Revisiting matrix product on heterogeneous platforms

Jean-François Pineau, Yves Robert, Frédéric Vivien
Jack Dongarra and Zhiao Shi

MAO
January 25, 2007

Outline

Outline

Why revisit matrix-product?

- A fundamental computational kernel
- Well-understood for *homogeneous 2D-arrays of processors*
  - Cannon algorithm
  - ScaLAPACK outer product algorithm
- Communications can no longer be neglected

Why revisit matrix-product?

- A fundamental computational kernel
- Well-understood for *homogeneous 2D-arrays of processors*
  - Cannon algorithm
  - ScaLAPACK outer product algorithm
- Communications can no longer be neglected

Why revisit matrix-product?

- A fundamental computational kernel
- Well-understood for *homogeneous 2D-arrays of processors*
  - Cannon algorithm
  - ScaLAPACK outer product algorithm
- Communications can no longer be neglected

Why revisit matrix-product?

- A fundamental computational kernel
- Well-understood for *homogeneous 2D-arrays of processors*
  - Cannon algorithm
  - ScaLAPACK outer product algorithm
- Communications can no longer be neglected

We target

- heterogeneous clusters
- star-shaped platform
- limited memory

Why revisit matrix-product?

- A fundamental computational kernel
- Well-understood for *homogeneous 2D-arrays of processors*
  - Cannon algorithm
  - ScaLAPACK outer product algorithm
- Communications can no longer be neglected

We target

- heterogeneous clusters
- star-shaped platform
- limited memory

Why revisit matrix-product?

- A fundamental computational kernel
- Well-understood for *homogeneous 2D-arrays of processors*
  - Cannon algorithm
  - ScaLAPACK outer product algorithm
- Communications can no longer be neglected

We target

- heterogeneous clusters
- star-shaped platform
- limited memory

The matrices

- **Input**: Three matrices $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$,
- **Goal**: Compute $\mathcal{C} = \mathcal{C} + \mathcal{A} \times \mathcal{B}$
- *Tools*: Very efficient matrix multiplication algorithms on one processor
- Idea: Manipulate blocks of size $q \times q$

The matrices

- **Input**: Three matrices $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$,
- **Goal**: Compute $\mathcal{C} = \mathcal{C} + \mathcal{A} \times \mathcal{B}$
- *Tools*: Very efficient matrix multiplication algorithms on one processor
- Idea: Manipulate blocks of size $q \times q$

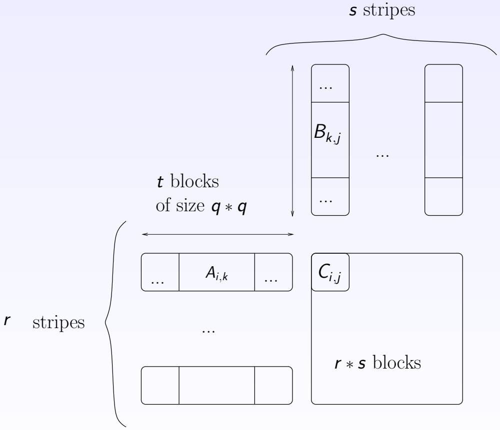
The matrices

- **Input**: Three matrices $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$,
- **Goal**: Compute $\mathcal{C} = \mathcal{C} + \mathcal{A} \times \mathcal{B}$
- *Tools*: Very efficient matrix multiplication algorithms on one processor
- Idea: Manipulate blocks of size $q \times q$

The matrices

- **Input**: Three matrices $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$,
- **Goal**: Compute $\mathcal{C} = \mathcal{C} + \mathcal{A} \times \mathcal{B}$
- *Tools*: Very efficient matrix multiplication algorithms on one processor
- Idea: Manipulate blocks of size $q \times q$

How to split the matrices?



Formally

- $\mathcal{A}$ is of size $n_\mathcal{A} \times n_{\mathcal{AB}}$:
  - split $\mathcal{A}$ into $r = n_\mathcal{A}/q$ horizontal stripes $\mathcal{A}_i$
  - split stripe $\mathcal{A}_i$ into $t = n_{\mathcal{AB}}/q$ square $q \times q$ blocks $\mathcal{A}_{ik}$
- $\mathcal{B}$ is of size $n_{\mathcal{AB}} \times n_\mathcal{B}$:
  - split $\mathcal{B}$ into $s = n_\mathcal{B}/q$ vertical stripes $\mathcal{B}_j$
  - split stripe $\mathcal{B}_j$ into t square $q \times q$ blocks $\mathcal{B}_{kj}$
- $\mathcal{C}$ is of size $n_\mathcal{A} \times n_\mathcal{B}$:
  - split $\mathcal{C}$ into $r \times s$ square $q \times q$ blocks $\mathcal{C}_{ij}$
- All stripes and blocks have same size

Formally

- $\mathcal{A}$ is of size $n_{\mathcal{A}} \times n_{\mathcal{AB}}$:
  - split $\mathcal{A}$ into $r = n_{\mathcal{A}}/q$ horizontal stripes $\mathcal{A}_i$
  - split stripe $\mathcal{A}_i$ into $t = n_{\mathcal{AB}}/q$ square $q \times q$ blocks $\mathcal{A}_{ik}$
- $\mathcal{B}$ is of size $n_{\mathcal{AB}} \times n_{\mathcal{B}}$:
  - split $\mathcal{B}$ into $s = n_{\mathcal{B}}/q$ vertical stripes $\mathcal{B}_j$
  - split stripe $\mathcal{B}_j$ into t square $q \times q$ blocks $\mathcal{B}_{kj}$
- $\mathcal{C}$ is of size $n_{\mathcal{A}} \times n_{\mathcal{B}}$:
  - split $\mathcal{C}$ into $r \times s$ square $q \times q$ blocks $\mathcal{C}_{ij}$
- All stripes and blocks have same size

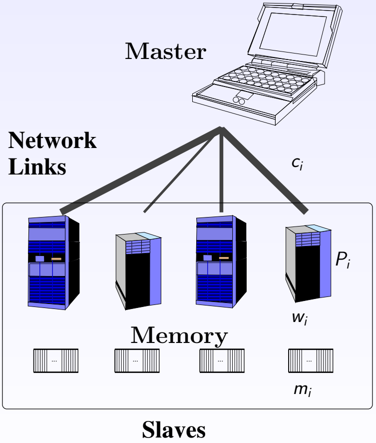
Formally

- $\mathcal{A}$ is of size $n_{\mathcal{A}} \times n_{\mathcal{AB}}$:
  - split $\mathcal{A}$ into $r = n_{\mathcal{A}}/q$ horizontal stripes $\mathcal{A}_i$
  - split stripe $\mathcal{A}_i$ into $t = n_{\mathcal{AB}}/q$ square $q \times q$ blocks $\mathcal{A}_{ik}$
- $\mathcal{B}$ is of size $n_{\mathcal{AB}} \times n_{\mathcal{B}}$:
  - split $\mathcal{B}$ into $s = n_{\mathcal{B}}/q$ vertical stripes $\mathcal{B}_j$
  - split stripe $\mathcal{B}_j$ into t square $q \times q$ blocks $\mathcal{B}_{kj}$
- $\mathcal{C}$ is of size $n_{\mathcal{A}} \times n_{\mathcal{B}}$:
  - split $\mathcal{C}$ into $r \times s$ square $q \times q$ blocks $\mathcal{C}_{ij}$
- All stripes and blocks have same size

Formally

- $\mathcal{A}$ is of size $n_{\mathcal{A}} \times n_{\mathcal{AB}}$:
  - split $\mathcal{A}$ into $r = n_{\mathcal{A}}/q$ horizontal stripes $\mathcal{A}_i$
  - split stripe $\mathcal{A}_i$ into $t = n_{\mathcal{AB}}/q$ square $q \times q$ blocks $\mathcal{A}_{ik}$
- $\mathcal{B}$ is of size $n_{\mathcal{AB}} \times n_{\mathcal{B}}$:
  - split $\mathcal{B}$ into $s = n_{\mathcal{B}}/q$ vertical stripes $\mathcal{B}_j$
  - split stripe $\mathcal{B}_j$ into t square $q \times q$ blocks $\mathcal{B}_{kj}$
- $\mathcal{C}$ is of size $n_{\mathcal{A}} \times n_{\mathcal{B}}$:
  - split $\mathcal{C}$ into $r \times s$ square $q \times q$ blocks $\mathcal{C}_{ij}$
- All stripes and blocks have same size

Platform model



Platform model (Formally)

- Master M and p workers P_i
- P_i needs $X.w_i$ time-units to execute a task of size X
- M needs $X.c_i$ time-units to send/receive a task of size X to/from P_i
- Master has no processing capability
- Enforce *one-port* model:
  - Master involved in a single communication, either send or receive
  - Worker can overlap communication and computation of independent tasks
- Memory limitation: P_i can only store m_i blocks

Platform model (Formally)

- Master M and p workers P_i
- P_i needs $X.w_i$ time-units to execute a task of size X
- M needs $X.c_i$ time-units to send/receive a task of size X to/from P_i
- Master has no processing capability
- Enforce *one-port* model:
    - Master involved in a single communication, either send or receive
    - Worker can overlap communication and computation of independent tasks
- Memory limitation: P_i can only store m_i blocks

Platform model (Formally)

- Master M and p workers P_i
- P_i needs $X.w_i$ time-units to execute a task of size X
- M needs $X.c_i$ time-units to send/receive a task of size X to/from P_i
- Master has no processing capability
- Enforce *one-port* model:
  - Master involved in a single communication, either send or receive
  - Worker can overlap communication and computation of independent tasks
- Memory limitation: P_i can only store m_i blocks

Platform model (Formally)

- Master M and p workers P_i
- P_i needs $X.w_i$ time-units to execute a task of size X
- M needs $X.c_i$ time-units to send/receive a task of size X to/from P_i
- Master has no processing capability
- Enforce *one-port* model:
  - Master involved in a single communication, either send or receive
  - Worker can overlap communication and computation of independent tasks
- Memory limitation: P_i can only store m_i blocks

Platform model (Formally)

- Master M and p workers P_i
- P_i needs $X.w_i$ time-units to execute a task of size X
- M needs $X.c_i$ time-units to send/receive a task of size X to/from P_i
- Master has no processing capability
- Enforce *one-port* model:
  - Master involved in a single communication, either send or receive
  - Worker can overlap communication and computation of independent tasks
- Memory limitation: P_i can only store m_i blocks

Platform model (Formally)

- Master M and p workers P_i
- P_i needs $X.w_i$ time-units to execute a task of size X
- M needs $X.c_i$ time-units to send/receive a task of size X to/from P_i
- Master has no processing capability
- Enforce *one-port* model:
  - Master involved in a single communication, either send or receive
  - Worker can overlap communication and computation of independent tasks
- Memory limitation: P_i can only store m_i blocks

Platform model (Formally)

- Master M and p workers P_i
- P_i needs $X.w_i$ time-units to execute a task of size X
- M needs $X.c_i$ time-units to send/receive a task of size X to/from P_i
- Master has no processing capability
- Enforce *one-port* model:
  - Master involved in a single communication, either send or receive
  - Worker can overlap communication and computation of independent tasks
- Memory limitation: P_i can only store m_i blocks

Platform model (Formally)

- Master M and p workers P_i
- P_i needs $X.w_i$ time-units to execute a task of size X
- M needs $X.c_i$ time-units to send/receive a task of size X to/from P_i
- Master has no processing capability
- Enforce *one-port* model:
  - Master involved in a single communication, either send or receive
  - Worker can overlap communication and computation of independent tasks
- Memory limitation: P_i can only store m_i blocks

Outline

Outline

What is the simplest problem?

Problem

- One worker
- Stripes instead of blocks, no return result
- No memory limitation

Scheduling

- In which order send the files?

What is the simplest problem?

Problem

- One worker
- Stripes instead of blocks, no return result
- No memory limitation

Scheduling

- In which order send the files?

What is the simplest problem?

Problem

- One worker
- Stripes instead of blocks, no return result
- No memory limitation

Scheduling

- In which order send the files?

What is the simplest problem?

Problem

- One worker
- Stripes instead of blocks, no return result
- No memory limitation

Scheduling

- In which order send the files?

What is remaining?

Parameters

- Platform:
  - c : communication time of a stripe
  - w : processing time of a stripe
- Application: r and s (number of stripes)

Objective

- Design optimal algorithm for makespan minimization

What is remaining?

Parameters

- Platform:
  - c : communication time of a stripe
  - w : processing time of a stripe
- Application: r and s (number of stripes)

Objective

- Design optimal algorithm for makespan minimization

What is remaining?

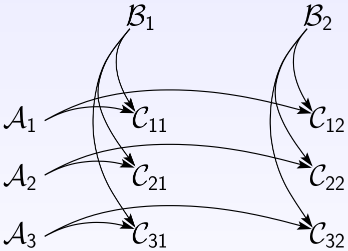
Parameters

- Platform:
  - c : communication time of a stripe
  - w : processing time of a stripe
- Application: r and s (number of stripes)

Objective

- Design optimal algorithm for makespan minimization

Dependence graph: files and tasks



Suggests alternating sends of $\mathcal{A}$ and $\mathcal{B}$

Alternating greedy

Alternating greedy

- Master sends stripes as soon as possible
- Alternates a stripe of type $\mathcal{A}$ and a stripe of type $\mathcal{B}$

Theorem

With a single worker, the alternating greedy algorithm is optimal.

Alternating greedy

Alternating greedy

- Master sends stripes as soon as possible
- Alternates a stripe of type $\mathcal{A}$ and a stripe of type $\mathcal{B}$

Theorem

With a single worker, the alternating greedy algorithm is optimal.

What is the second simplest problem?

Problem

- Fully homogeneous platform (identical workers and communication links)
- Stripes instead of blocks, no return result
- No memory limitation

Scheduling

- How many workers to enroll?
- Which files sent to which workers, and in which order?

What is the second simplest problem?

Problem

- Fully homogeneous platform (identical workers and communication links)
- Stripes instead of blocks, no return result
- No memory limitation

Scheduling

- How many workers to enroll?
- Which files sent to which workers, and in which order?

What is remaining?

Parameters

- Platform:
  - p workers,
  - c : communication time of a stripe
  - w : processing time of a stripe
- Application: r and s (number of stripes)

Objective

- Makespan minimization
- Design optimal algorithm (includes resource selection)

What is remaining?

Parameters

- Platform:
  - p workers,
  - c : communication time of a stripe
  - w : processing time of a stripe
- Application: r and s (number of stripes)

Objective

- Makespan minimization
- Design optimal algorithm (includes resource selection)

Extend the *Alternating greedy* algorithm to several workers

Thrifty: a natural greedy algorithm

- Send enough tasks to first worker so that it is never idle
- Send tasks to second worker during available communication slots
- Enroll new worker only when all previous ones are not delayed

Min-min: another natural greedy algorithm

- Min-min heuristic
- Start best new task on best processor

Extend the *Alternating greedy* algorithm to several workers

Thrifty: a natural greedy algorithm

- Send enough tasks to first worker so that it is never idle
- Send tasks to second worker during available communication slots
- Enroll new worker only when all previous ones are not delayed

Min-min: another natural greedy algorithm

- Min-min heuristic
- Start best new task on best processor

Extend the *Alternating greedy* algorithm to several workers

Thrifty: a natural greedy algorithm

- Send enough tasks to first worker so that it is never idle
- Send tasks to second worker during available communication slots
- Enroll new worker only when all previous ones are not delayed

Min-min: another natural greedy algorithm

- Min-min heuristic
- Start best new task on best processor

Extend the *Alternating greedy* algorithm to several workers

Thrifty: a natural greedy algorithm

- Send enough tasks to first worker so that it is never idle
- Send tasks to second worker during available communication slots
- Enroll new worker only when all previous ones are not delayed

Min-min: another natural greedy algorithm

- Min-min heuristic
- Start best new task on best processor

Extend the *Alternating greedy* algorithm to several workers

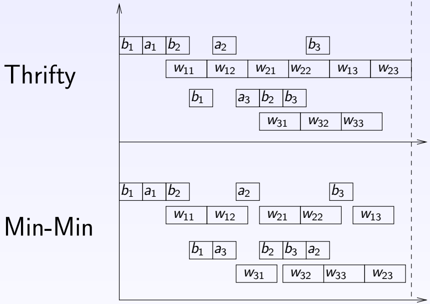
Thrifty: a natural greedy algorithm

- Send enough tasks to first worker so that it is never idle
- Send tasks to second worker during available communication slots
- Enroll new worker only when all previous ones are not delayed

Min-min: another natural greedy algorithm

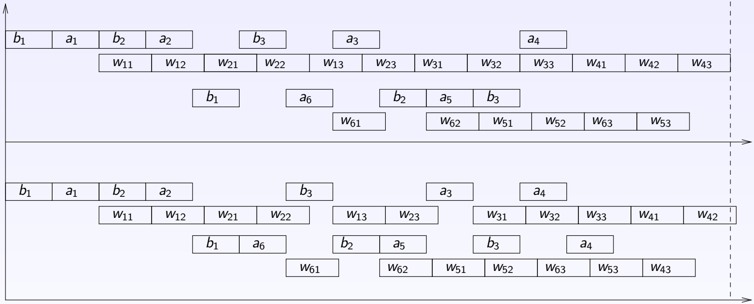
- Min-min heuristic
- Start best new task on best processor

Thrifty Optimal ?



$p = 2,\ c = 4,\ w = 7,\ r = s = 3,$ Min-min wins

Min-Min Optimal ?



$p = 2,\ c = 8,\ w = 9,\ r = 6,\ s = 3,$ Thrifty wins

Outline

New problem

- Homogeneous platform
- Master sends blocks $\mathcal{A}_{ik}$, $\mathcal{B}_{kj}$, and $\mathcal{C}_{ij}$
- Master retrieves final values of blocks $\mathcal{C}_{ij}$
- Memory limitation: only m buffers available
  $\longrightarrow$ at most m blocks simultaneously stored on worker

New problem

- Homogeneous platform
- Master sends blocks $\mathcal{A}_{ik}$, $\mathcal{B}_{kj}$, and $\mathcal{C}_{ij}$
- Master retrieves final values of blocks $\mathcal{C}_{ij}$
- Memory limitation: only m buffers available
  $\longrightarrow$ at most m blocks simultaneously stored on worker

New problem

- Homogeneous platform
- Master sends blocks $\mathcal{A}_{ik}$, $\mathcal{B}_{kj}$, and $\mathcal{C}_{ij}$
- Master retrieves final values of blocks $\mathcal{C}_{ij}$
- Memory limitation: only m buffers available
  $\longrightarrow$ at most m blocks simultaneously stored on worker

New problem

- Homogeneous platform

- Master sends blocks $\mathcal{A}_{ik}$, $\mathcal{B}_{kj}$, and $\mathcal{C}_{ij}$
- Master retrieves final values of blocks $\mathcal{C}_{ij}$
- Memory limitation: only m buffers available
  $\longrightarrow$ at most m blocks simultaneously stored on worker

New problem

- Homogeneous platform

- Master sends blocks $\mathcal{A}_{ik}$, $\mathcal{B}_{kj}$, and $\mathcal{C}_{ij}$
- Master retrieves final values of blocks $\mathcal{C}_{ij}$
- Memory limitation: only m buffers available
  $\longrightarrow$ at most m blocks simultaneously stored on worker

Previous objective

- Minimizing the total execution time

New problem

- Homogeneous platform

- Master sends blocks $\mathcal{A}_{ik}$, $\mathcal{B}_{kj}$, and $\mathcal{C}_{ij}$
- Master retrieves final values of blocks $\mathcal{C}_{ij}$
- Memory limitation: only m buffers available
  $\longrightarrow$ at most m blocks simultaneously stored on worker

New objective

- Minimizing the total communication volume

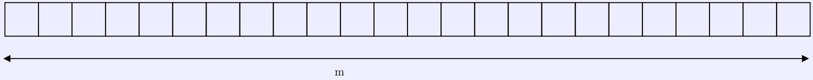
New problem

- Homogeneous platform
  (*Any parallel algorithm can be simulated on one single worker*)
- Master sends blocks $\mathcal{A}_{ik}$, $\mathcal{B}_{kj}$, and $\mathcal{C}_{ij}$
- Master retrieves final values of blocks $\mathcal{C}_{ij}$
- Memory limitation: only m buffers available
  $\longrightarrow$ at most m blocks simultaneously stored on worker

New objective

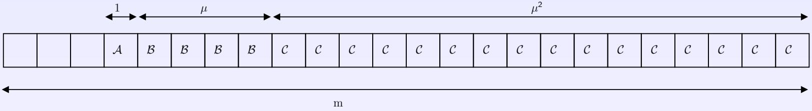
- Minimizing the total communication volume

The *maximum re-use* algorithm



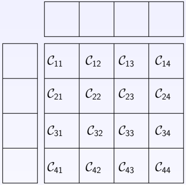
- Find largest μ such that $1 + \mu + \mu^2 \leq m$

The *maximum re-use* algorithm



- Find largest μ such that $1 + \mu + \mu^2 \leq m$

The *maximum re-use* algorithm



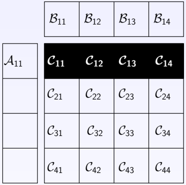
- Store $\mu \times \mu$ blocks of $\mathcal{C}$ in memory

The *maximum re-use* algorithm

	$\mathcal{B}_{11}$	$\mathcal{B}_{12}$	$\mathcal{B}_{13}$	$\mathcal{B}_{14}$
	$\mathcal{C}_{11}$	$\mathcal{C}_{12}$	$\mathcal{C}_{13}$	$\mathcal{C}_{14}$
	$\mathcal{C}_{21}$	$\mathcal{C}_{22}$	$\mathcal{C}_{23}$	$\mathcal{C}_{24}$
	$\mathcal{C}_{31}$	$\mathcal{C}_{32}$	$\mathcal{C}_{33}$	$\mathcal{C}_{34}$
	$\mathcal{C}_{41}$	$\mathcal{C}_{42}$	$\mathcal{C}_{43}$	$\mathcal{C}_{44}$

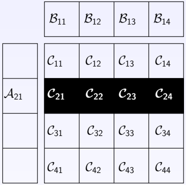
- For each k from 1 to t:
  - Send corresponding μ elements of the k^{th} row of $\mathcal{B}$

The *maximum re-use* algorithm



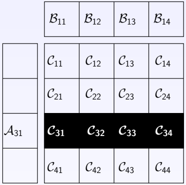
- For each k from 1 to t:
  - Sequentially send corresponding μ elements of the k^{th} column of $\mathcal{A}$. For each block of $\mathcal{A}$, update μ elements of $\mathcal{C}$

The *maximum re-use* algorithm



- For each k from 1 to t:
  - Sequentially send corresponding μ elements of the k^{th} column of $\mathcal{A}$. For each block of $\mathcal{A}$, update μ elements of $\mathcal{C}$

The *maximum re-use* algorithm



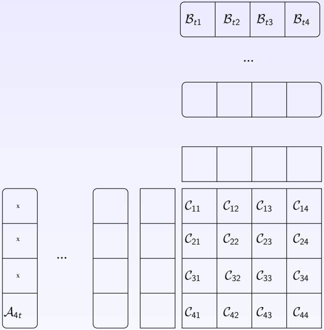
- For each k from 1 to t:
  - Sequentially send corresponding μ elements of the k^{th} column of $\mathcal{A}$. For each block of $\mathcal{A}$, update μ elements of $\mathcal{C}$

The *maximum re-use* algorithm

	$\mathcal{B}_{11}$	$\mathcal{B}_{12}$	$\mathcal{B}_{13}$	$\mathcal{B}_{14}$
	$\mathcal{C}_{11}$	$\mathcal{C}_{12}$	$\mathcal{C}_{13}$	$\mathcal{C}_{14}$
	$\mathcal{C}_{21}$	$\mathcal{C}_{22}$	$\mathcal{C}_{23}$	$\mathcal{C}_{24}$
	$\mathcal{C}_{31}$	$\mathcal{C}_{32}$	$\mathcal{C}_{33}$	$\mathcal{C}_{34}$
$\mathcal{A}_{41}$	$\mathcal{C}_{41}$	$\mathcal{C}_{42}$	$\mathcal{C}_{43}$	$\mathcal{C}_{44}$

- For each k from 1 to t:
  - Sequentially send corresponding μ elements of the k^{th} column of $\mathcal{A}$. For each block of $\mathcal{A}$, update μ elements of $\mathcal{C}$

The *maximum re-use* algorithm



- Return results to master

Performance

- Need $2\mu^2$ communications to send/retrieve $\mathcal{C}$
- For each value of t:
  - need μ elements of $\mathcal{A}$ and μ elements of $\mathcal{B}$
  - perform rank-1 update of $\mathcal{C}$ square $\longrightarrow$ μ^2 computations
- Communication-to-computation ratio:

$$\frac{2\mu^2 + 2\mu t}{\mu^2 t} = \frac{2}{t} + \frac{2}{\mu} \to \frac{2}{\sqrt{m}}$$

Performance

- Need $2\mu^2$ communications to send/retrieve $\mathcal{C}$
- For each value of t:
  - need μ elements of $\mathcal{A}$ and μ elements of $\mathcal{B}$
  - perform rank-1 update of $\mathcal{C}$ square $\rightarrow$ μ^2 computations
- Communication-to-computation ratio:

$$\frac{2\mu^2 + 2\mu t}{\mu^2 t} = \frac{2}{t} + \frac{2}{\mu} \rightarrow \frac{2}{\sqrt{m}}$$

Performance

- Need $2\mu^2$ communications to send/retrieve $\mathcal{C}$
- For each value of t:
  - need μ elements of $\mathcal{A}$ and μ elements of $\mathcal{B}$
  - perform rank-1 update of $\mathcal{C}$ square $\rightarrow$ μ^2 computations
- Communication-to-computation ratio:

$$\frac{2\mu^2 + 2\mu t}{\mu^2 t} = \frac{2}{t} + \frac{2}{\mu} \rightarrow \frac{2}{\sqrt{m}}$$

Assessing that performance

- Estimate number of computations made during m consecutive communication steps
- Notations:
  - α_{old}, β_{old}, and γ_{old} number of buffers dedicated to $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ at the beginning
  - α_{recv}, β_{recv}, and γ_{recv} number of $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ elements sent by master during m steps
  - γ_{sent} number of $\mathcal{C}$ elements returned to master during m steps

Assessing that performance

- Estimate number of computations made during m consecutive communication steps
- Notations:
  - α_{old}, β_{old}, and γ_{old} number of buffers dedicated to $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ at the beginning
  - α_{recv}, β_{recv}, and γ_{recv} number of $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ elements sent by master during m steps
  - γ_{sent} number of $\mathcal{C}$ elements returned to master during m steps

Assessing that performance

- Equations:

$$\begin{cases} \alpha_{old} + \beta_{old} + \gamma_{old} \leq m \\ \alpha_{recv} + \beta_{recv} + \gamma_{recv} + \gamma_{sent} = m \end{cases}$$

- Simplify notations:

$$\begin{cases} \alpha_{old} + \alpha_{recv} = \alpha m \\ \beta_{old} + \beta_{recv} = \beta m \\ \gamma_{old} + \gamma_{recv} = \gamma m \end{cases}$$

Assessing that performance

- Equations:

$$\begin{cases} \alpha_{old} + \beta_{old} + \gamma_{old} \leq m \\ \alpha_{recv} + \beta_{recv} + \gamma_{recv} + \gamma_{sent} = m \end{cases}$$

- Simplify notations:

$$\begin{cases} \alpha_{old} + \alpha_{recv} = \alpha m \\ \beta_{old} + \beta_{recv} = \beta m \\ \gamma_{old} + \gamma_{recv} = \gamma m \end{cases}$$

Assessing the performance

Loomis-Whitney inequality

if N_A elements of $\mathcal{A}$, N_B elements of $\mathcal{B}$ and N_C elements of $\mathcal{C}$ are accessed, then no more than K computations can be done:

$$K = \sqrt{N_A N_B N_C}$$

- Here

$$K = \sqrt{\alpha + \beta + \gamma} \times m\sqrt{m}$$

Assessing the performance

- Solution:

$$\alpha = \beta = \gamma = \frac{2}{3}, \text{ AND } k = \sqrt{\frac{8}{27}}$$

- Lower bound for communication-to-computation ratio:

$$\frac{m}{K} = \frac{m}{km\sqrt{m}} = \sqrt{\frac{27}{8m}}$$

- *Maximum re-use algorithm* communication-to-computation ratio:

$$\frac{2}{\sqrt{m}} = \sqrt{\frac{32}{8m}}$$

Assessing the performance

- Solution:

$$\alpha = \beta = \gamma = \frac{2}{3}, \quad \text{AND} \quad k = \sqrt{\frac{8}{27}}$$

- Lower bound for communication-to-computation ratio:

$$\frac{m}{K} = \frac{m}{km\sqrt{m}} = \sqrt{\frac{27}{8m}}$$

- *Maximum re-use algorithm* communication-to-computation ratio:

$$\frac{2}{\sqrt{m}} = \sqrt{\frac{32}{8m}}$$

Assessing the performance

- Solution:

$$\alpha = \beta = \gamma = \frac{2}{3}, \quad \text{AND} \quad k = \sqrt{\frac{8}{27}}$$

- Lower bound for communication-to-computation ratio:

$$\frac{m}{K} = \frac{m}{km\sqrt{m}} = \sqrt{\frac{27}{8m}}$$

- *Maximum re-use algorithm* communication-to-computation ratio:

$$\frac{2}{\sqrt{m}} = \sqrt{\frac{32}{8m}}$$

Outline

Outline

With several workers

Problem

- How to extend the *maximum re-use* algorithm?
- How many workers to enroll?

With several workers

Problem

- How to extend the *maximum re-use* algorithm?
- How many workers to enroll?

With several workers

Solution

- How to extend the *maximum re-use* algorithm?
  - Send files to workers according to the *maximum re-use* algorithm in a *Round-Robin* way
- How many workers to enroll?

With several workers

Solution

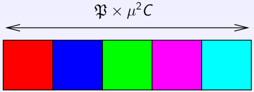
- How to extend the *maximum re-use* algorithm?
  - Send files to workers according to the *maximum re-use* algorithm in a *Round-Robin* way
- How many workers to enroll?
  - Enroll workers until the first one finishes its task

Resource selection

$c = 2,\ w = 4.5,\ \mu = 4,\ t = 100$, enroll $\mathfrak{P} = 5$ workers

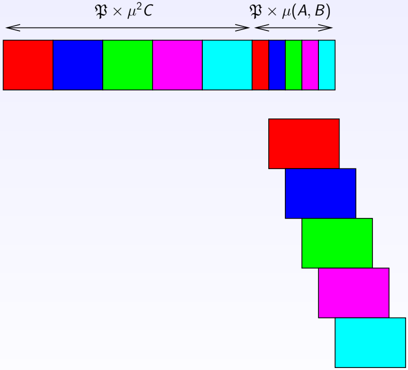
Resource selection

$$c = 2,\ w = 4.5,\ \mu = 4,\ t = 100,\ \text{enroll}\ \mathfrak{P} = 5\ \text{workers}$$



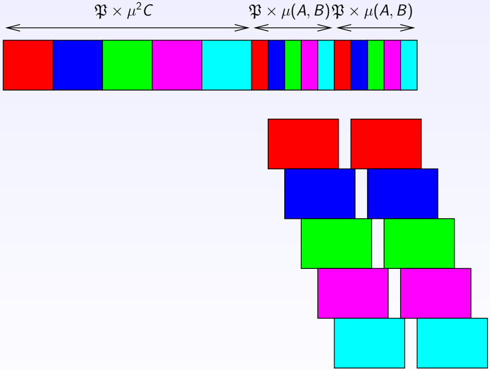
Resource selection

$$c = 2,\ w = 4.5,\ \mu = 4,\ t = 100,\ \text{enroll}\ \mathfrak{P} = 5\ \text{workers}$$



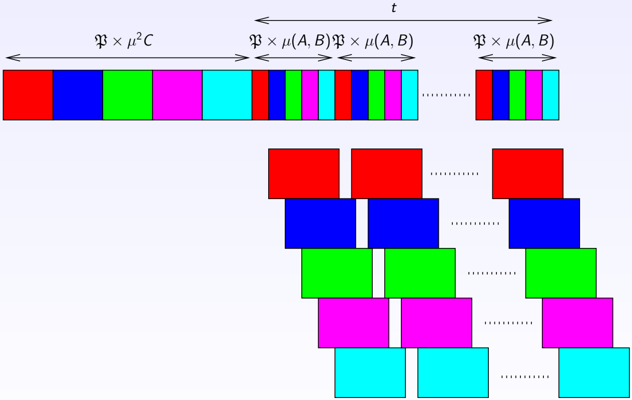
Resource selection

$c = 2,\ w = 4.5,\ \mu = 4,\ t = 100,$ enroll $\mathfrak{P} = 5$ workers



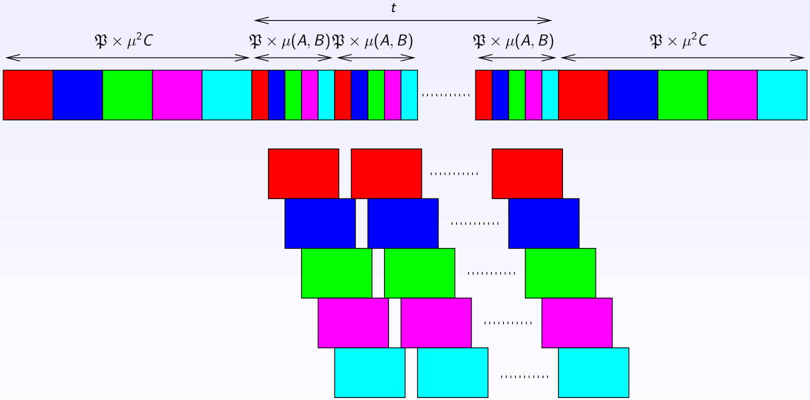
Resource selection

$c = 2,\ w = 4.5,\ \mu = 4,\ t = 100,$ enroll $\mathfrak{P} = 5$ workers



Resource selection

$c = 2,\ w = 4.5,\ \mu = 4,\ t = 100,$ enroll $\mathfrak{P} = 5$ workers



Performance

- Assume $\mathfrak{P} \leq p$ participating workers
- In a round (computing a $\mathcal{C}$ block entirely), master communicates with each worker:
  - $2\mu^2$ blocks of $\mathcal{C}$ (either sent or received)
  - $2\mu t$ blocks of $\mathcal{A}$ and $\mathcal{B}$
- In a round, each worker computes $\mu^2 t$ updates
- For large t, neglect input/output of C blocks, and find the smallest $\mathfrak{P}$ such that

$$(2\mu tc) \times \mathfrak{P} \geq \mu^2 tw \Leftrightarrow \mathfrak{P} = \left\lceil \frac{\mu w}{2c} \right\rceil$$

In the example, $\mathfrak{P} = \lceil 4.5 \rceil$

Performance

- Assume $\mathfrak{P} \leq p$ participating workers
- In a round (computing a $\mathcal{C}$ block entirely), master communicates with each worker:
  - $2\mu^2$ blocks of $\mathcal{C}$ (either sent or received)
  - $2\mu t$ blocks of $\mathcal{A}$ and $\mathcal{B}$
- In a round, each worker computes $\mu^2 t$ updates
- For large t, neglect input/output of C blocks, and find the smallest $\mathfrak{P}$ such that

$$(2\mu tc) \times \mathfrak{P} \geq \mu^2 tw \Leftrightarrow \mathfrak{P} = \left\lceil \frac{\mu w}{2c} \right\rceil$$

In the example, $\mathfrak{P} = \lceil 4.5 \rceil$

Performance

- Assume $\mathfrak{P} \leq p$ participating workers
- In a round (computing a $\mathcal{C}$ block entirely), master communicates with each worker:
  - $2\mu^2$ blocks of $\mathcal{C}$ (either sent or received)
  - $2\mu t$ blocks of $\mathcal{A}$ and $\mathcal{B}$
- In a round, each worker computes $\mu^2 t$ updates
- For large t, neglect input/output of C blocks, and find the smallest $\mathfrak{P}$ such that

$$(2\mu tc)\times\mathfrak{P} \geq \mu^2 tw \Leftrightarrow \mathfrak{P} = \left\lceil \frac{\mu w}{2c} \right\rceil$$ In the example, $\mathfrak{P} = \lceil 4.5 \rceil$

Performance

- Assume $\mathfrak{P} \leq p$ participating workers
- In a round (computing a $\mathcal{C}$ block entirely), master communicates with each worker:
  - $2\mu^2$ blocks of $\mathcal{C}$ (either sent or received)
  - $2\mu t$ blocks of $\mathcal{A}$ and $\mathcal{B}$
- In a round, each worker computes $\mu^2 t$ updates
- For large t, neglect input/output of C blocks, and find the smallest $\mathfrak{P}$ such that

$$(2\mu tc) \times \mathfrak{P} \geq \mu^2 tw \;\Leftrightarrow\; \mathfrak{P} = \left\lceil \frac{\mu w}{2c} \right\rceil$$

In the example, $\mathfrak{P} = \lceil 4.5 \rceil$

Performance

- Assume $\mathfrak{P} \leq p$ participating workers
- In a round (computing a $\mathcal{C}$ block entirely), master communicates with each worker:
  - $2\mu^2$ blocks of $\mathcal{C}$ (either sent or received)
  - $2\mu t$ blocks of $\mathcal{A}$ and $\mathcal{B}$
- In a round, each worker computes $\mu^2 t$ updates
- For large t, neglect input/output of C blocks, and find the smallest $\mathfrak{P}$ such that

$$(2\mu tc)\times\mathfrak{P} \geq \mu^2 tw \;\Leftrightarrow \mathfrak{P} = \left\lceil \frac{\mu w}{2c} \right\rceil$$

In the example, $\mathfrak{P} = \lceil 4.5 \rceil$

Outline

Resource selection

Problem

- Each worker P_i has parameters c_i, w_i, and $\mu_i = \sqrt{m_i}$.
- Each participating P_i needs $\delta_i = 2\mu_i t c_i$ communications to process $\phi_i = t\mu_i^2 w_i$ computations (neglect I/O for C blocks)
- Which workers to enroll?

Steady-State

- In steady-state, P_i receives y_i A and B blocks per time-unit
- In steady-state, P_i computes x_i C blocks per time-unit

$$\left\{\begin{array}{l} \textsc{Maximize } \sum_i x_i \\ \textsc{subject to} \\ \qquad \dfrac{x_i}{\mu_i^2} \leq \dfrac{y_i}{2\mu_i} \\ \qquad x_i w_i \leq 1 \\ \qquad \displaystyle\sum_i y_i c_i \leq 1 \end{array}\right.$$

Steady-State

$$\left\{\begin{array}{l}\textsc{Maximize}\ \sum_i x_i \\ \textsc{subject to} \\ \qquad \dfrac{x_i}{\mu_i^2} \leq \dfrac{y_i}{2\mu_i} \\ \qquad x_i w_i \leq 1 \\ \qquad \displaystyle\sum_i y_i c_i \leq 1 \end{array}\right.$$

Claim $y_i = \frac{2x_i}{\mu_i}$

$$\Leftrightarrow \left\{\begin{array}{l}\textsc{Maximize}\ \sum_i x_i \\ \textsc{subject to} \\ \qquad x_i \leq \dfrac{1}{w_i} \\ \qquad \displaystyle\sum_i \dfrac{2c_i}{\mu_i} x_i \leq 1 \end{array}\right.$$

Steady-State

$$\begin{cases} \textsc{Maximize} \ \sum_i x_i \\ \textsc{subject to} \\ \qquad x_i \leq \dfrac{1}{w_i} \\ \qquad \displaystyle\sum_i \frac{2c_i}{\mu_i} x_i \leq 1 \end{cases}$$

- Bandwidth-centric strategy:
  - Sort workers by non-decreasing $\frac{2c_i}{\mu_i}$
  - Enroll them as long as $\sum \frac{2c_i}{\mu_i w_i} \leq 1$
  - Achieve throughput $\rho \approx \sum_{i \text{ enrolled}} \frac{1}{w_i}$

Steady-State

$$\begin{cases} \text{MAXIMIZE } \sum_i x_i \\ \text{SUBJECT TO} \\ \qquad x_i \leq \dfrac{1}{w_i} \end{cases}$$

Eh wait!

Do you have enough memory?!

- Bandwidth-
  - Sort workers by non-decreasing $\frac{2c_i}{\mu_i}$
  - Enroll them as long as $\sum \frac{2c_i}{\mu_i w_i} \leq 1$
  - Achieve throughput $\rho \approx \sum_{i \text{ enrolled}} \frac{1}{w_i}$

No, we don't have enough memory!

	P_1	P_2
c_i	1	20
w_i	2	40
μ_i	2	2
$\frac{2c_i}{\mu_i w_i}$	$\frac{1}{2}$	$\frac{1}{2}$

- Every 160 seconds:
  - P_1 receives 80 blocks (20 $\mu_1 \times \mu_1$ chunks) in 80 seconds
  - P_1 computes 80 blocks in 160 seconds
  - P_2 receives 4 blocks (1 $\mu_2 \times \mu_2$ chunk) in 80 seconds
  - P_2 computes 4 blocks in 160 seconds
- P_1 computes two slowly and needs buffers to store 20 blocks!

$$\underbrace{11111111111111111111}_{P_1}\ \underbrace{20}_{P_2}\ \underbrace{11111111111111111111}_{P_1}\ \underbrace{20}_{P_2}\ \underbrace{111111111\ldots}_{P_1\ldots}$$

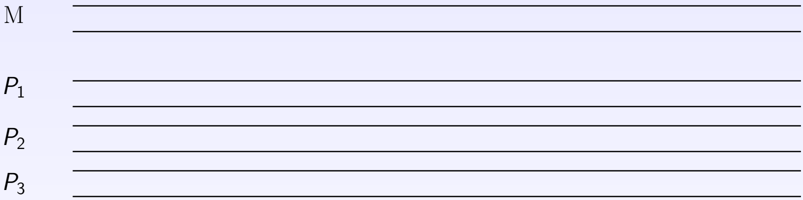
No, we don't have enough memory!

	P_1	P_2
c_i	1	20
w_i	2	40
μ_i	2	2
$\frac{2c_i}{\mu_i w_i}$	$\frac{1}{2}$	$\frac{1}{2}$

- Every 160 seconds:
  - P_1 receives 80 blocks (20 $\mu_1 \times \mu_1$ chunks) in 80 seconds
  - P_1 computes 80 blocks in 160 seconds
  - P_2 receives 4 blocks (1 $\mu_2 \times \mu_2$ chunk) in 80 seconds
  - P_2 computes 4 blocks in 160 seconds
- P_1 computes two slowly and needs buffers to store 20 blocks!

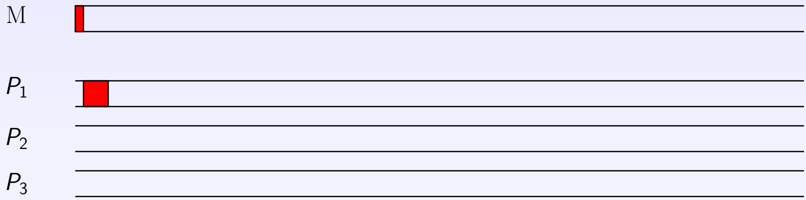
$$\underbrace{11111111111111111111}_{P_1}\ \underbrace{20}_{P_2}\ \underbrace{11111111111111111111}_{P_1}\ \underbrace{20}_{P_2}\ \underbrace{111111111\ldots}_{P_1\ldots}$$

Greedy heuristic for heterogeneous platforms



	P_1	P_2	P_3
c_i	2	3	5
w_i	2	3	1
μ_i	6	18	10
$2\mu_i c_i$	24	108	100
$\mu_i^2 w_i$	72	972	100
$\frac{2c_i}{\mu_i}$	$\frac{2}{3}$	$\frac{1}{3}$	1
$\frac{2c_i}{\mu_i w_i}$	$\frac{1}{3}$	$\frac{1}{9}$	$1 \longrightarrow \frac{5}{9}$

Greedy heuristic for heterogeneous platforms

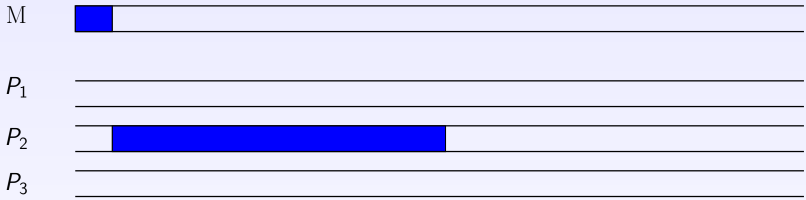


If first communication to P_1,

$$ratio = \frac{\mu_1^2}{2\mu_1 c_1} = \frac{36}{24} = \frac{3}{2}$$

Ratios: P_1 : 1.5

Greedy heuristic for heterogeneous platforms

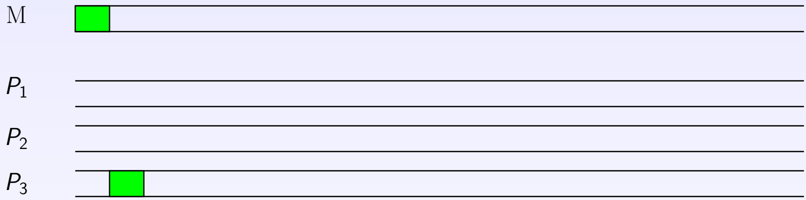


If first communication to P_2,

$$ratio = \frac{\mu_2^2}{2\mu_2 c_2} = \frac{324}{108} = 3$$

Ratios: P_1 : 1.5 P_2 : 3

Greedy heuristic for heterogeneous platforms

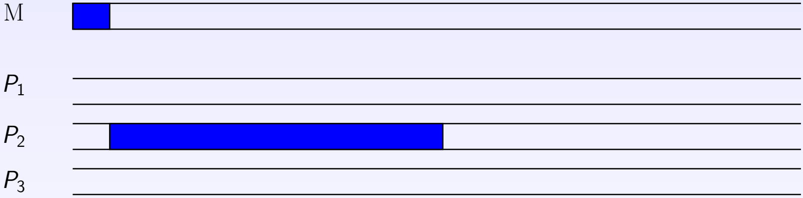


If first communication to P_3,

$$ratio = \frac{\mu_3^2}{2\mu_3 c_3} = \frac{100}{100} = 1$$

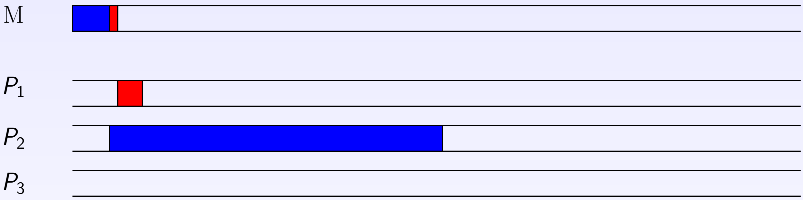
Ratios: P_1 : 1.5 P_2 : 3 P_3 : 1

Greedy heuristic for heterogeneous platforms



Best solution : first communication to P_2

Greedy heuristic for heterogeneous platforms



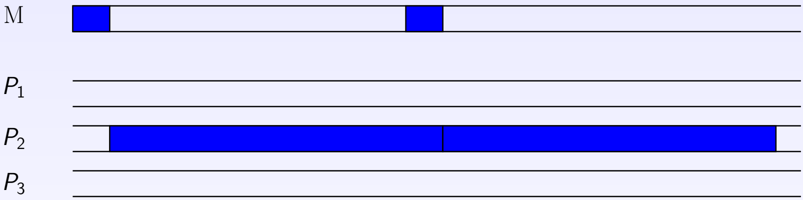
Two policy: Local

If second communication to P_1,

$$\text{ratio} = \frac{\mu_1^2}{2\mu_1 c_1} = \frac{36}{24} = \frac{3}{2}$$

Ratios: P_1 : 1.5

Greedy heuristic for heterogeneous platforms



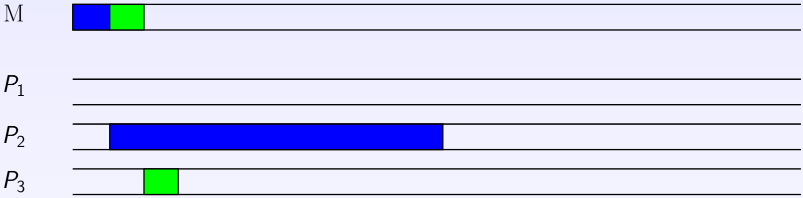
Two policy: Local

If second communication to P_2,

$$ratio = \frac{\mu_2^2}{2\mu_2 c_2 + \max\{\mu_2{}^2 w_2, 2\mu_2 c_2\}} = \frac{324}{1080} = 0.30$$

Ratios: P_1 : 1.5 P_2 : 0.30

Greedy heuristic for heterogeneous platforms



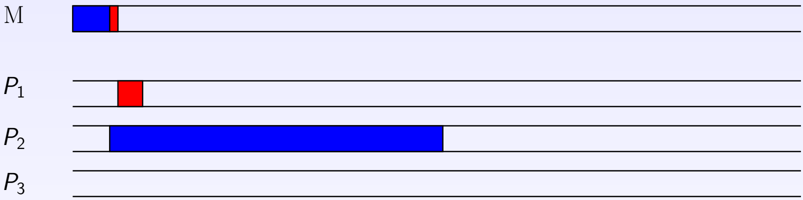
Two policy: Local

If second communication to P_3,

$$ratio = \frac{\mu_3^2}{2\mu_3 c_3} = \frac{100}{100} = 1$$

Ratios: P_1 : 1.5 P_2 : 0.30 P_3 : 1

Greedy heuristic for heterogeneous platforms



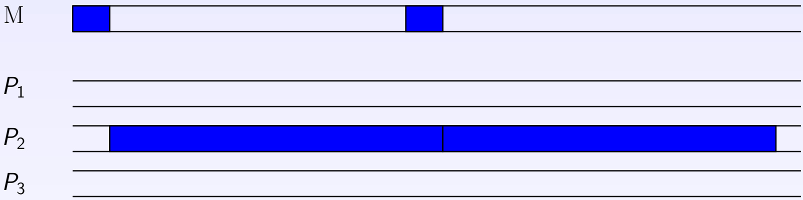
Two policy: Global

If second communication to P_1,

$$ratio = \frac{\mu_2^2 + \mu_1^2}{2\mu_2 c_2 + 2\mu_1 c_1} = \frac{324 + 36}{108 + 24} = 2.71$$

Ratios: P_1 : 2.71

Greedy heuristic for heterogeneous platforms



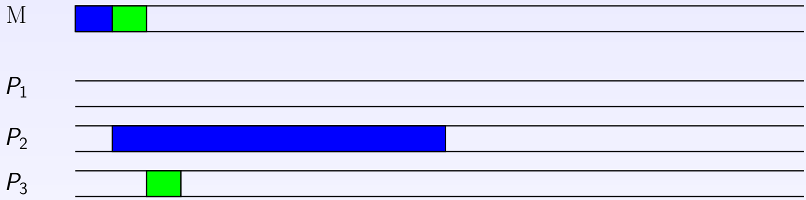
Two policy: Global

If second communication to P_2,

$$ratio = \frac{\mu_2^2 + \mu_2^2}{2\mu_2 c_2 + 2\mu_2 c_2} = \frac{324 + 324}{1080} = 0.60$$

Ratios: P_1 : 2.71 P_2 : 0.60

Greedy heuristic for heterogeneous platforms



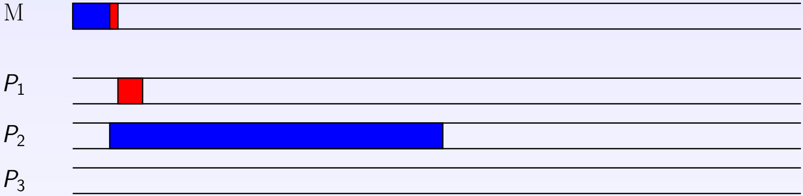
Two policy: Global

If second communication to P_3,

$$ratio = \frac{\mu_2^2 + \mu_3^2}{2\mu_2 c_2 + 2\mu_3 c_3} = \frac{324 + 100}{108 + 100} = 2.04$$

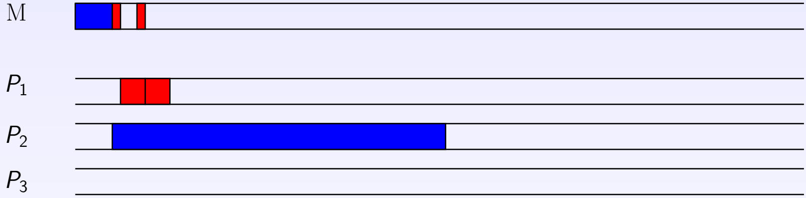
Ratios: P_1 : 2.71 P_2 : 0.60 P_3 : 2.04

Greedy heuristic for heterogeneous platforms



Best solution : second communication to P_1

Greedy heuristic for heterogeneous platforms

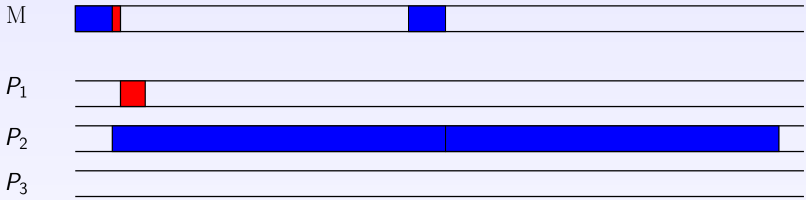


If third communication to P_1,

$$ratio = \frac{\mu_2^2 + \mu_1^2 + \mu_1^2}{t_{\text{comm}}} = \frac{360 + 36}{168} = 2.36$$

Ratios: P_1 : 1.93

Greedy heuristic for heterogeneous platforms

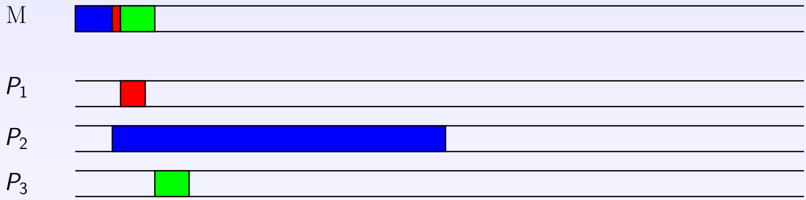


If third communication to P_2,

$$ratio = \frac{\mu_2^2 + \mu_1^2 + \mu_2^2}{t_{\text{comm}}} = \frac{360 + 324}{1080} = 0.63$$

Ratios: P_1 : 1.93 P_2 : 0.63

Greedy heuristic for heterogeneous platforms



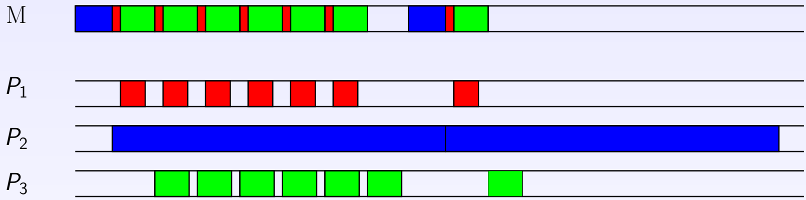
If third communication to P_3,

$$ratio = \frac{\mu_2^2 + \mu_1^2 + \mu_3^2}{2\mu_2 c_2 + 2\mu_1 c_1 + 2\mu_3 c_3} = \frac{360 + 100}{132 + 100} = 1.97$$

Ratios: $P_1 : 1.93$ $P_2 : 0.63$ $P_3 : 1.97$

Best solution: third communication to P_3

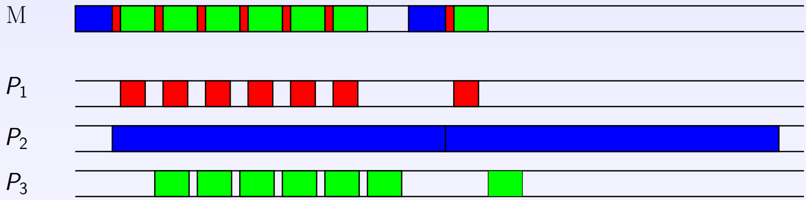
Greedy heuristic for heterogeneous platforms



Asymptotic ratio: 1.17 (*divisible* throughput 1.39)
Allocated bandwidths: 14.8%, 11.2%, and 61.7% (instead of 33.3%, 11.1%, and 55.6%)

Two-block look-ahead greedy
Asymptotic ratio: 1.30 (*divisible* throughput 1.39)
Allocated bandwidths: 17.2%, 11.1%, and 71.7%

Greedy heuristic for heterogeneous platforms



Asymptotic ratio: 1.17 (*divisible* throughput 1.39)
Allocated bandwidths: 14.8%, 11.2%, and 61.7% (instead of 33.3%, 11.1%, and 55.6%)

Two-block look-ahead greedy
Asymptotic ratio: 1.30 (*divisible* throughput 1.39)
Allocated bandwidths: 17.2%, 11.1%, and 71.7%

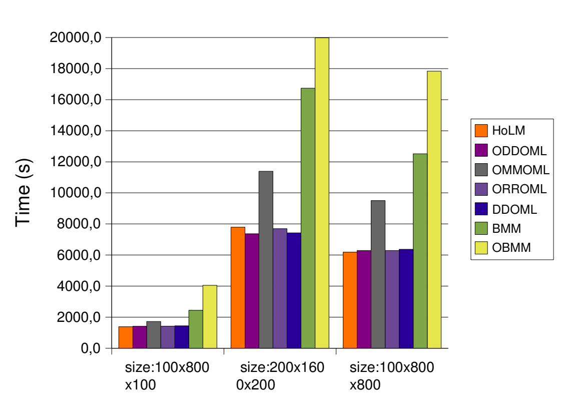
Outline

1. Framework
2. Theoretical study
   - The simplest problem
   - Limited memory
3. Parallel algorithms
   - Homogeneous platforms
   - Heterogeneous platforms
4. Experiments
5. Conclusion

The studied algorithms

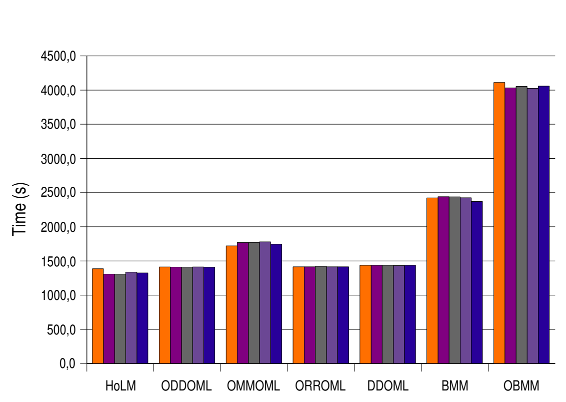
- Homogeneous algorithm
- Overlapped Round-Robin, Optimized Memory Layout (**ORROML**)
- Overlapped Min-Min, Optimized Memory Layout (**OMMOML**)
- Overlapped Demand-Driven, Optimized Memory Layout (**ODDOML**)
- Demand-Driven, Optimized Memory Layout (**DDOML**)
- Block Matrix Multiply (**BMM**)
- Overlapped Block Matrix Multiply (**OBMM**)

Results



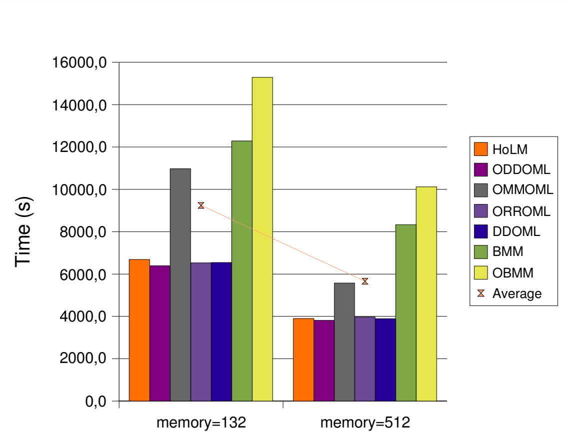
Performance of the algorithms on different matrices.

Results



Variation of algorithm execution times.

Results



Impact of memory size on algorithm performance.

Outline

1. Framework
2. Theoretical study
   - The simplest problem
   - Limited memory
3. Parallel algorithms
   - Homogeneous platforms
   - Heterogeneous platforms
4. Experiments
5. Conclusion

Conclusion

- Key points:
  - Realistic platform model
  - Lower bound on total number of communications
  - Design of efficient parallel algorithms
- Extensions:
  - Improve lower bound to match algorithm performance
  - Run heterogeneous experiments
  - Investigate LU/Cholesky