/p = T_S/pT_P$

Since $T_O = pT_P - T_S$ or $pT_P = T_S + T_O$

Scalability analysis

$E = S/p = T_S/pT_P$

Since $T_O = pT_P - T_S$ or $pT_P = T_S + T_O$

Therefore, $E = T_S/(T_S+T_O)$, or $E = kW/(kW+T_O)$, because $T_S = kW$

Scalability analysis

$E = S/p = T_S/pT_P$

Since $T_O = pT_P - T_S$ or $pT_P = T_S + T_O$

Therefore, $E = T_S/(T_S+T_O)$, or $E = kW/(kW+T_O)$, because $T_S = kW$

$W = T_O.E/k(1-E)$

Scalability analysis

$E = S/p = T_S/pT_P$

Since $T_O = pT_P - T_S$ or $pT_P = T_S+T_O$

Therefore, $E = T_S/(T_S+T_O)$, or $E = kW/(kW+T_O)$, because $T_S = kW$

$W = T_O.E/k(1-E)$

$W \sim T_O$

Scalability analysis: $W = O(n^3)$

Algorithm A

$T_P = O(n^3/p) + O(n^2/\sqrt{p})$

Algorithm B

$T_P = O(n^3/p) + O(n\sqrt{n})$

Scalability analysis: $W = O(n^3)$

Algorithm A

$T_P = O(n^3/p) + O(n^2/\sqrt{p})$

$W = O(n^3) \Rightarrow n^3 \sim n^2\sqrt{p}$

Algorithm B

$T_P = O(n^3/p) + O(n\sqrt{n})$

$W = O(n^3) \Rightarrow n^3 \sim n^{1.5}p$

Scalability analysis: $W = O(n^3)$

Algorithm A

$T_P = O(n^3/p) + O(n^2/\sqrt{p})$

$W = O(n^3) \Rightarrow n^3 \sim n^2\sqrt{p}$

$n \sim \sqrt{p}$

Algorithm B

$T_P = O(n^3/p) + O(n\sqrt{n})$

$W = O(n^3) \Rightarrow n^3 \sim n^{1.5}p$

$n^{1.5} \sim p$

Scalability analysis: $W = O(n^3)$

Algorithm A

$T_P = O(n^3/p) + O(n^2/\sqrt{p})$

$W = O(n^3) \Rightarrow n^3 \sim n^2\sqrt{p}$

$n \sim \sqrt{p}$

$W = O(n^3) = O(p^{1.5})$

Algorithm B

$T_P = O(n^3/p) + O(n\sqrt{n})$

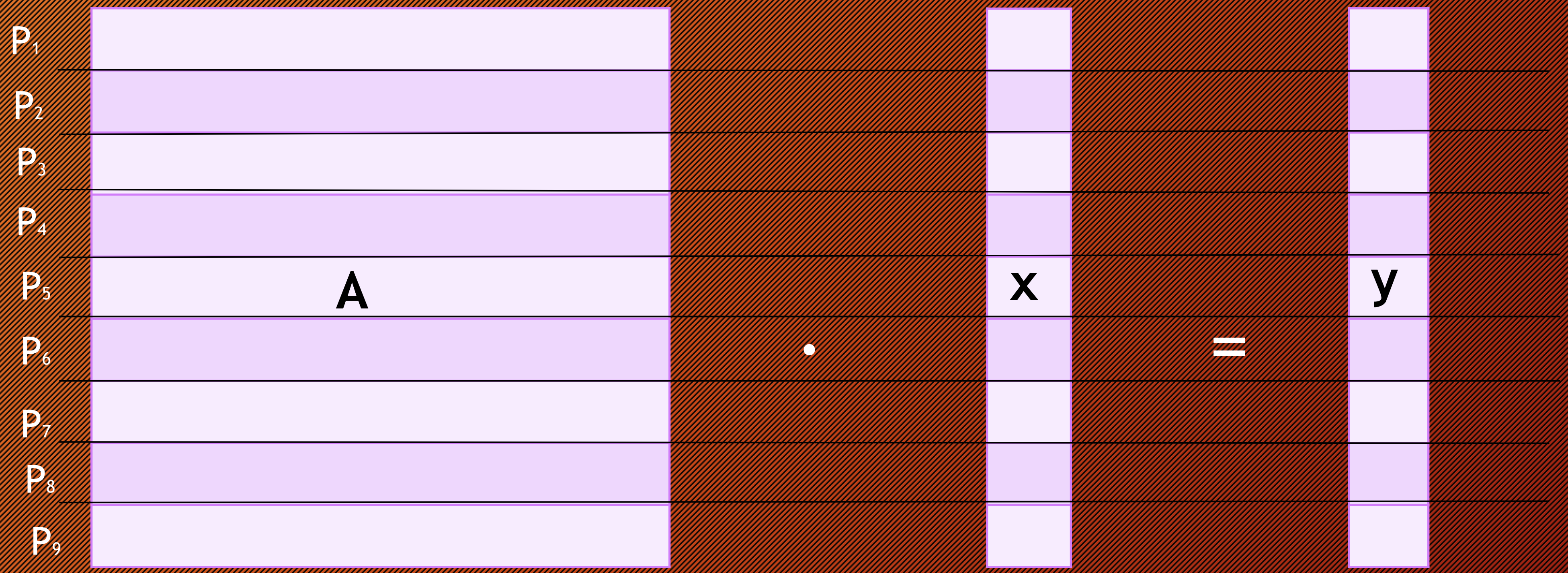
$W = O(n^3) \Rightarrow n^3 \sim n^{1.5}p$

$n^{1.5} \sim p$

$W = O(n^3) = O(p^2)$

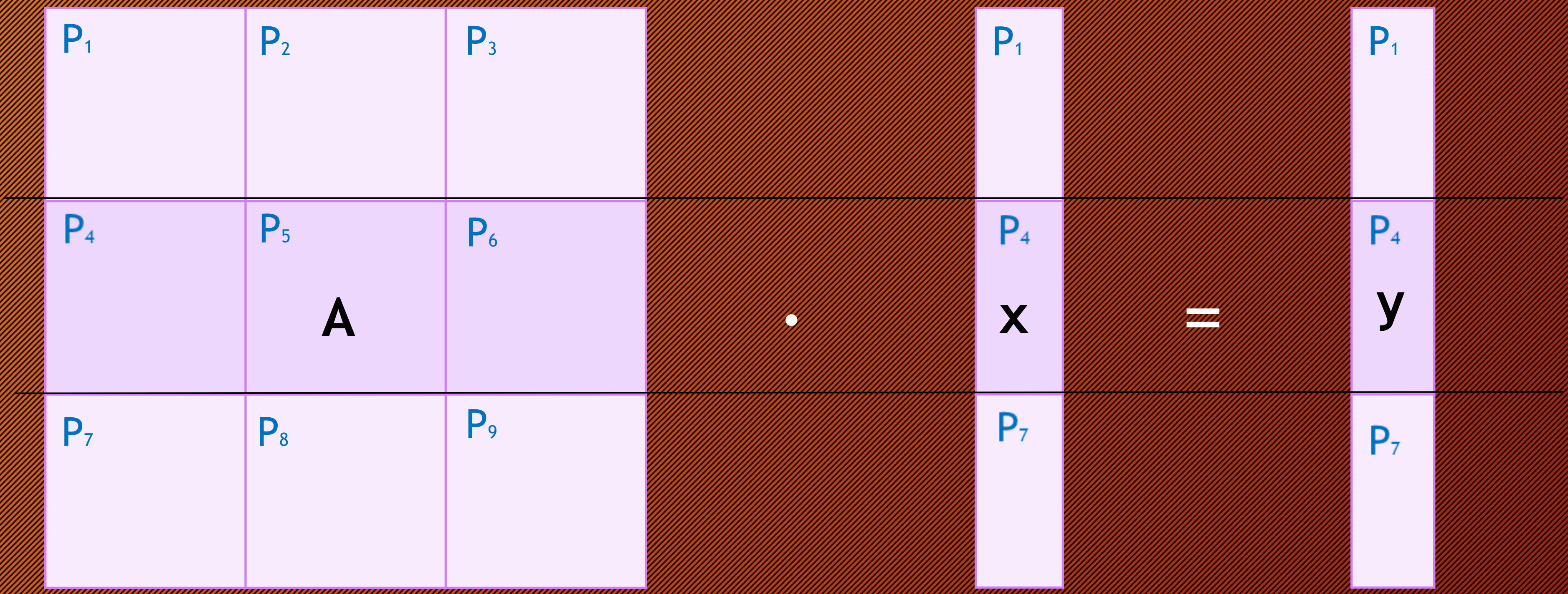
Parallel algorithm design and analysis

Dense Matrix-Vector Multiplication (1-D decomposition)

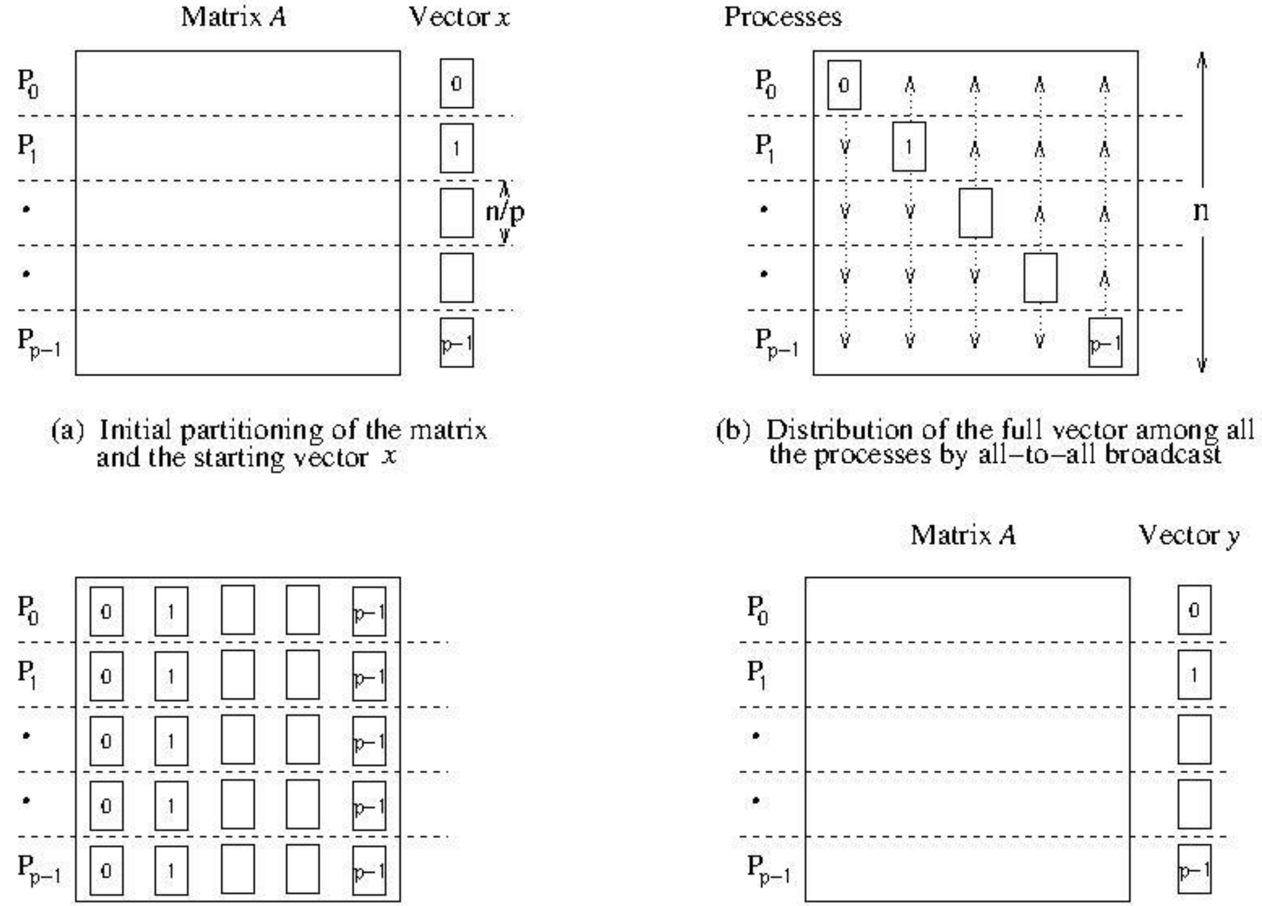


Parallel algorithm design and analysis

Dense Matrix-Vector Multiplication (2-D decomposition)



Parallel matrix-vector multiplication: 1-D decomposition



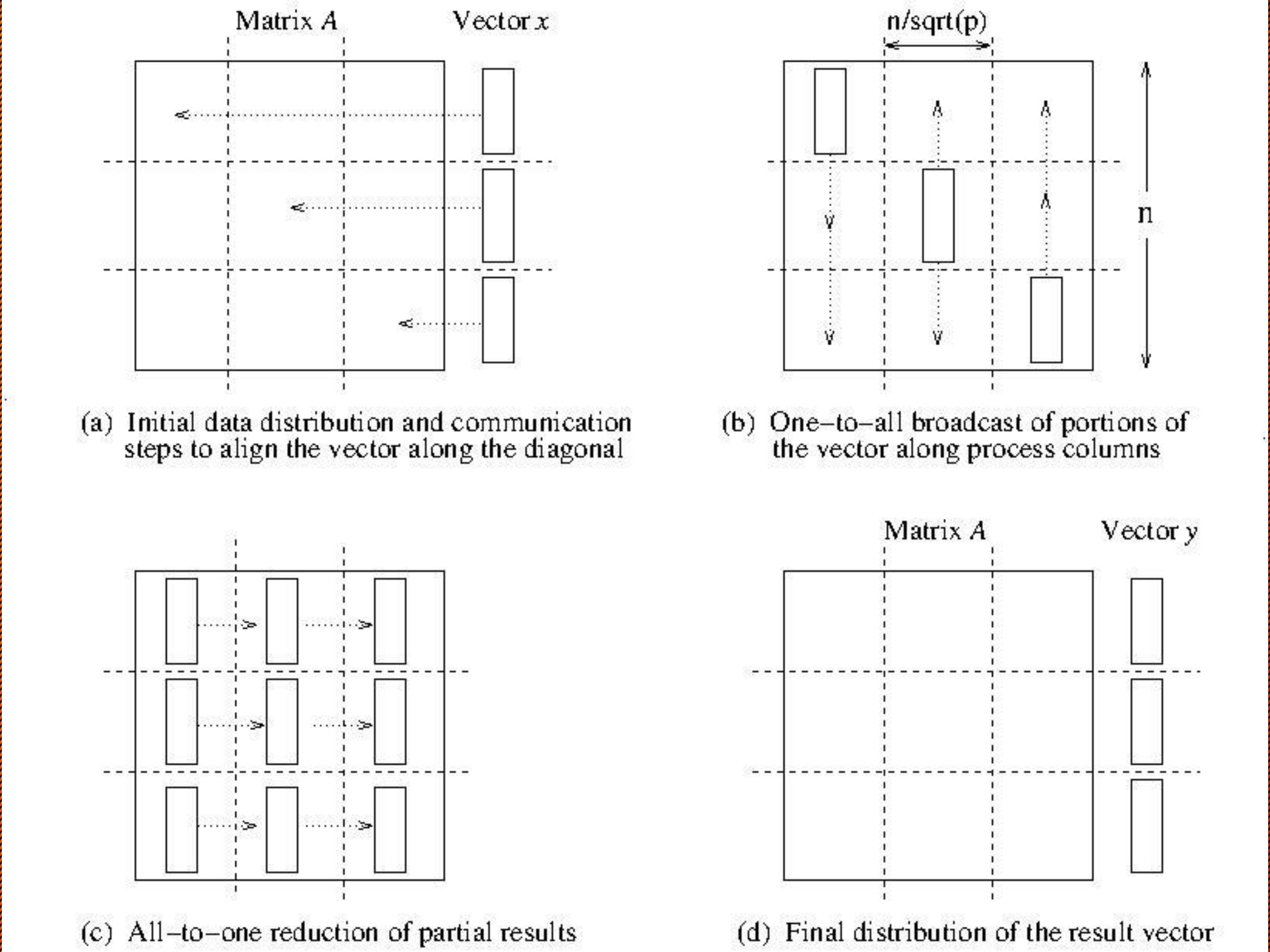
(a) Initial partitioning of the matrix and the starting vector x

(b) Distribution of the full vector among all the processes by all−to−all broadcast

(c) Entire vector distributed to each process after the broadcast

(d) Final distribution of the matrix and the result vector y

Parallel matrix-vector multiplication: 2-D decomposition



Scalability analysis of matrix-vector multiplication (1-D decomposition)

$T_P = n^2/p + t_s \log(p) + t_w n$

Scalability analysis of matrix-vector multiplication (1-D decomposition)

$T_P = n^2/p + t_s\log(p) + t_w n$

$T_O = t_s p\log(p) + t_w pn$

Scalability analysis of matrix-vector multiplication (1-D decomposition)

$T_P = n^2/p + t_s \log(p) + t_w n$

$T_O = t_s p \log(p) + t_w pn$

$W = O(n^2)$

Scalability analysis of matrix-vector multiplication (1-D decomposition)

$T_P = n^2/p + t_s \log(p) + t_w n$

$T_O = t_s p\log(p) + t_w pn$

$W = O(n^2)$

1: $n^2 \sim p\log(p)$

Scalability analysis of matrix-vector multiplication (1-D decomposition)

$T_P = n^2/p + t_s \log(p) + t_w n$

$T_O = t_s p\log(p) + t_w pn$

$W = O(n^2)$

1: $n^2 \sim p\log(p)$

2: $n^2 \sim pn$, or $n \sim p$, or $W = O(n^2) = O(p^2)$

Scalability analysis of matrix-vector multiplication (1-D decomposition)

$T_P = n^2/p + t_s\log(p) + t_w n$

$T_O = t_s p\log(p) + t_w pn$

$W = O(n^2)$

1: $n^2 \sim p\log(p)$

2: $n^2 \sim pn$, or $n \sim p$, or **$\underline{W = O(n^2) = O(p^2)}$**

Scalability analysis of matrix-vector multiplication (2-D decomposition)

$T_P = n^2/p + t_s\log(p) + t_w(n/\sqrt{p})\log(p)$

Scalability analysis of matrix-vector multiplication (2-D decomposition)

$T_P = n^2/p + t_s\log(p) + t_w(n/\sqrt{p})\log(p)$

$T_O = t_s p\log(p) + t_w n\sqrt{p}\log(p)$

Scalability analysis of matrix-vector multiplication (2-D decomposition)

$T_P = n^2/p + t_s\log(p) + t_w(n/\sqrt{p})\log(p)$

$T_O = t_s p\log(p) + t_w n\sqrt{p}\log(p)$

$W = O(n^2)$

Scalability analysis of matrix-vector multiplication (2-D decomposition)

$T_P = n^2/p + t_s\log(p) + t_w(n/\surd p)\log(p)$

$T_O = t_s p\log(p) + t_w n\surd p\log(p)$

$W = O(n^2)$

1: $n^2 \sim p\log(p)$

Scalability analysis of matrix-vector multiplication (2-D decomposition)

$T_P = n^2/p + t_s \log(p) + t_w(n/\sqrt{p})\log(p)$

$T_O = t_s p\log(p) + t_w n\sqrt{p}\log(p)$

$W = O(n^2)$

1: $n^2 \sim p\log(p)$

2: $n^2 \sim n\sqrt{p}\log(p)$, or $n \sim \sqrt{p}\log(p)$,

or $W = O(n^2) = O(p\log^2 p)$

Scalability analysis of matrix-vector multiplication (2-D decomposition)

$T_P = n^2/p + t_s\log(p) + t_w(n/\surd p)\log(p)$

$T_O = t_s p\log(p) + t_w n\surd p\log(p)$

$W = O(n^2)$

1: $n^2 \sim p\log(p)$

2: $n^2 \sim n\surd p\log(p)$, or $n \sim \surd p\log(p)$,

or $\underline{\boldsymbol{W = O(n^2) = O(p\log^2 p)}}$

Isoefficiency function of dense sparse matrix-vector multiplication

1-D decomposition

$W \sim p^2$

2-D decomposition

$W \sim p\log^2 p$

2-D decomposition is likely to yield higher speedups, require smaller problems to deliver the speedups, and scale more readily to larger number of computing agents.

Concluding remarks

- Parallelism necessary for continued performance improvement.

Concluding remarks

- Parallelism necessary for continued performance improvement.
- Complex hierarchy of parallel computing hardware and programming paradigms

Concluding remarks

- Parallelism necessary for continued performance improvement.
- Complex hierarchy of parallel computing hardware and programming paradigms
- Systematic top down parallel application design

Concluding remarks

- Parallelism necessary for continued performance improvement.
- Complex hierarchy of parallel computing hardware and programming paradigms
- Systematic top down parallel application design
- Decomposition strategy is critical

Concluding remarks

- Parallelism necessary for continued performance improvement.
- Complex hierarchy of parallel computing hardware and programming paradigms
- Systematic top down parallel application design
- Decomposition strategy is critical
- Analysis important to understand scalability

Thank you!