# Design and Implementation of Parallel Algorithms for Modern Heterogeneous Platforms Based on Functional Performance Models

David Clarke

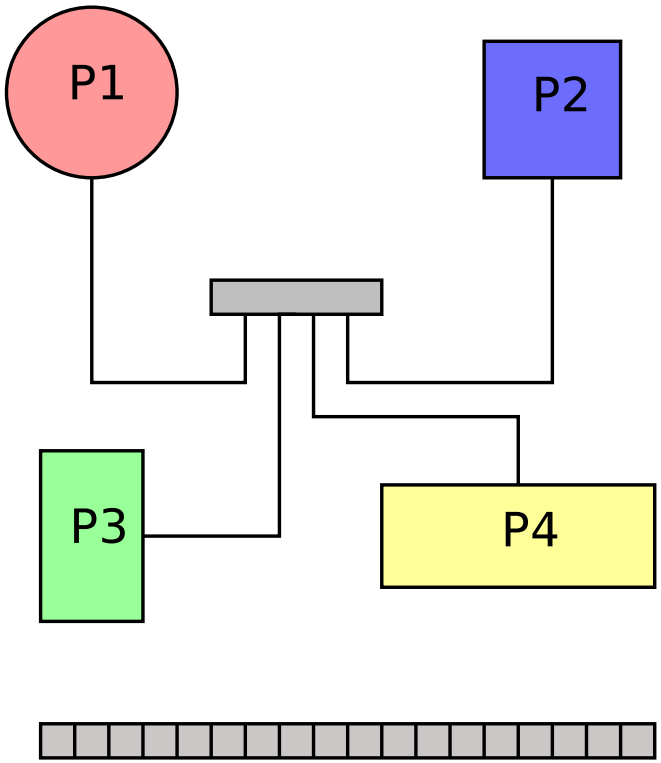
Heterogeneous Computing Laboratory, UCD, Dublin

May 11, 2012



# Platform and Application

## Generalised Heterogeneous Platform

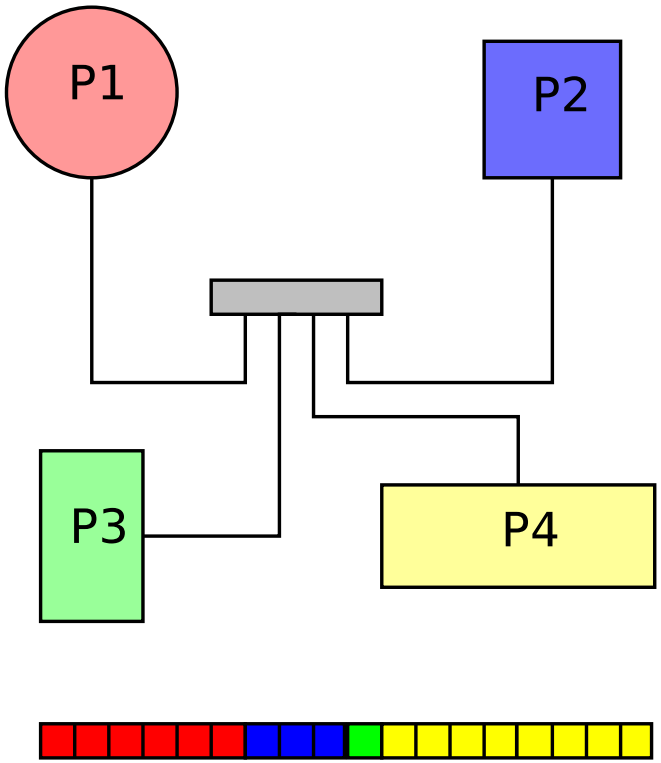



## Data Parallel Application

```
...

while(...) {

  compute_parallel(data, size);

  syncronise (data);

}

...
```

# Platform and Application

## Generalised Heterogeneous Platform

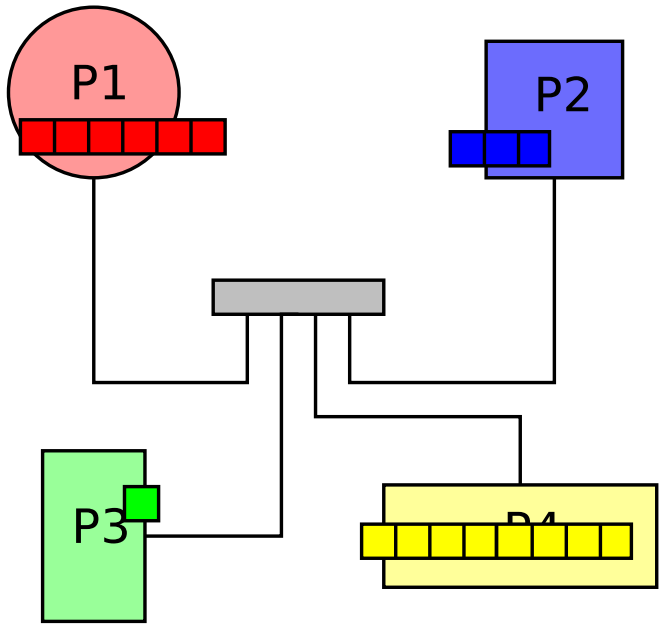



## Data Parallel Application

```
...

while(...) {

  compute_parallel(data, size);

  syncronise (data);

}

...
```

# Platform and Application

## Generalised Heterogeneous Platform

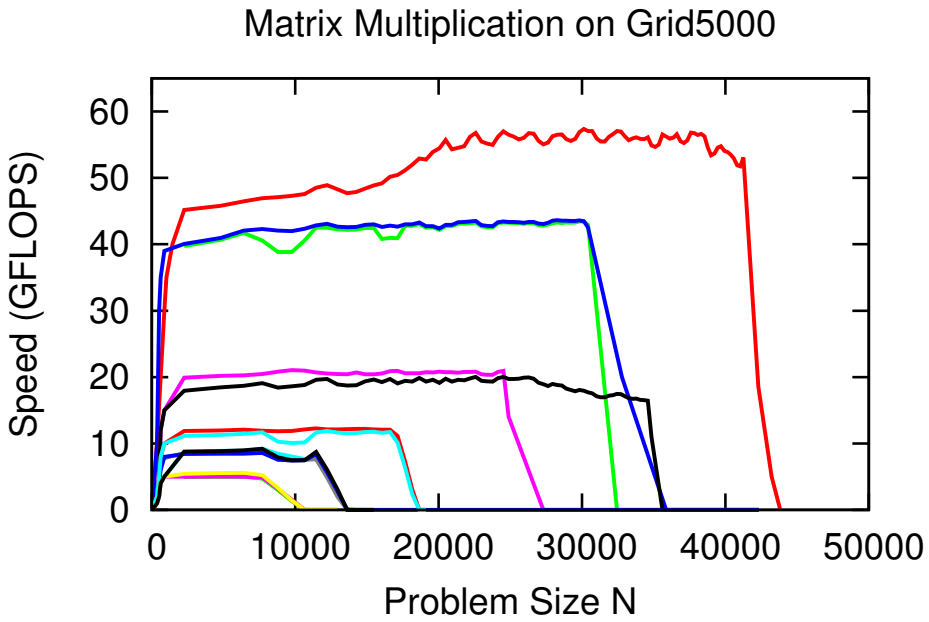



## Data Parallel Application

```
...

while(...) {

  compute_parallel(data, size);

  syncronise (data);

}

...
```

# Traditional load Balancing

- Traditionally, processor performance is defined by a constant number.
- Computational units are partitioned as $d_i = N \times \frac{s_i}{\sum_{j=1}^{p} s_j}$.
- In reality, speed is a function of problem size.



## Load Balanced when:

$$t_1(d_1) \approx t_2(d_2) \approx \ldots \approx t_p(d_p)$$

$$\frac{d_1}{s_1(d_1)} \approx \frac{d_2}{s_2(d_2)} \approx \ldots \approx \frac{d_p}{s_p(d_p)}$$

$$d_1 + d_2 + \ldots + d_p = N$$

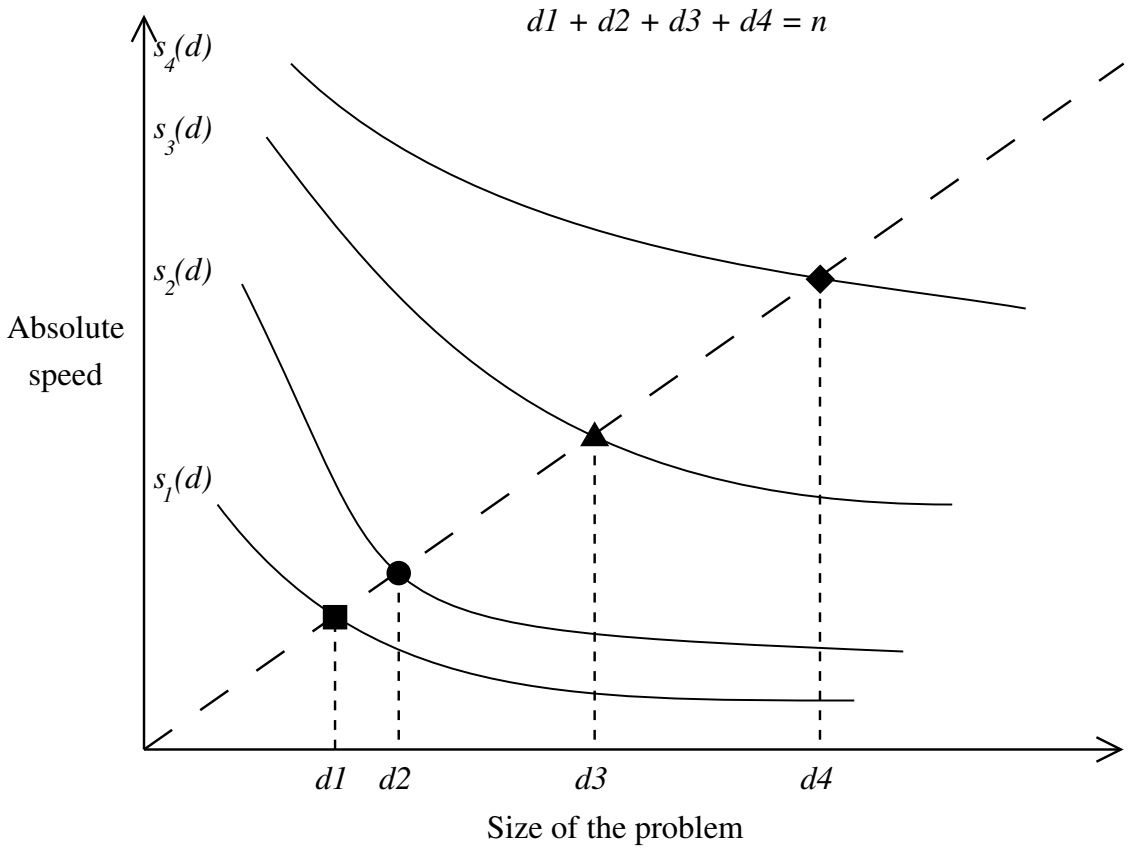
Developed two algorithms:

- **Geometric Partitioning Algorithm**
  - Convergence guaranteed
  - Restriction on shape of speed functions
- **Numerical Partitioning Algorithm**
  - No restriction on shape
  - No guarantee of convergence
    (in practice converges for realistic functions)

Input: FPMs and $N$. Output: $d_1, d_2, \ldots, d_p$.

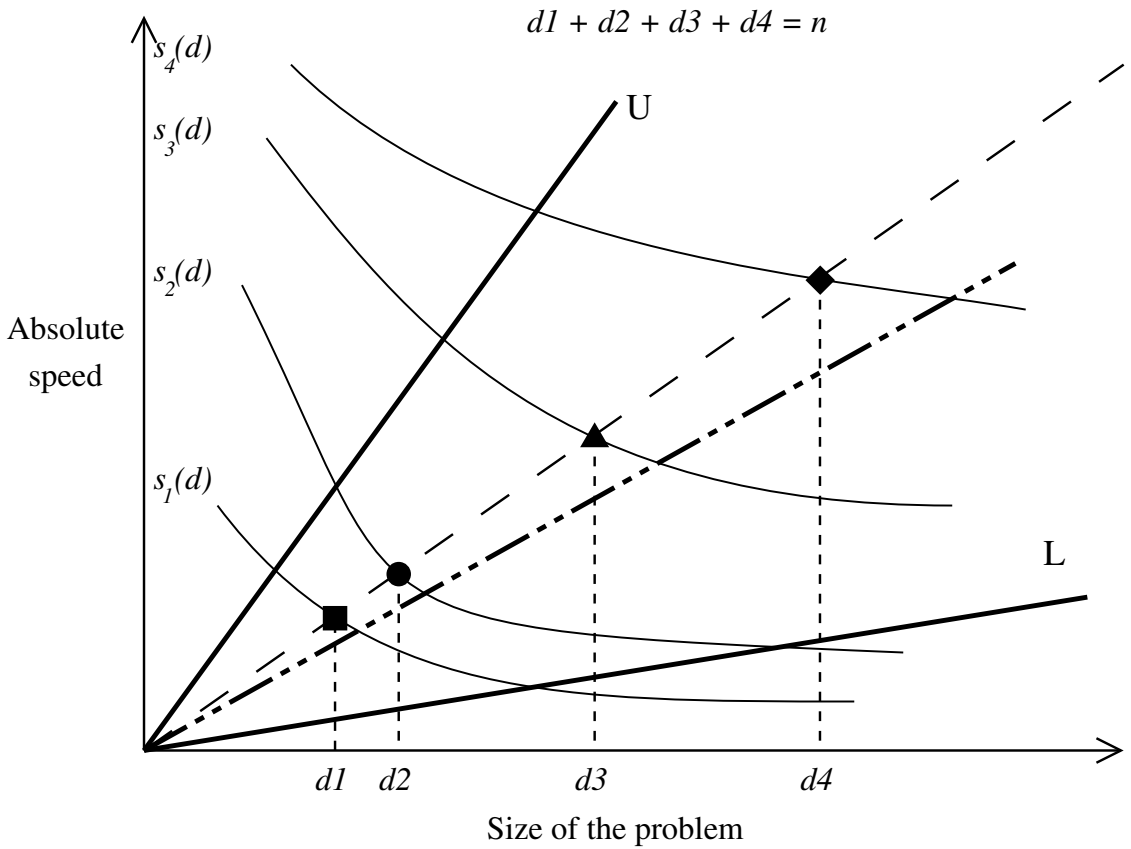
# Geometric Partitioning Algorithm

- Points $\left(d_i, s_i(d_i)\right)$ lie on a line passing through the origin when $\frac{d_i}{s_i(d_i)} = constant$.
- Value of $N$ determines the slope.
- Algorithm iteratively bisects solution space to find values $d_i$.



# Geometric Partitioning Algorithm

- Points $\left(d_i, s_i(d_i)\right)$ lie on a line passing through the origin when $\frac{d_i}{s_i(d_i)} = constant$.
- Value of $N$ determines the slope.
- Algorithm iteratively bisects solution space to find values $d_i$.



# Numerical Partitioning Algorithm

Optimal partitioning can be formulated as a system of nonlinear equations, $F(x) = 0$

$$F(x) = \begin{cases} n - \sum_{i=1}^{p} x_i \\ \frac{x_i}{s_i(x_i)} - \frac{x_1}{s_1(x_1)}, \quad 2 \leq i \leq p \end{cases} \tag{1}$$

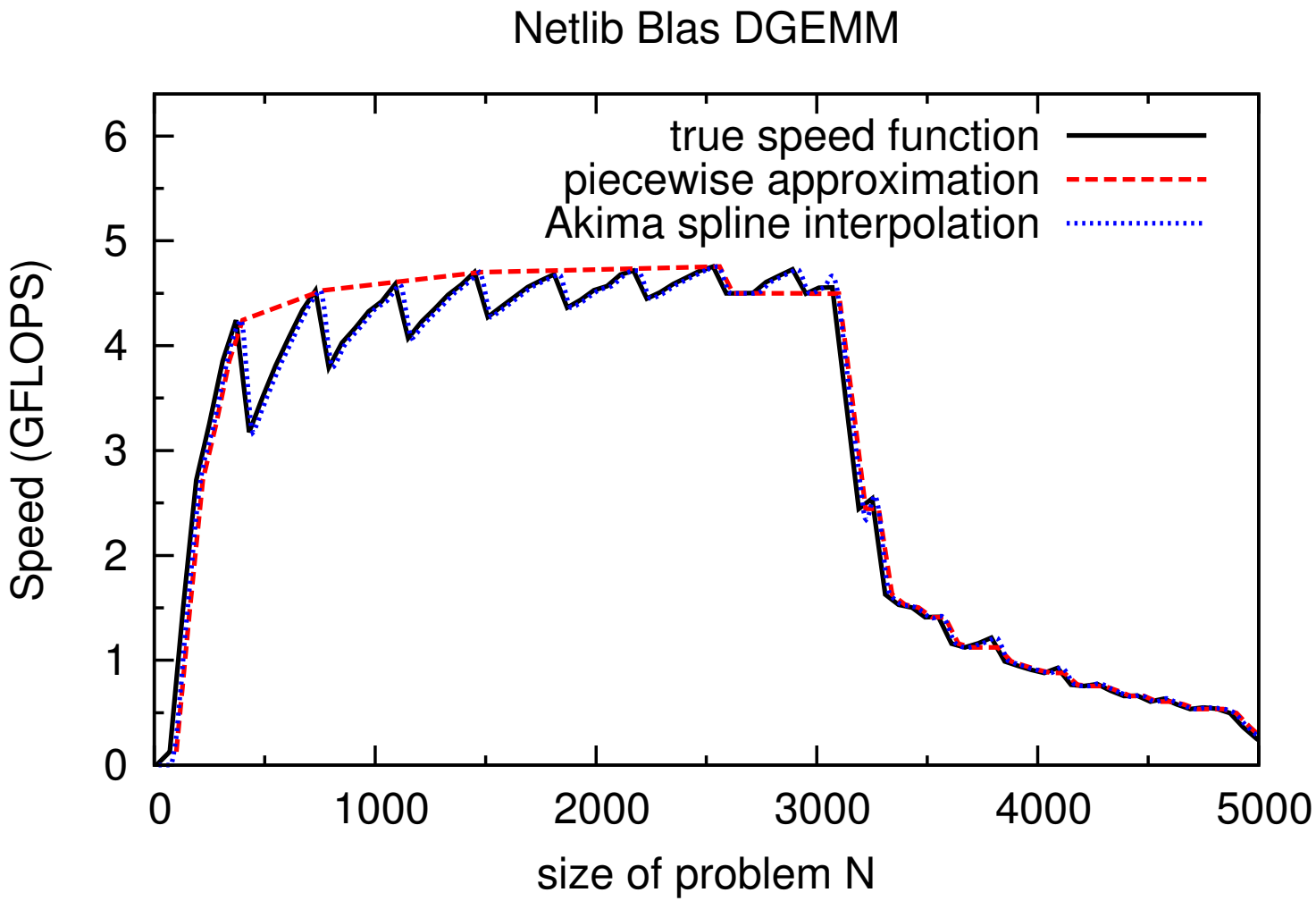
Powell's Hybrid method used to solve:

$$x_{k+1} = x_k - J(x_k)F(x_k) \tag{2}$$

With Jacobian matrix:

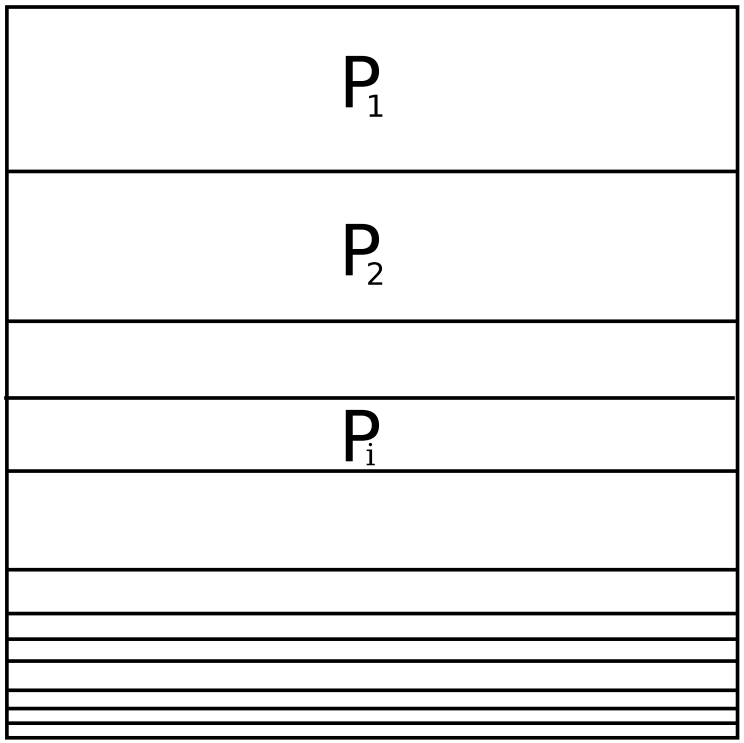
$$J(x) = \begin{pmatrix} -1 & -1 & \dots & -1 \\ -\frac{s_1(x_1) - x_1 s_1'(x_1)}{s_1^2(x_1)} & \frac{s_2(x_2) - x_2 s_2'(x_2)}{s_2^2(x_2)} & 0 & 0 \\ \dots & 0 & \dots & 0 \\ -\frac{s_1(x_1) - x_1 s_1'(x_1)}{s_1^2(x_1)} & 0 & 0 & \frac{s_p(x_p) - x_p s_p'(x_p)}{s_p^2(x_p)} \end{pmatrix} \tag{3}$$

# Fitting Continuous Functions to Discreet Data

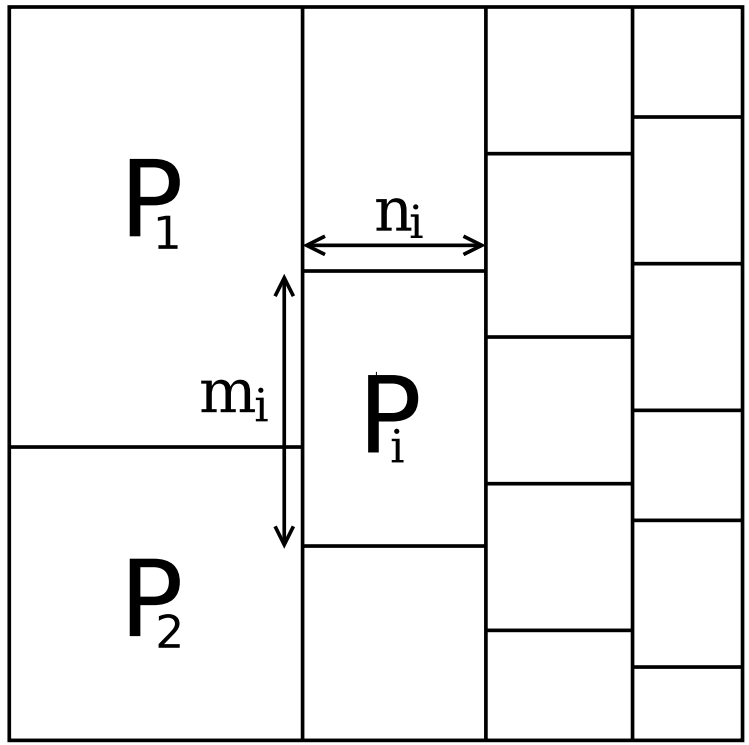


# Case study: Matrix Partitioning based on FPMs

## Simple Partitioning

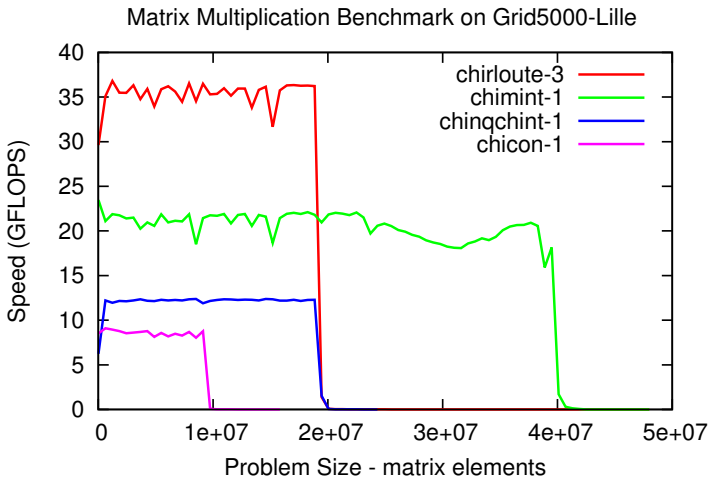


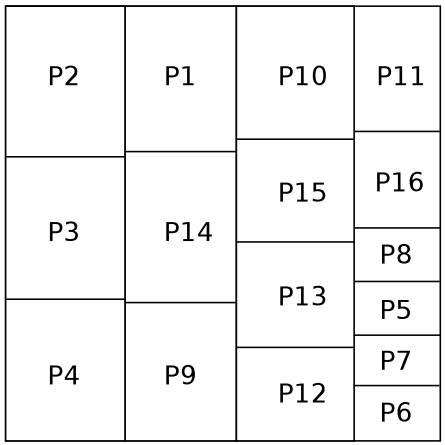

## 2D Partitioning



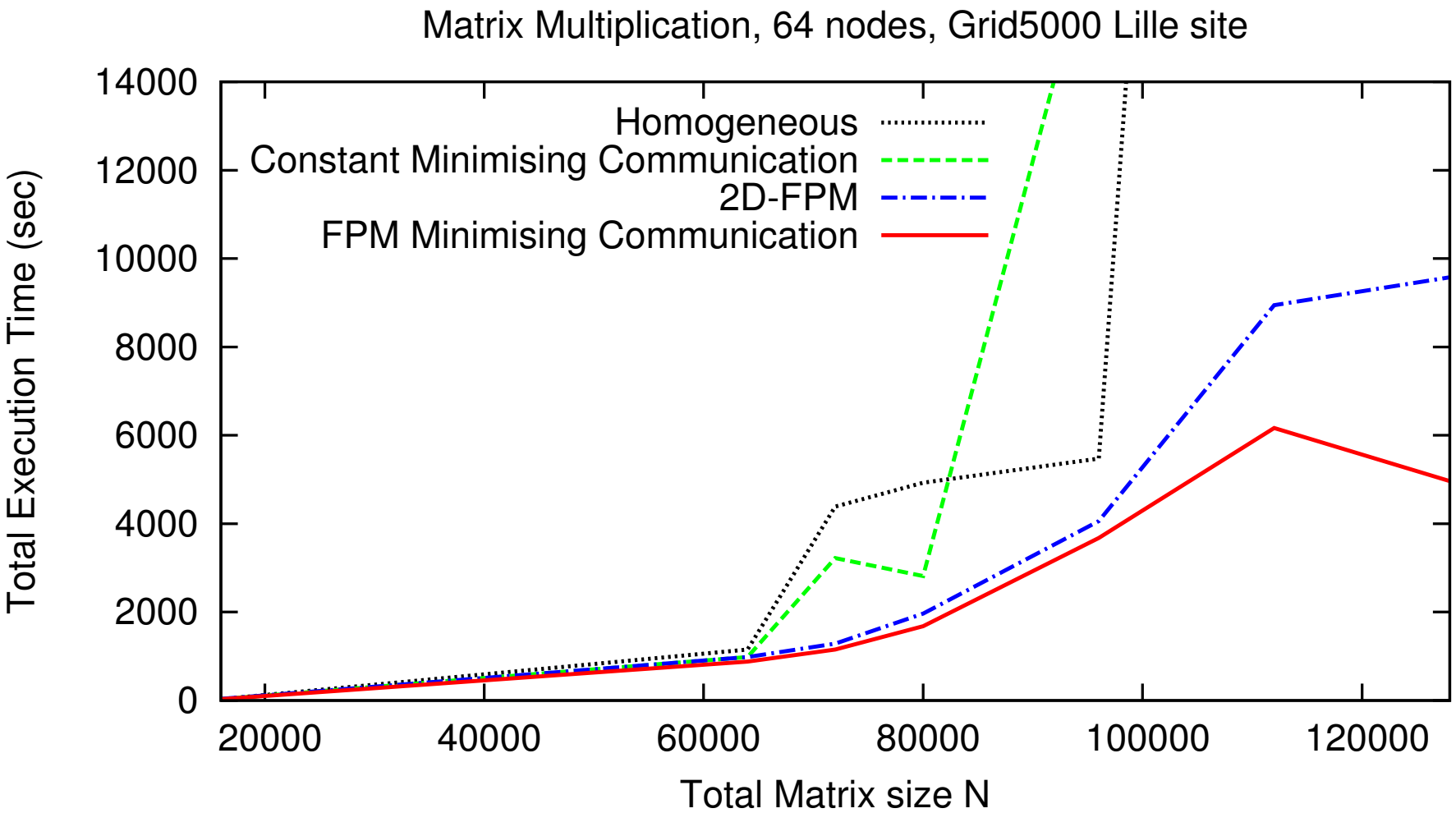
# Two-Dimensional Matrix Partitioning with 1D FPMs

- Height and width combined into one parameter, area $d_i = m_i \times n_i$.
- Square areas are benchmarked $m = n = \sqrt{d}$.
- Partition with 1D FPM algorithm to find area of rectangles (geometric or numerical).
- Use communication volume minimising algorithm* to compute ordering and shape of these rectangles.



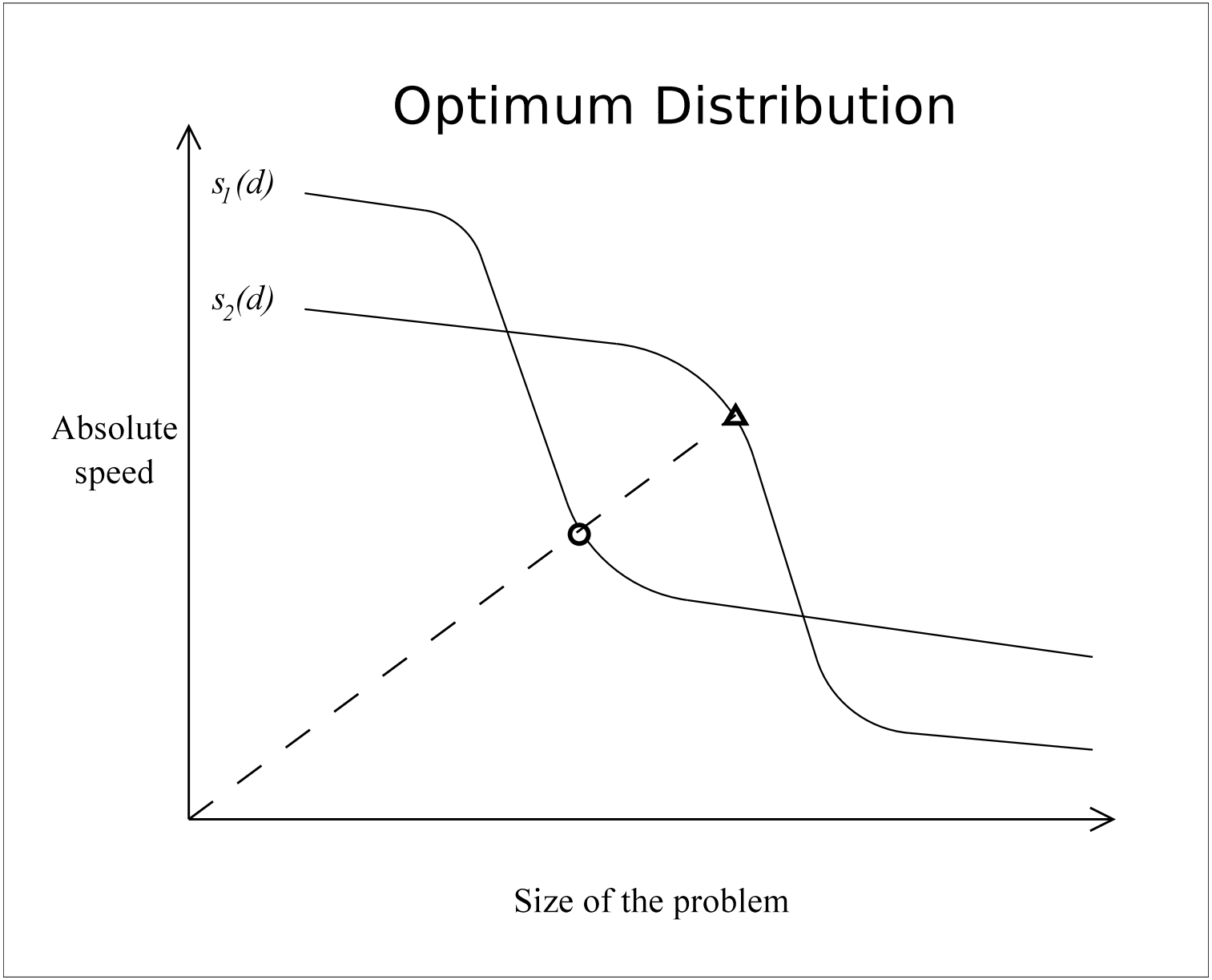



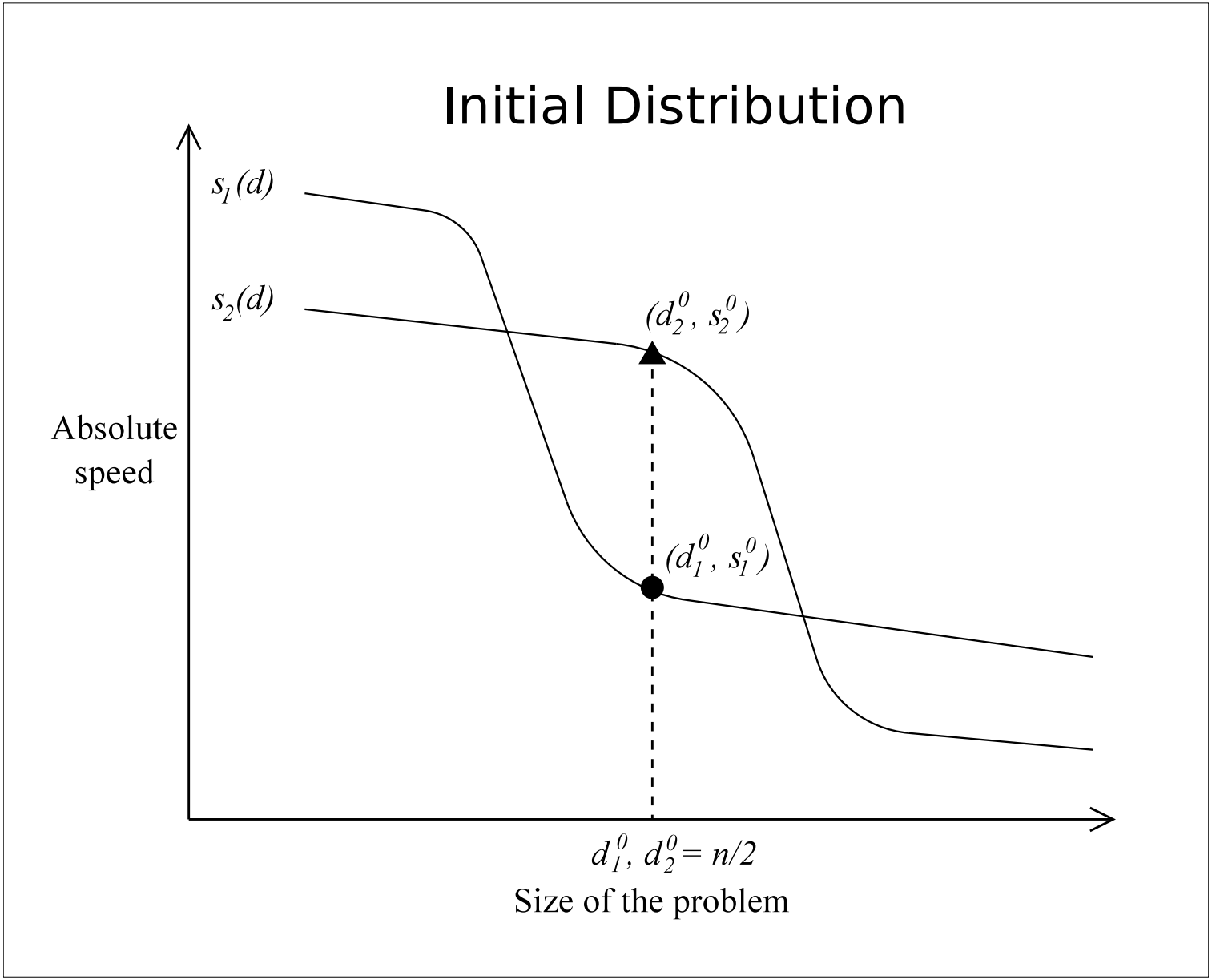

* Beaumont, O., Boudet, V., Rastello, F., Robert, Y.: Matrix Multiplication on Heterogeneous Platforms. 2001

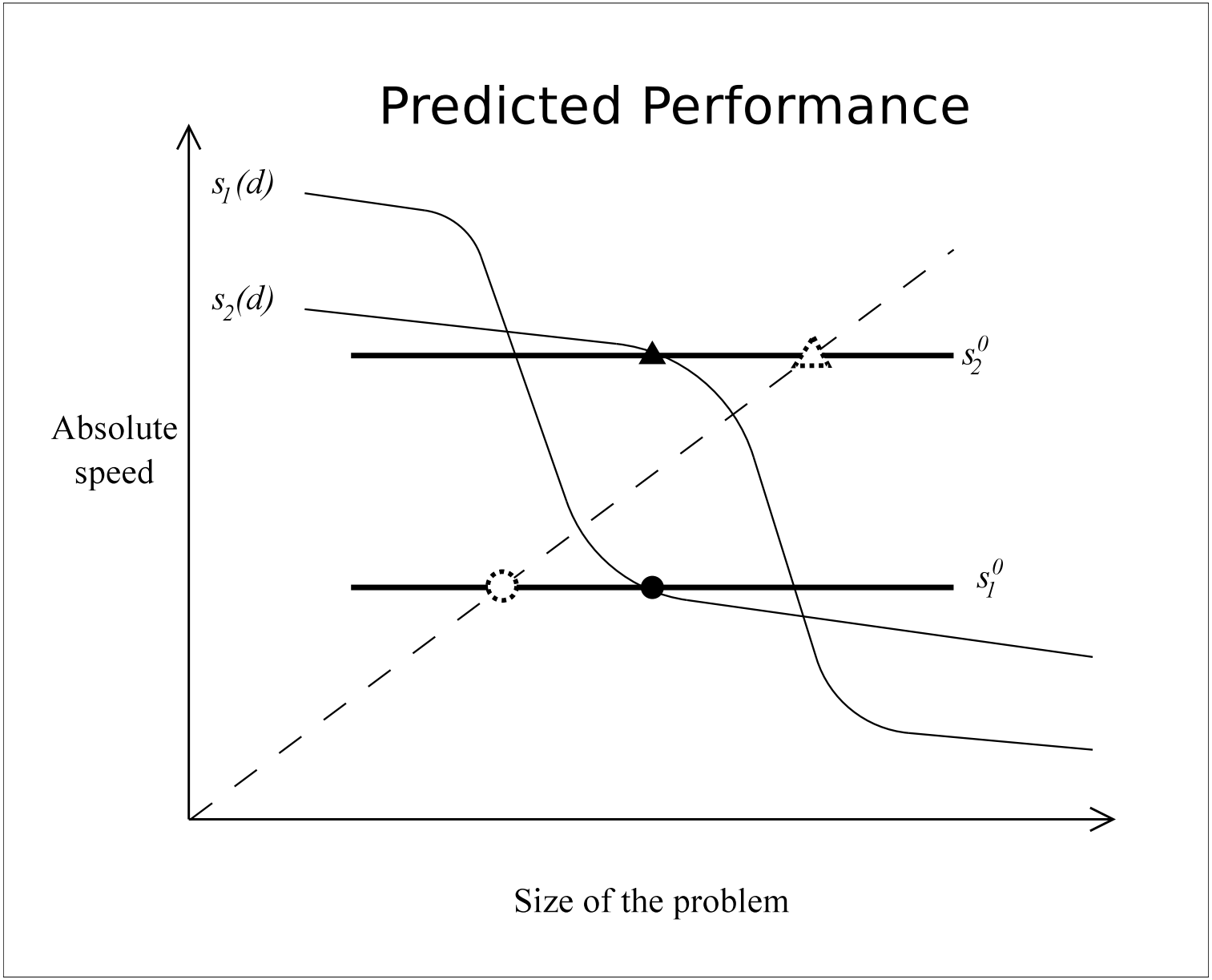

Matrix Multiplication, 64 nodes, Grid5000 Lille site
Total Execution Time (sec)
Total Matrix size N
Homogeneous
Constant Minimising Communication
2D-FPM
FPM Minimising Communication
0
2000
4000
6000
8000
10000
12000
14000
20000
40000
60000
80000
100000
120000

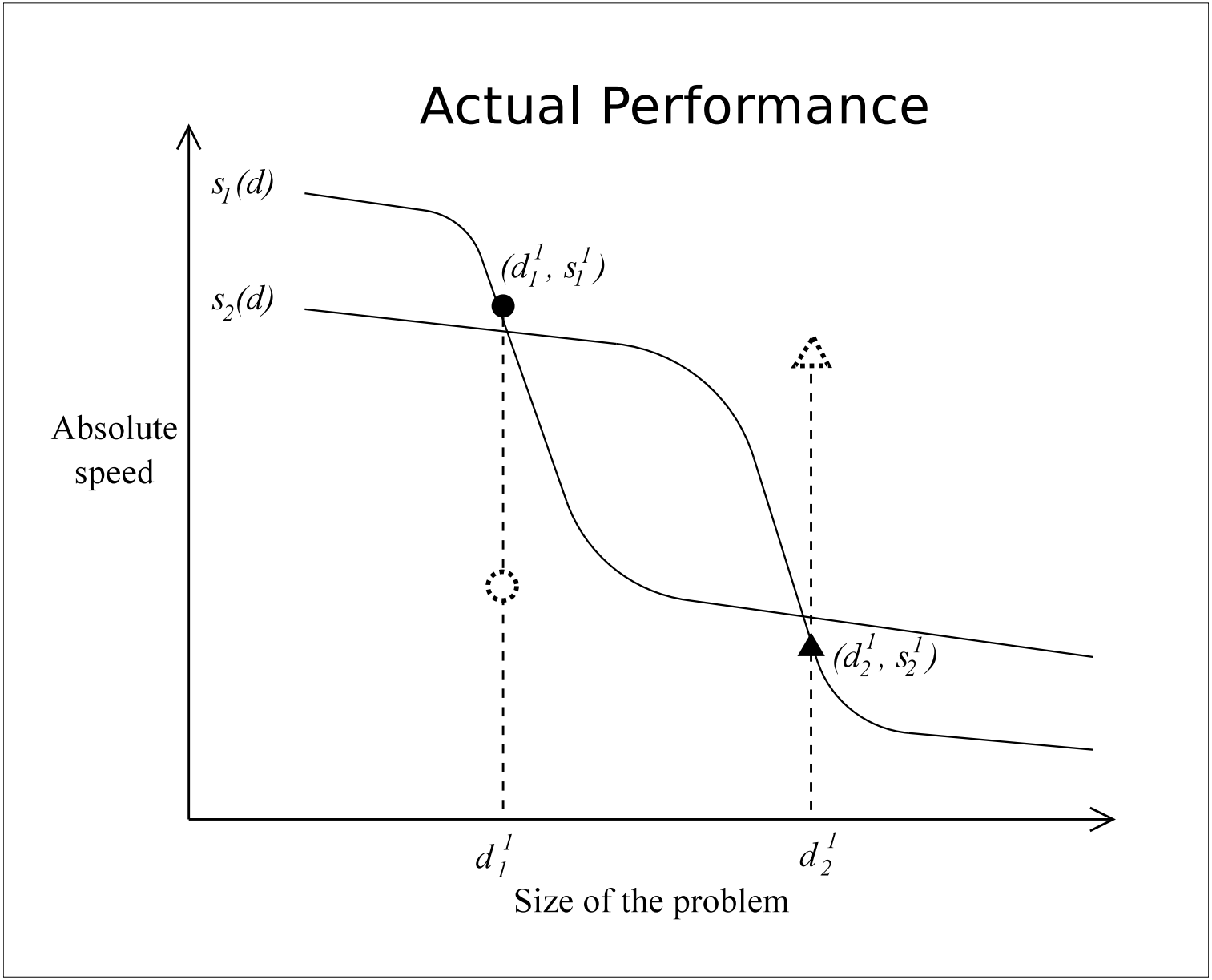
# Dynamically Built Models

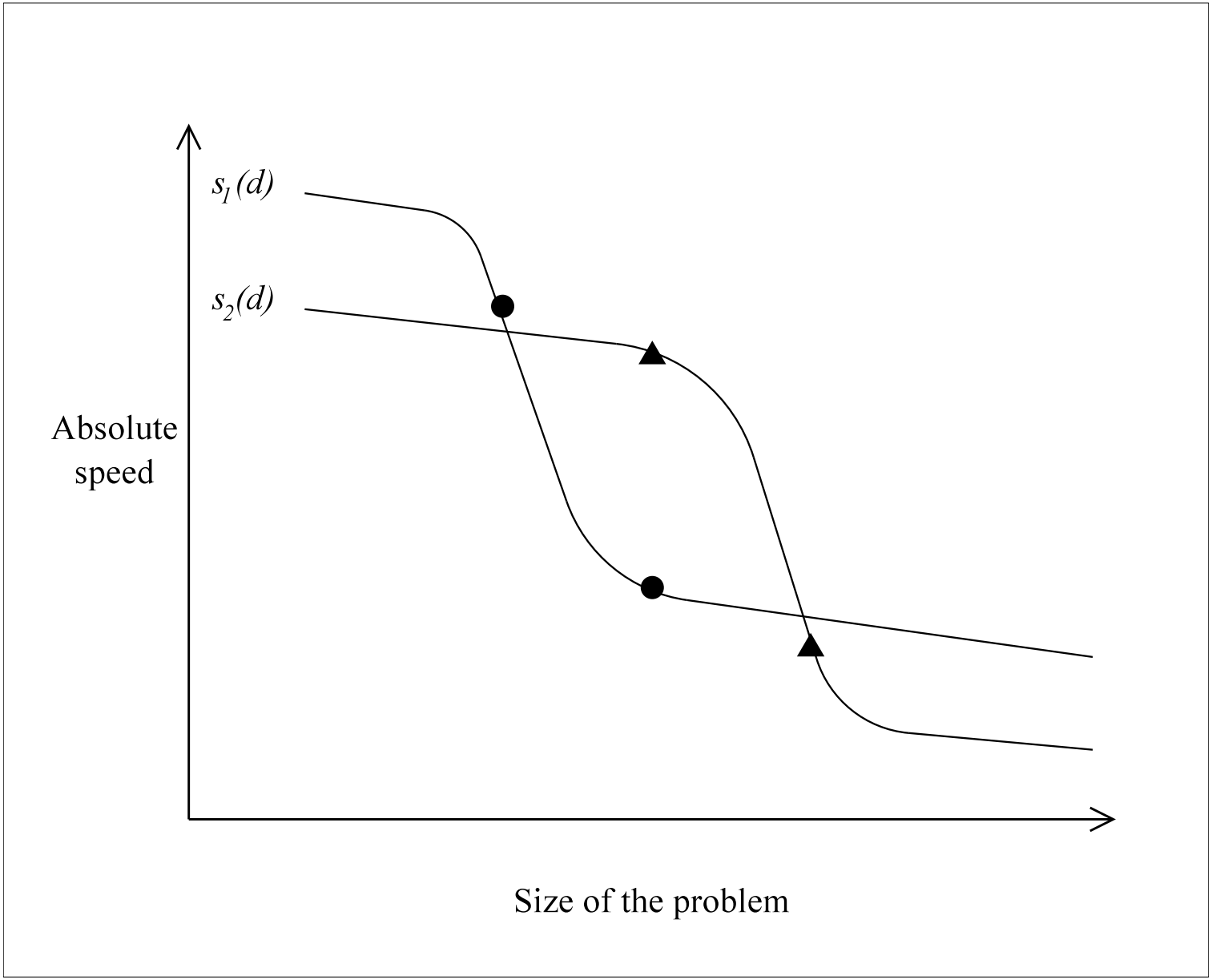
- Functional performance models are different for each application and each platform.
- Building these models for all conceivable problem sizes is computationally expensive.
- Building full models is not an option for a self adaptive algorithm.
- Present an algorithm that dynamically builds the models at relevant problem sizes

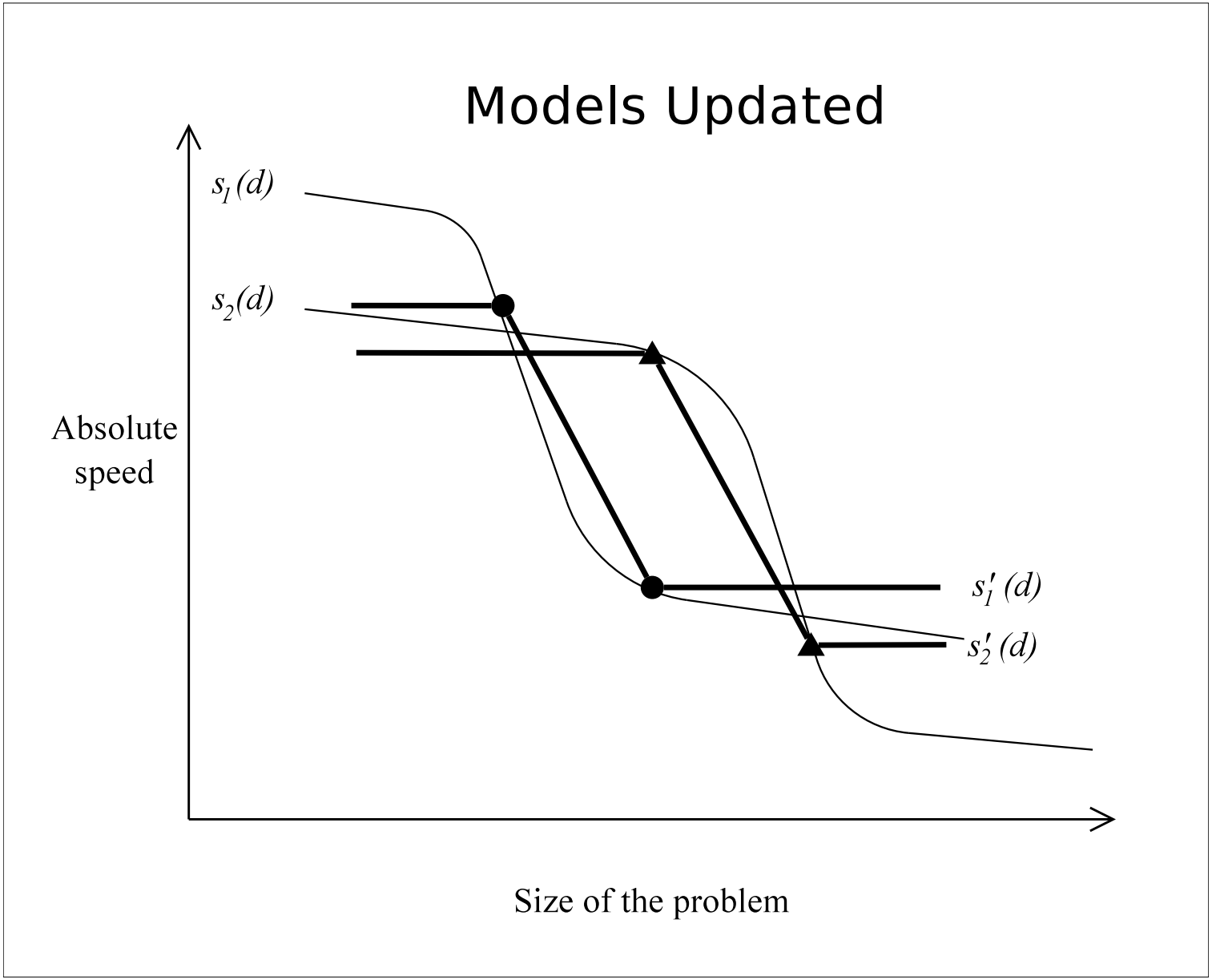

Optimum Distribution
$s_1(d)$
$s_2(d)$
Absolute speed
Size of the problem

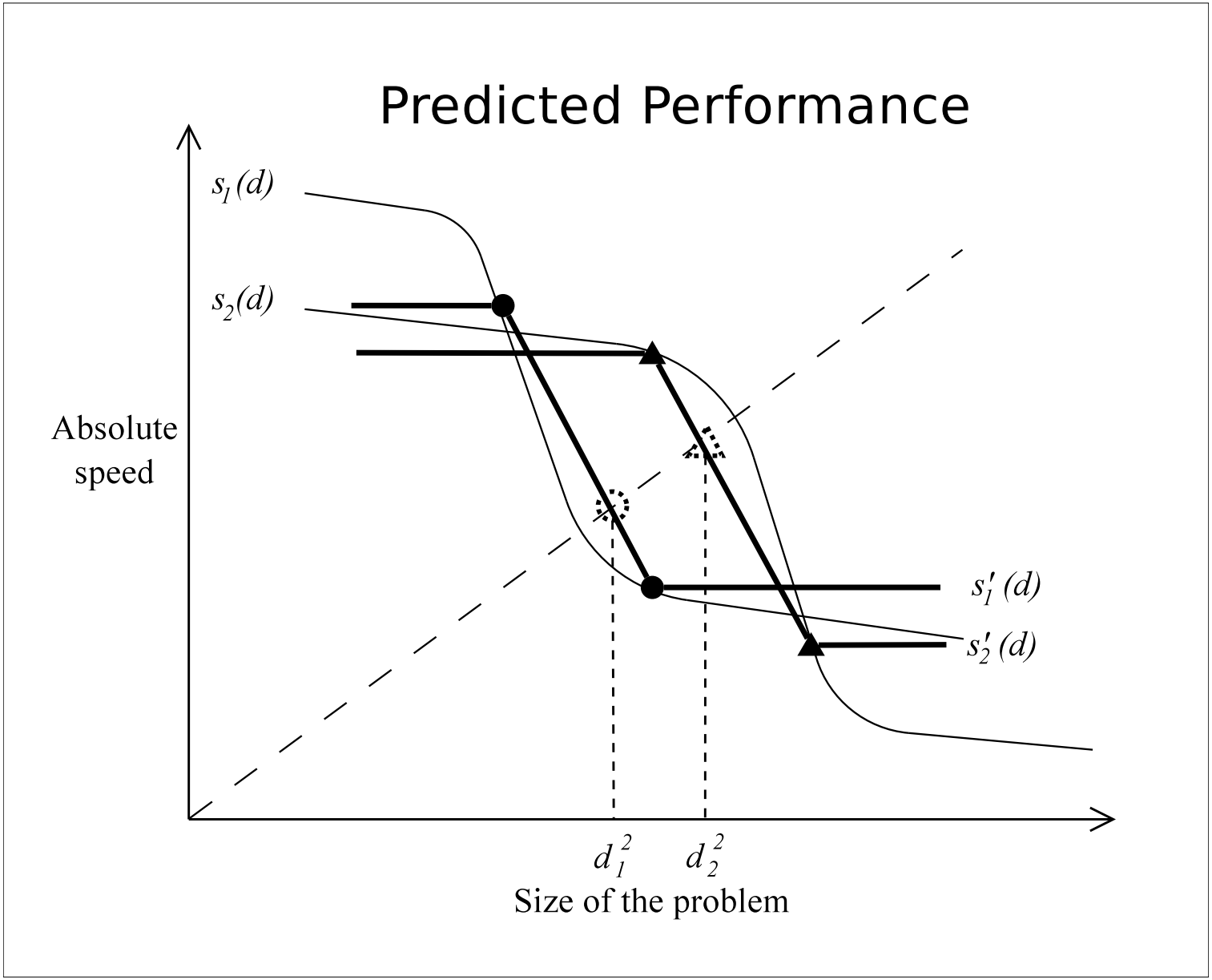

Initial Distribution
$s_1(d)$
$s_2(d)$
$(d_2^0, s_2^0)$
$(d_1^0, s_1^0)$
Absolute speed
$d_1^0, d_2^0 = n/2$
Size of the problem

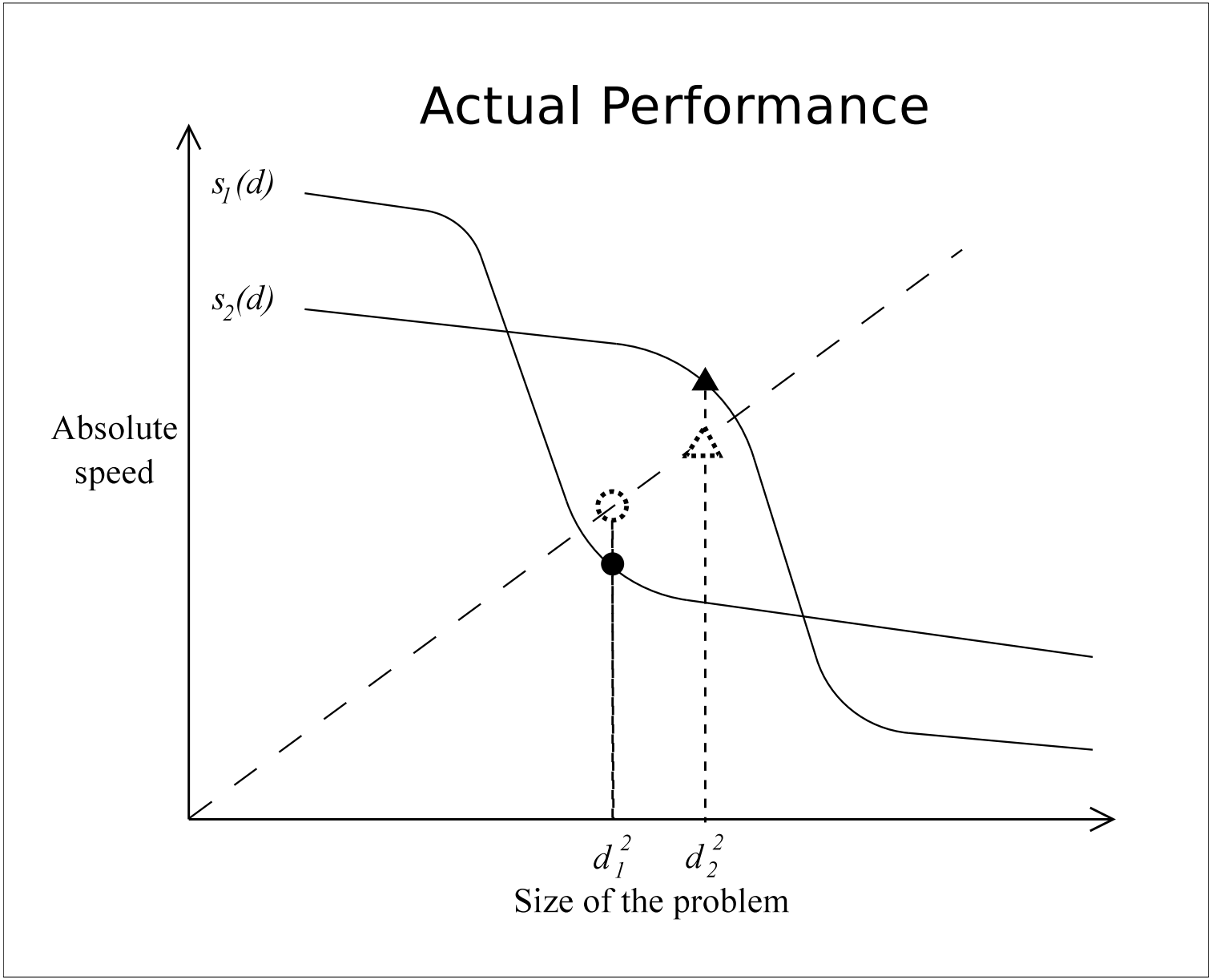

Predicted Performance
$s_1(d)$
$s_2(d)$
$s_2^0$
$s_1^0$
Absolute speed
Size of the problem

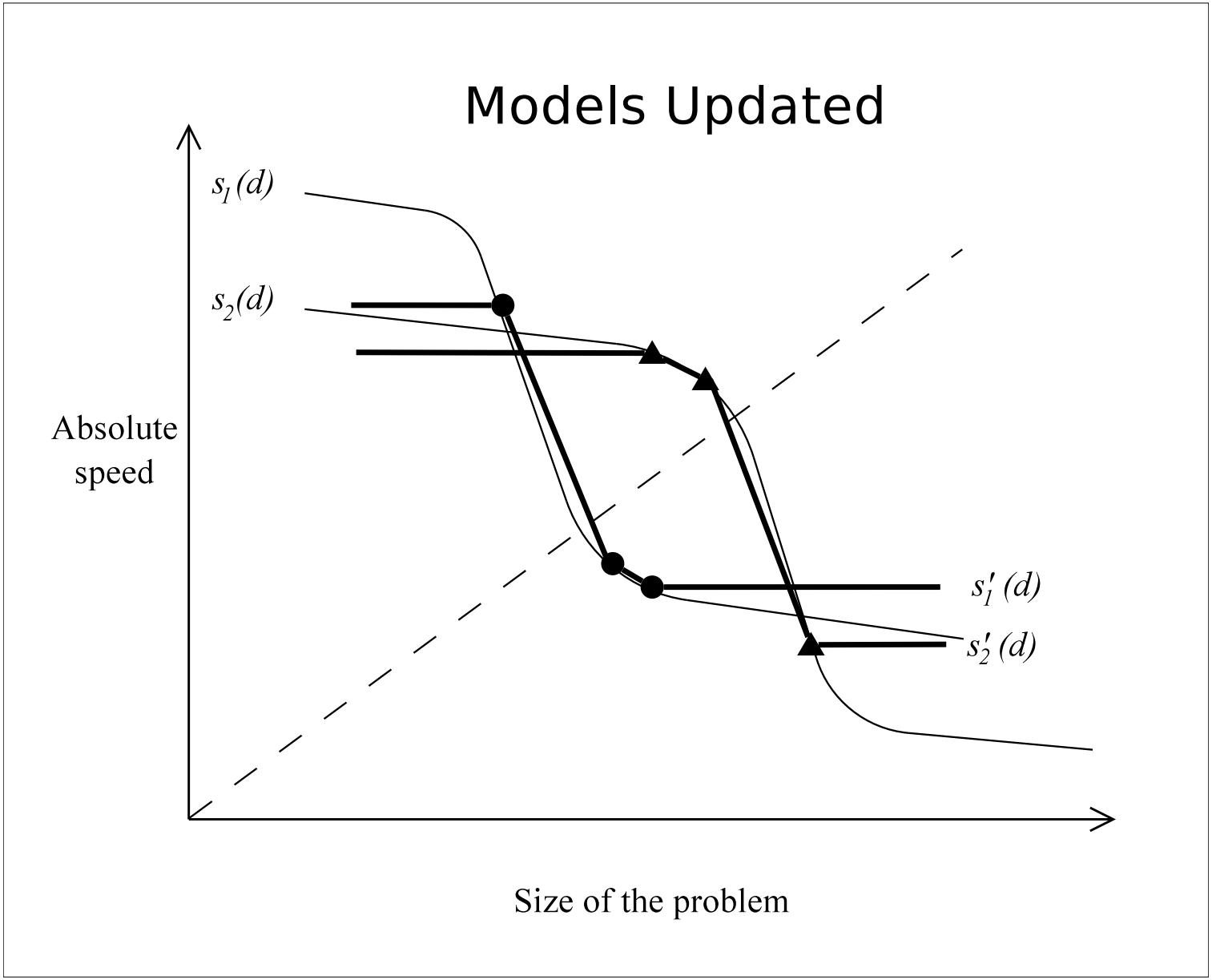

Actual Performance
$s_1(d)$
$s_2(d)$
$(d_1^1, s_1^1)$
$(d_2^1, s_2^1)$
Absolute speed
$d_1^1$
$d_2^1$
Size of the problem

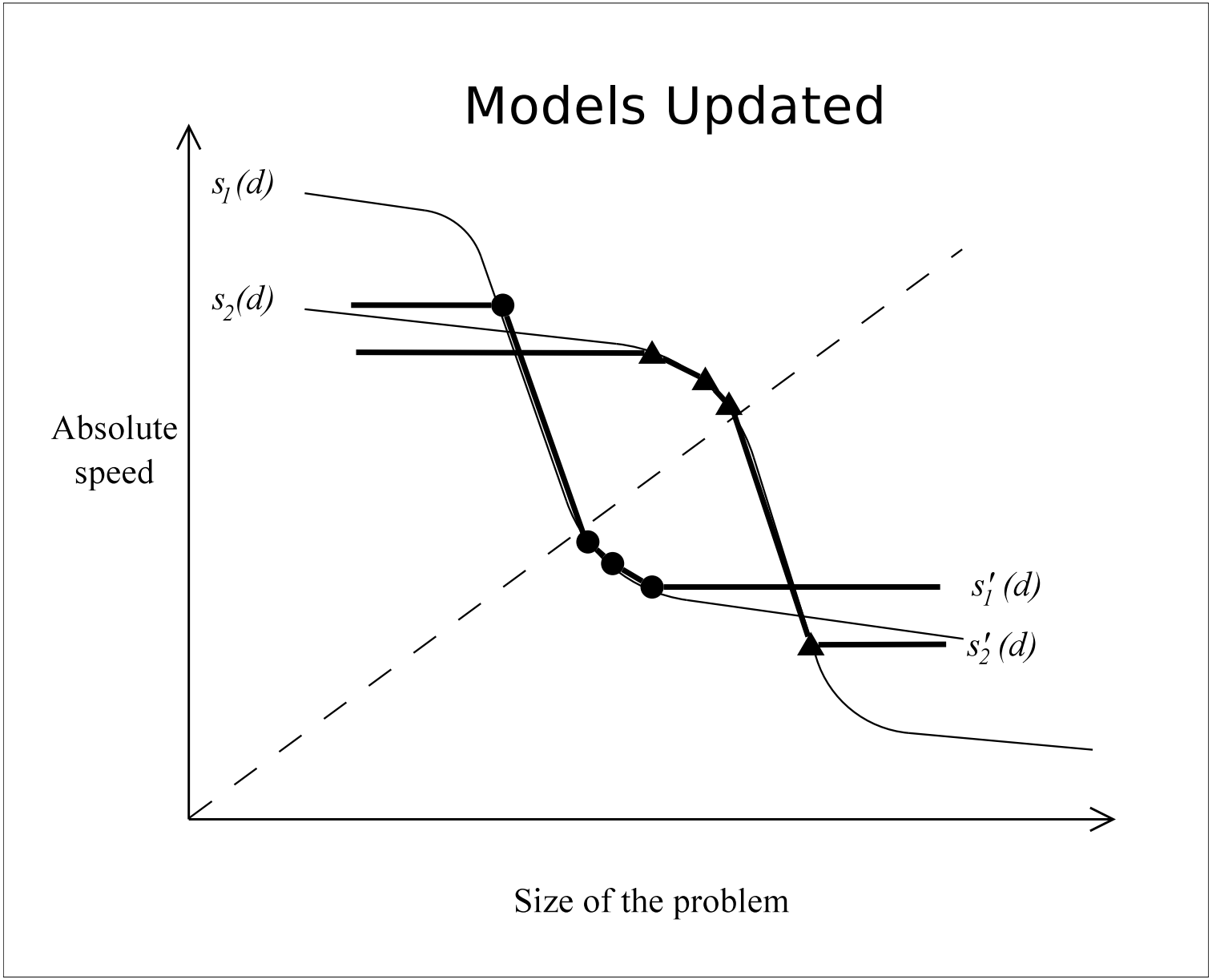

$s_1(d)$
$s_2(d)$
Absolute speed
Size of the problem


Models Updated
$s_1(d)$
$s_2(d)$
Absolute speed
$s_1'(d)$
$s_2'(d)$
Size of the problem

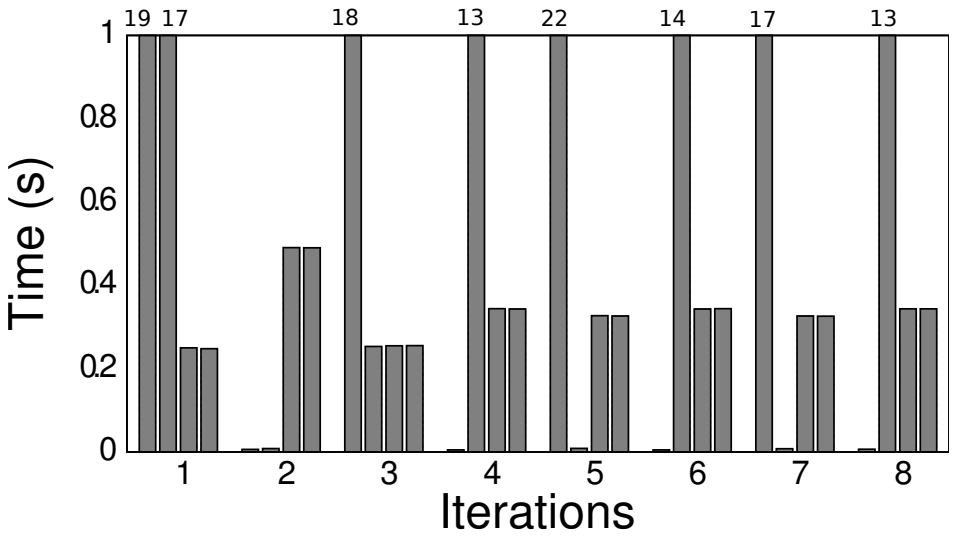

Predicted Performance
$s_1(d)$
$s_2(d)$
Absolute speed
$s_1'(d)$
$s_2'(d)$
$d_1^2$
$d_2^2$
Size of the problem

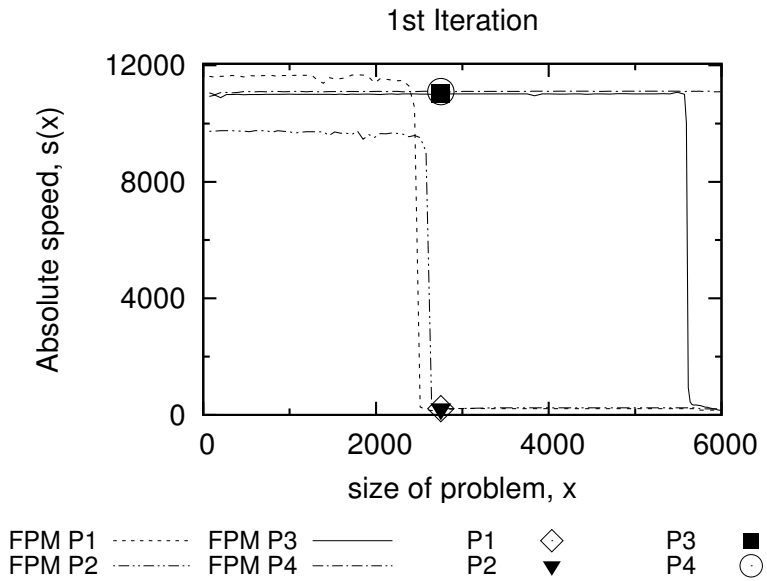

Actual Performance
$s_1(d)$
$s_2(d)$
Absolute speed
$d_1^2$
$d_2^2$
Size of the problem


Models Updated
$s_1(d)$
$s_2(d)$
Absolute speed
$s_1'(d)$
$s_2'(d)$
Size of the problem

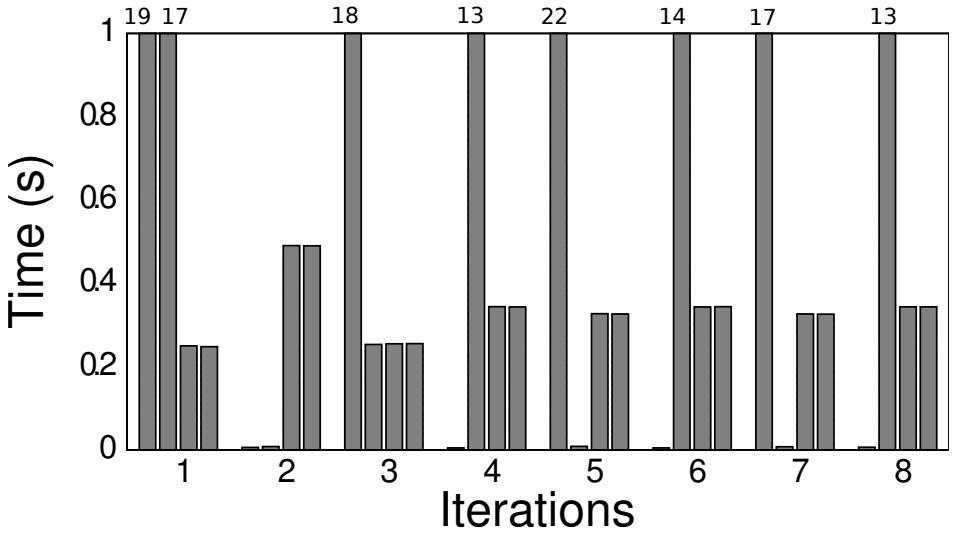

Models Updated
$s_1(d)$
$s_2(d)$
Absolute speed
$s_1'(d)$
$s_2'(d)$
Size of the problem

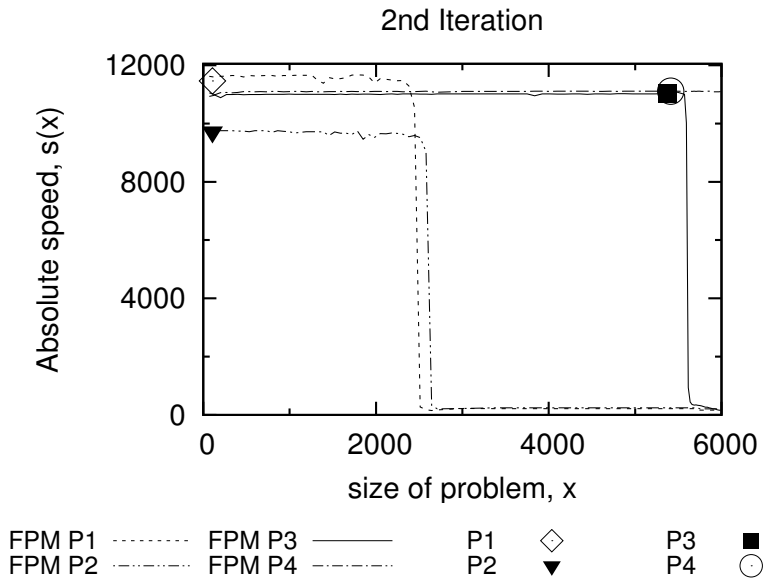
# Experimental Results: Constant Model



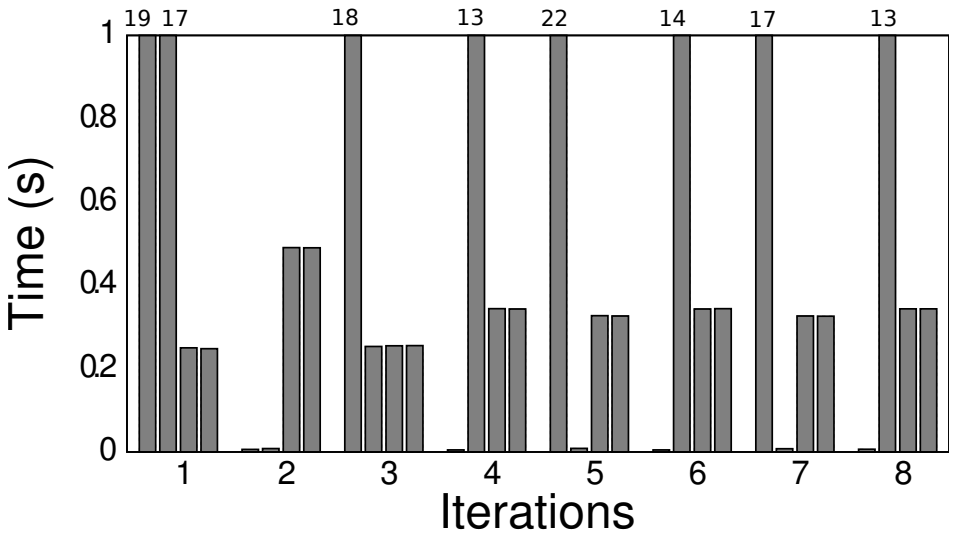



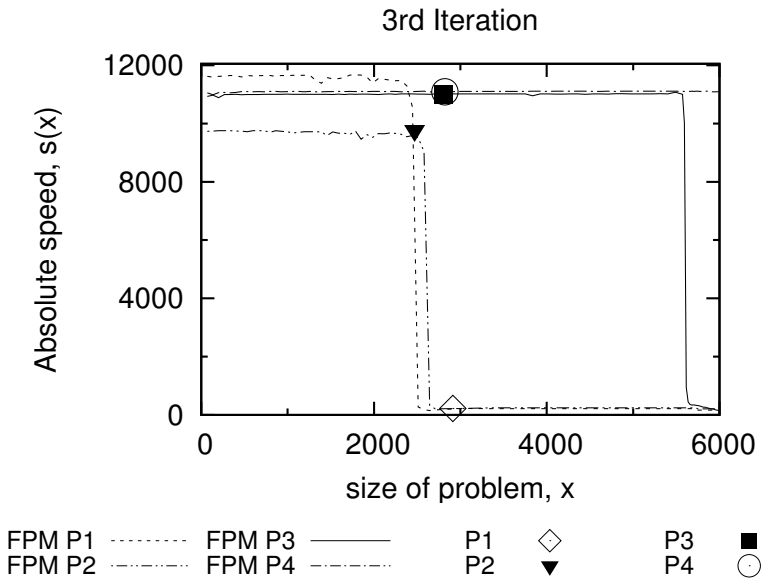
# Experimental Results: Constant Model



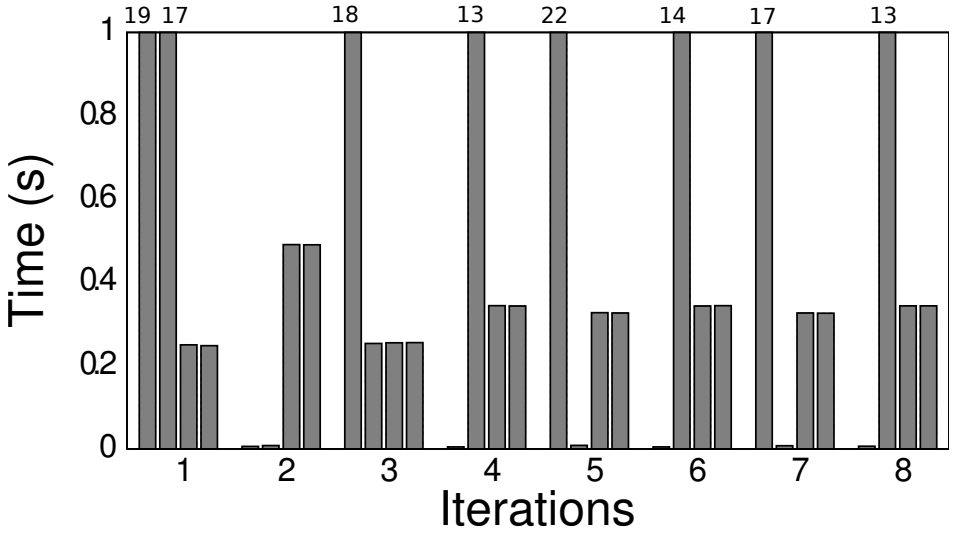



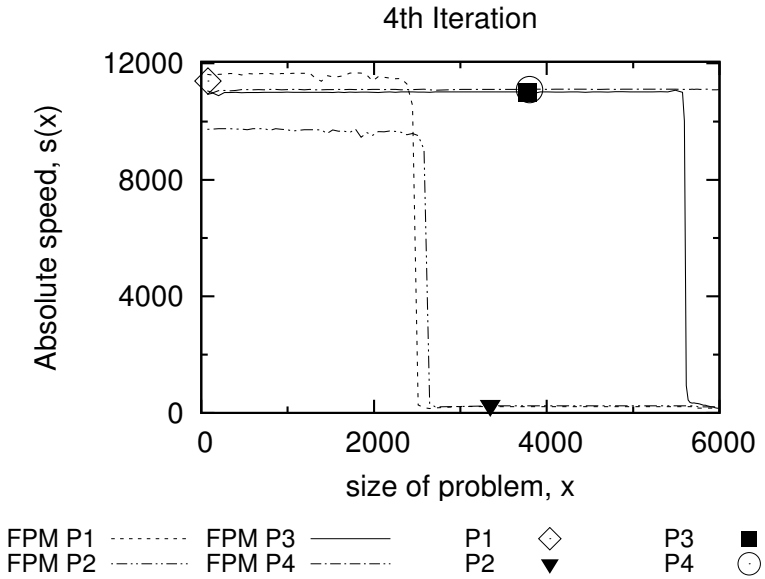
# Experimental Results: Constant Model



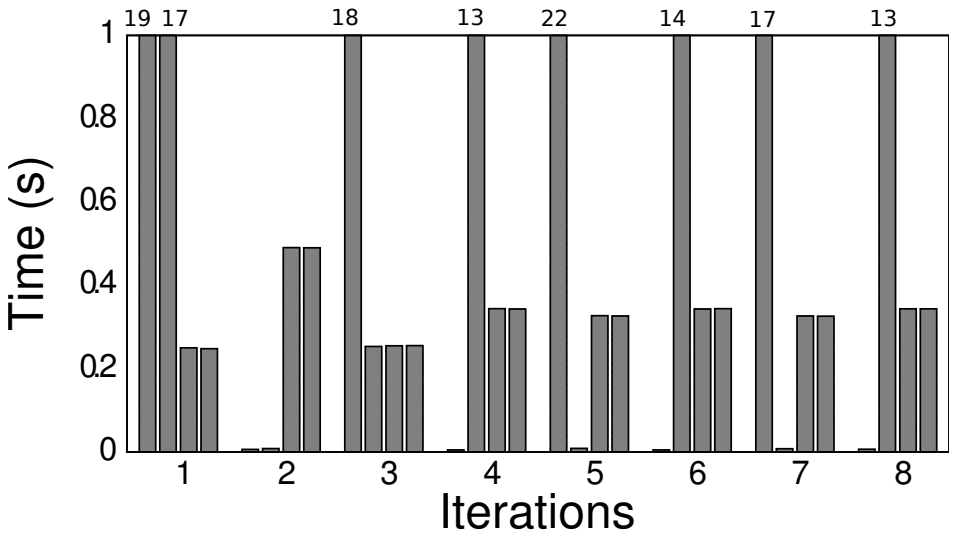



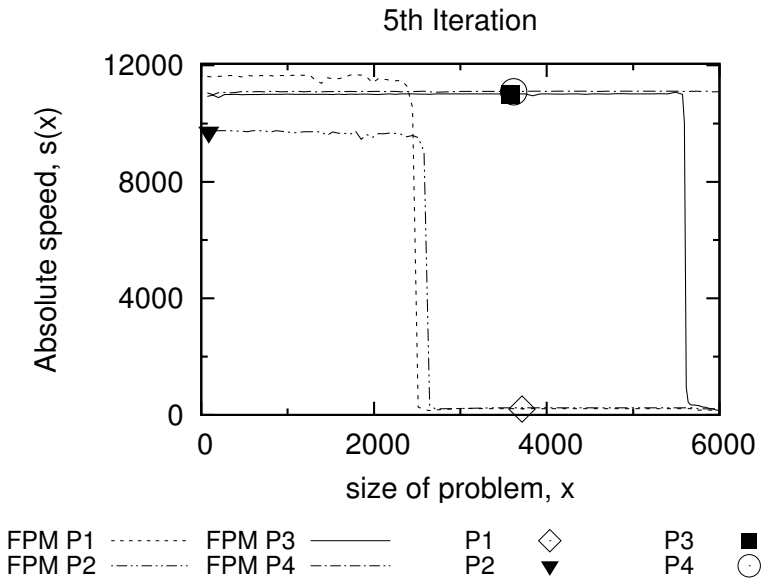
# Experimental Results: Constant Model



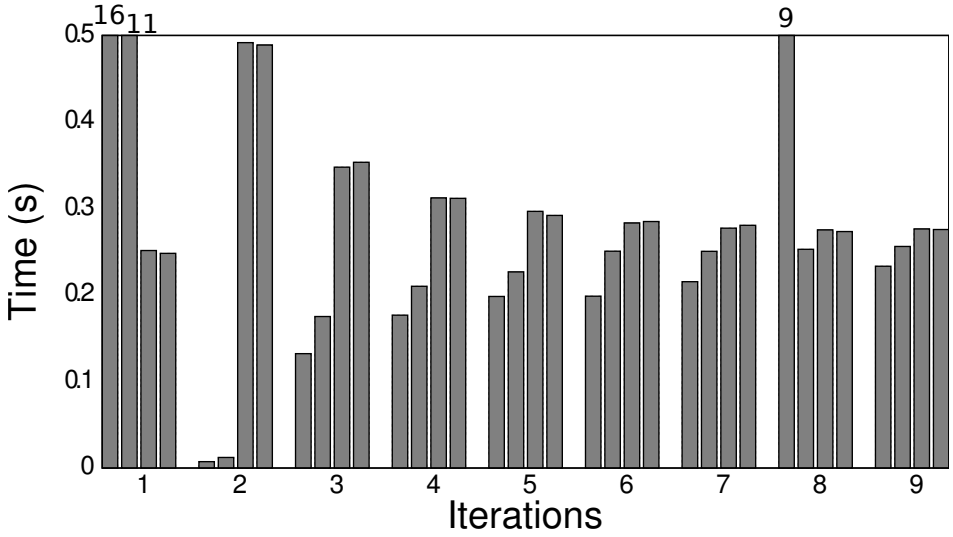



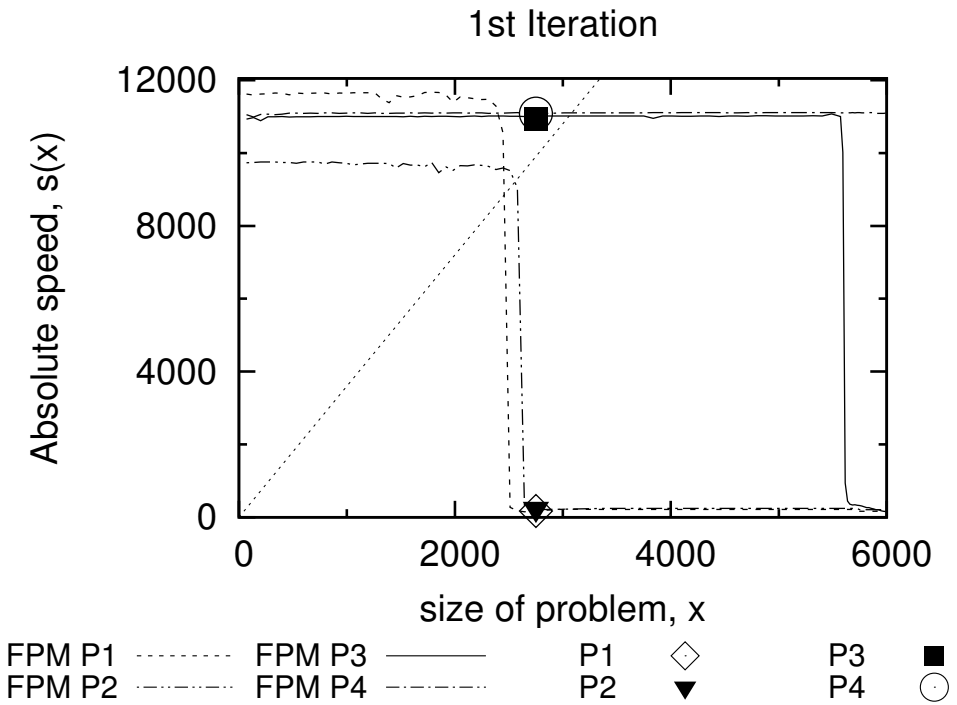
# Experimental Results: Constant Model



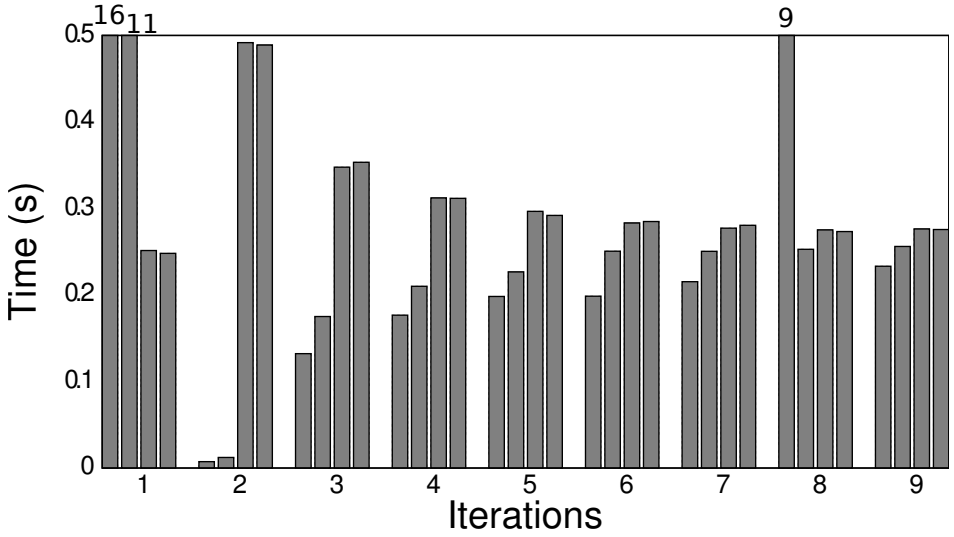



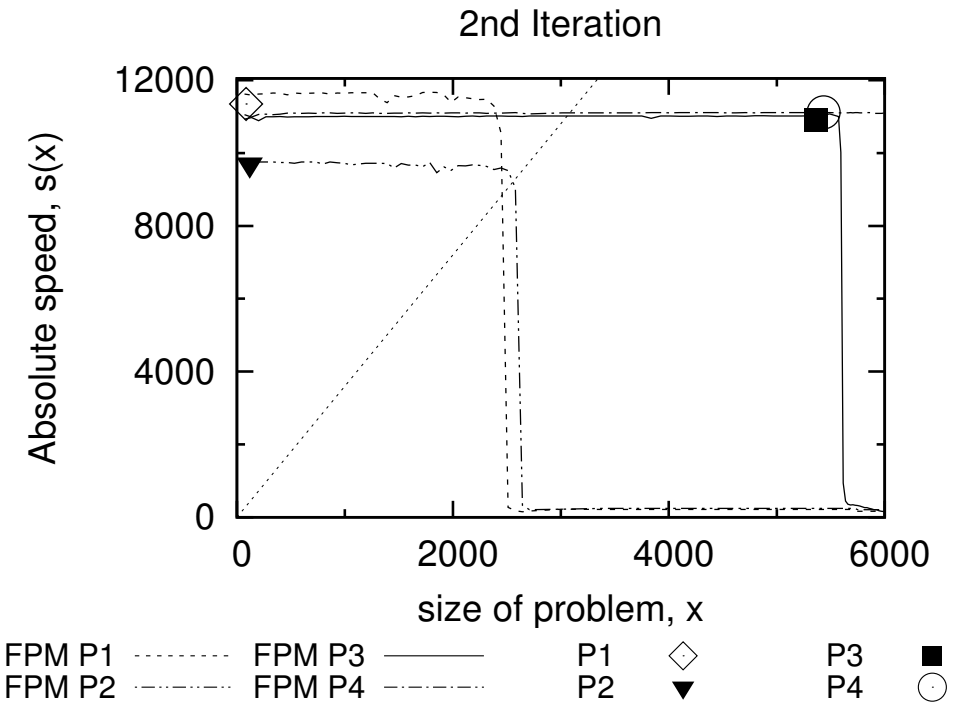
# Experimental Results: FPM based Partitioning



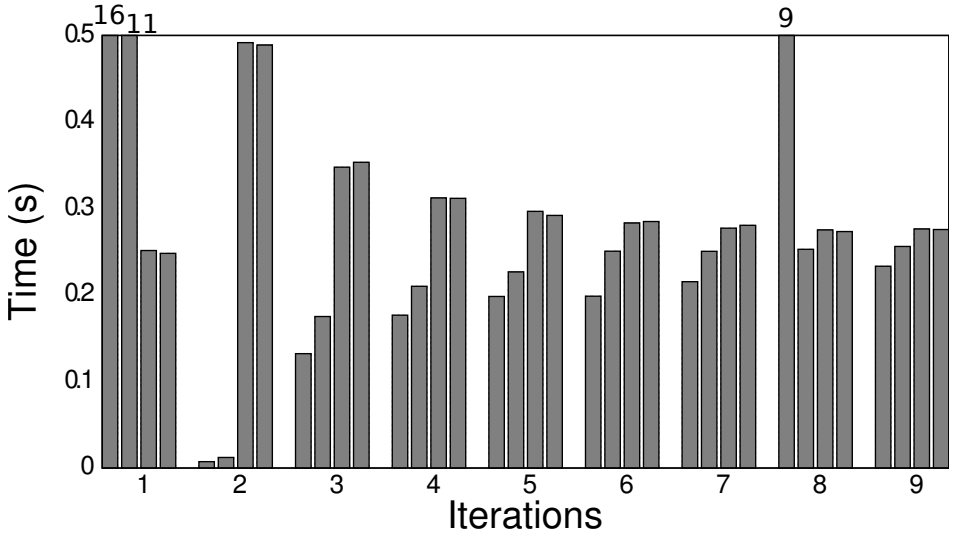



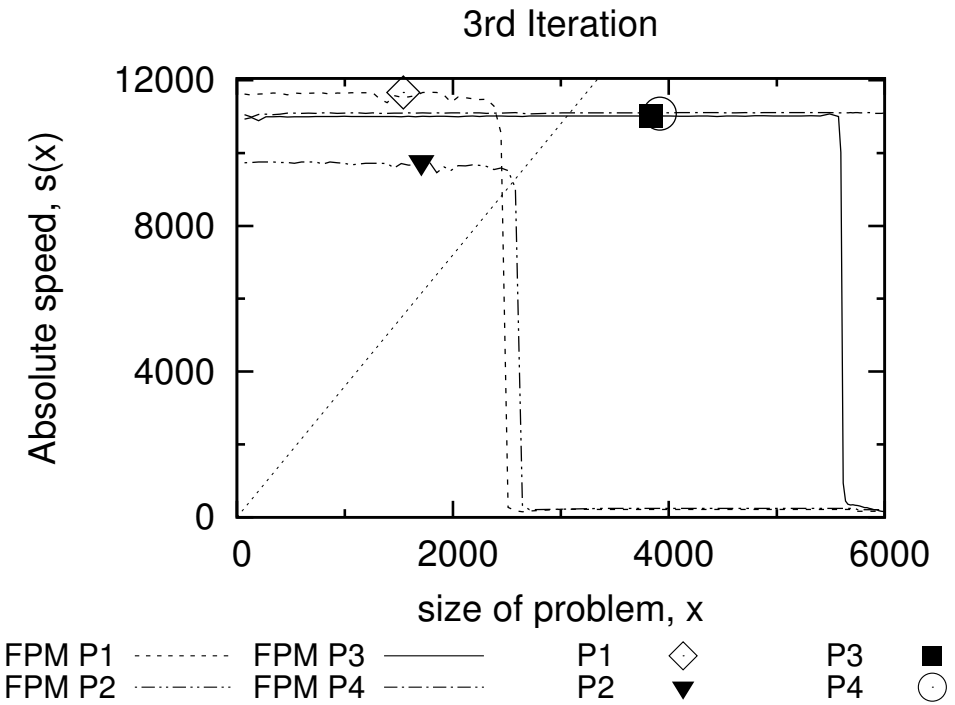
# Experimental Results: FPM based Partitioning



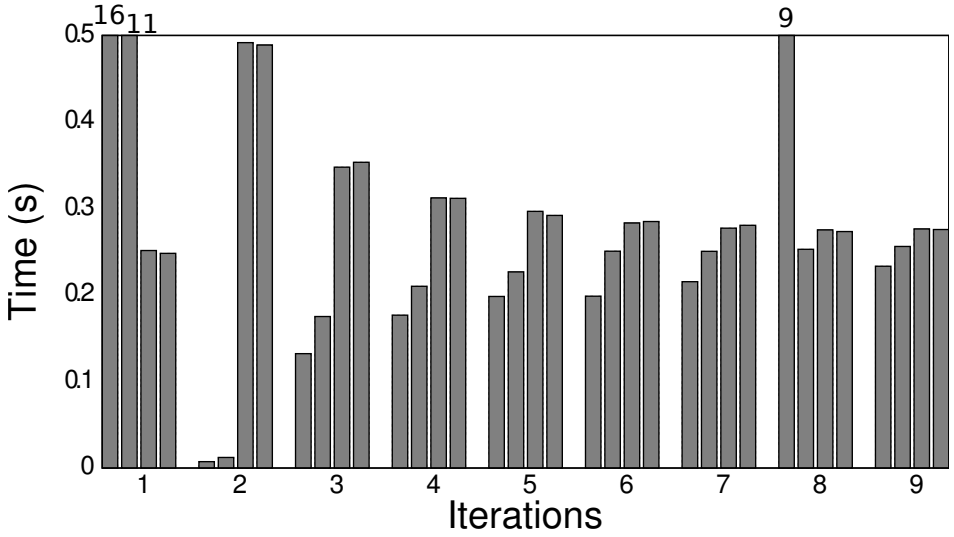



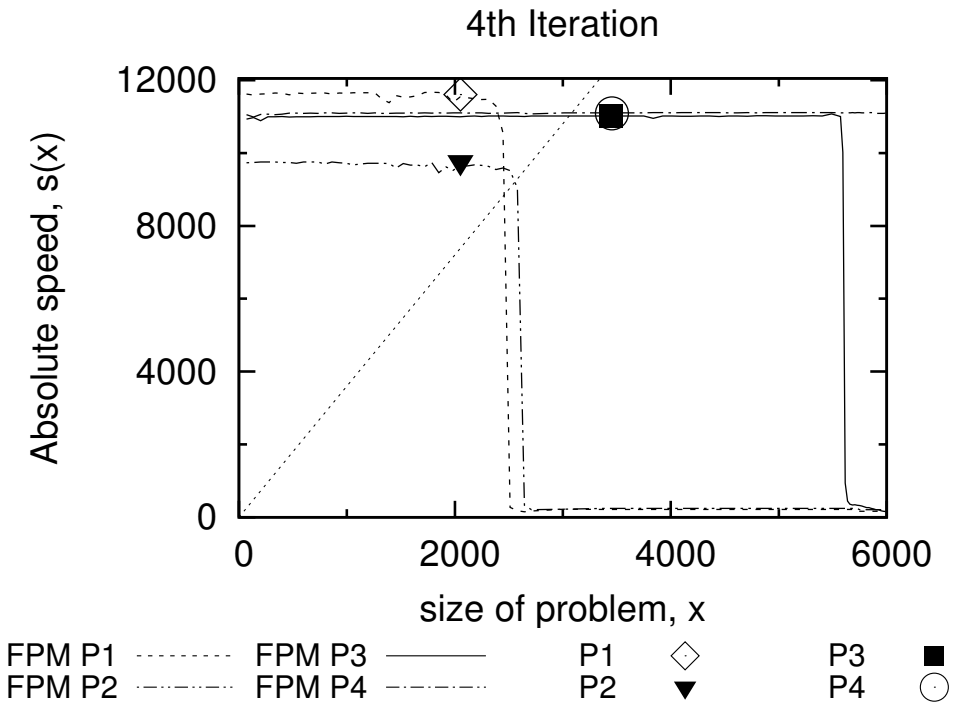
# Experimental Results: FPM based Partitioning



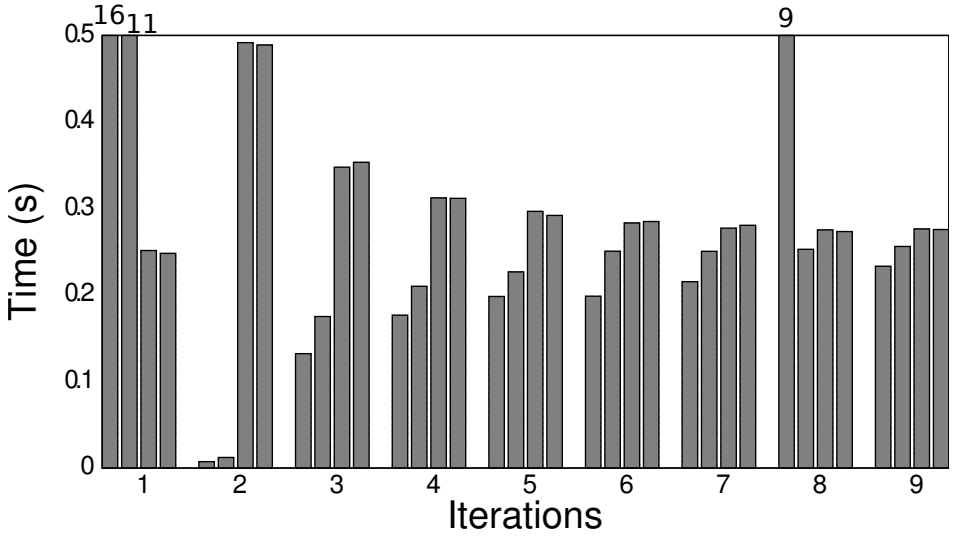



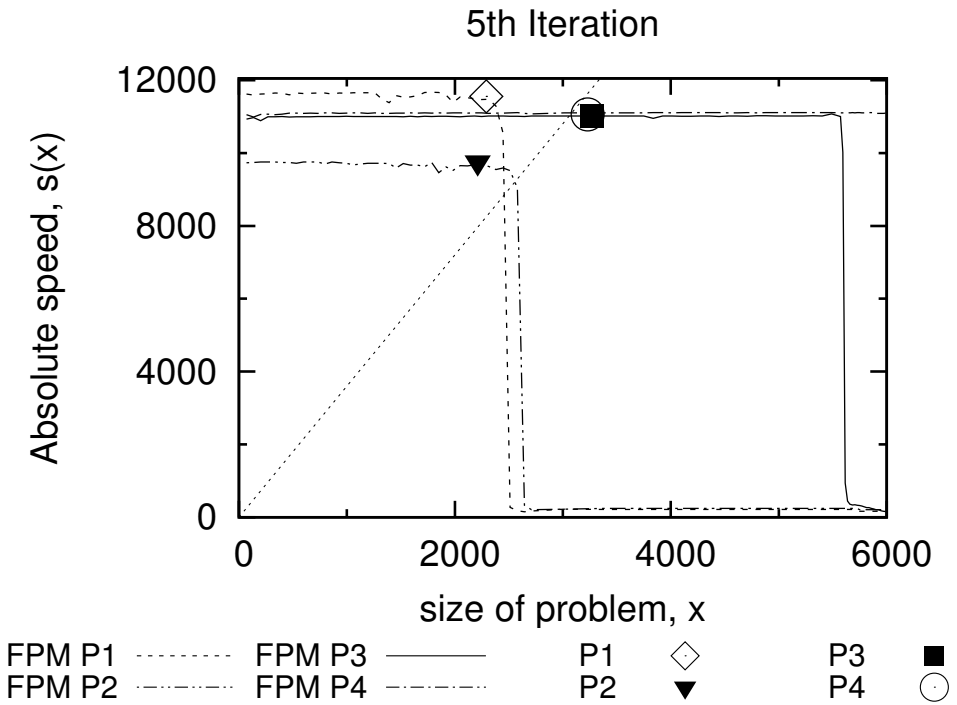
# Experimental Results: FPM based Partitioning



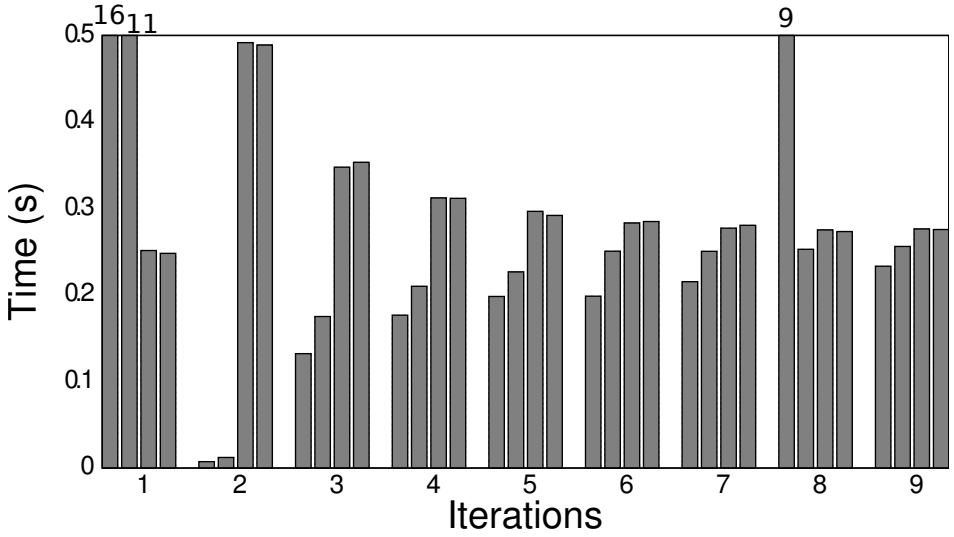



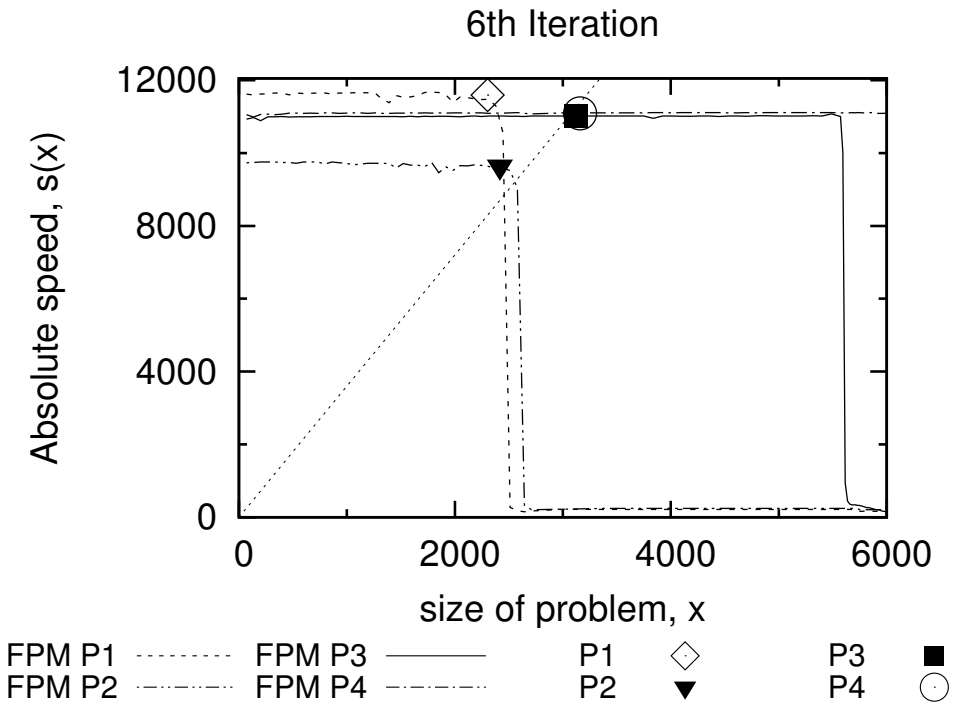
# Experimental Results: FPM based Partitioning



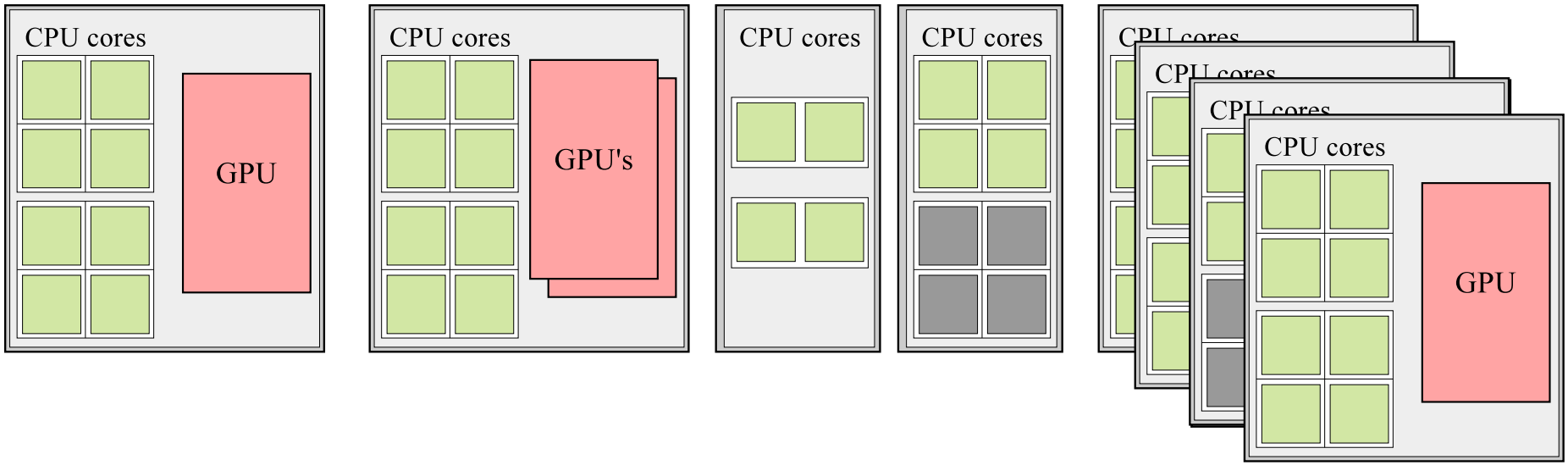



# Experimental Results: FPM based Partitioning



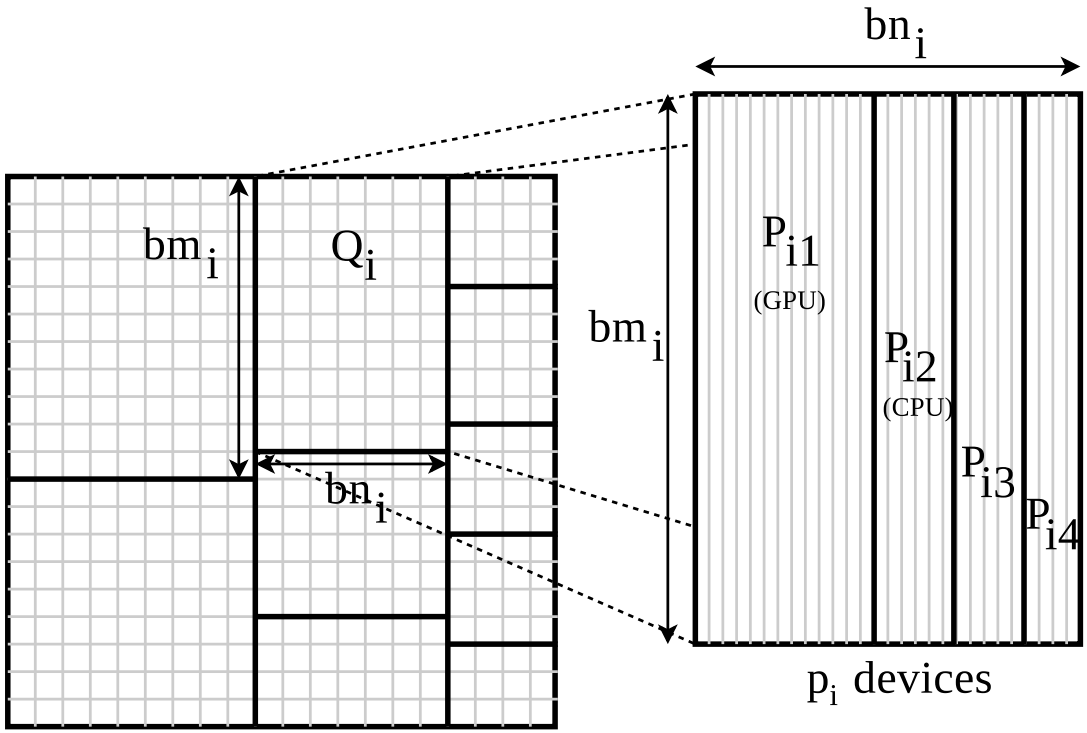



# Target Hierarchical Heterogeneous Platform



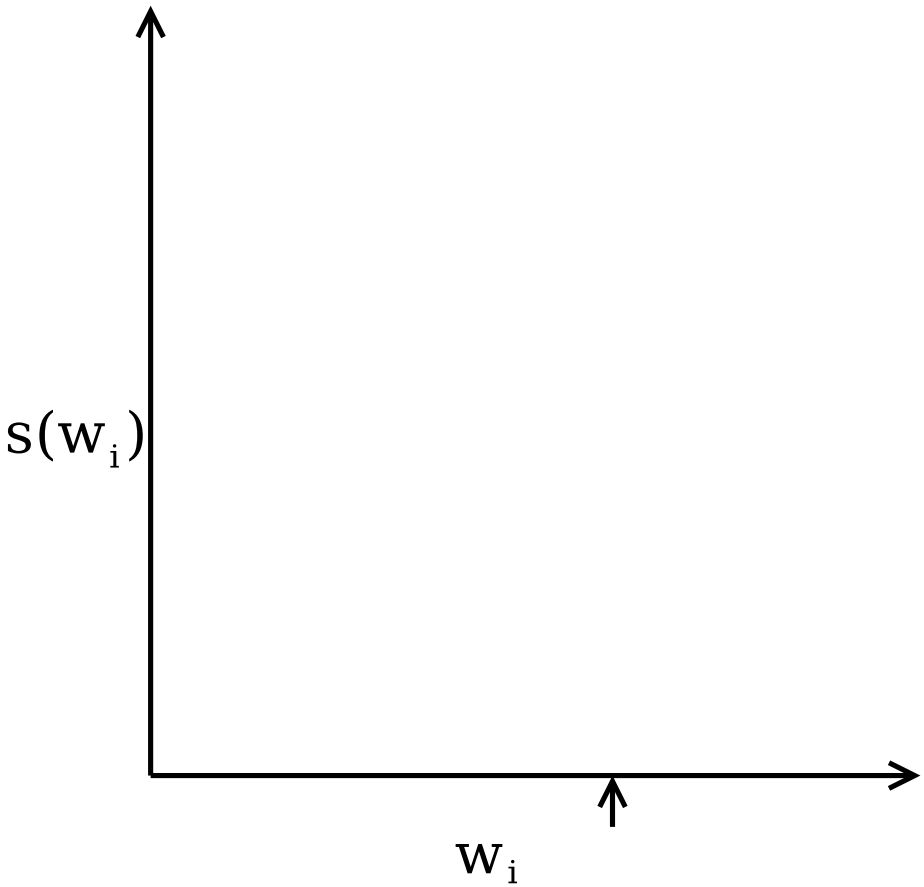

- Heterogeneity between nodes
- Heterogeneity between devices within a node
- Partition matrix between nodes
- Sub-partition between devices within a node
- eg. Grid'5000 Grenoble site.

# Hierarchical Partitioning Algorithm

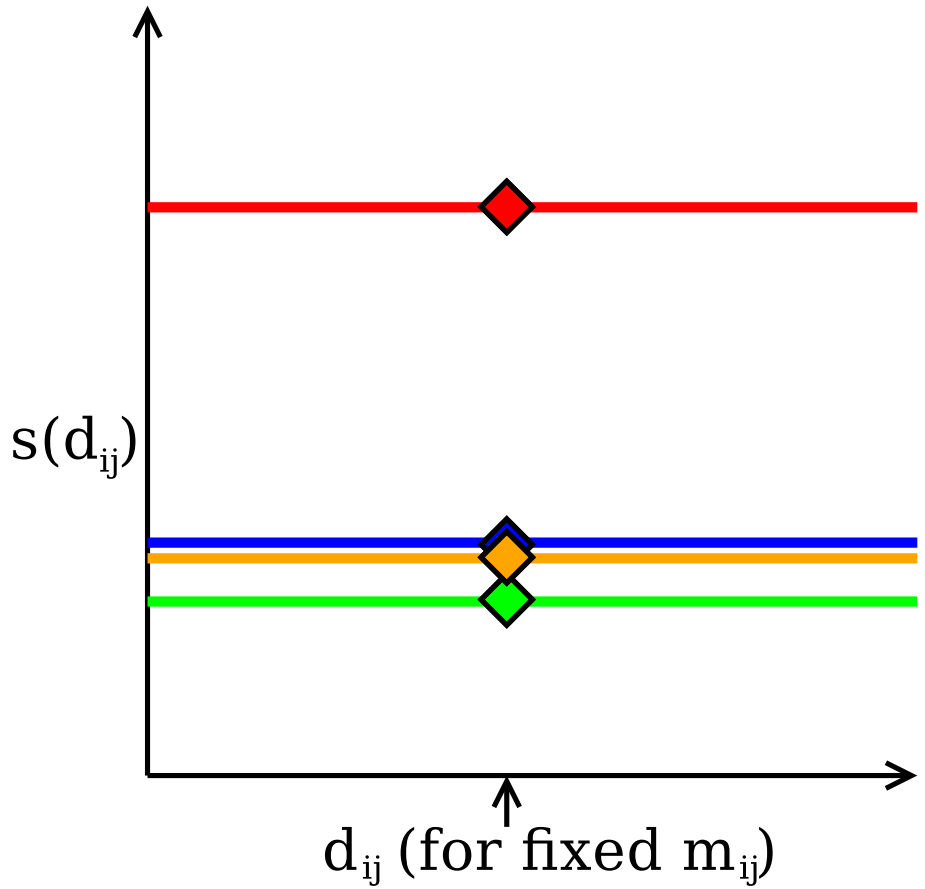



- Hierarchy in platform $\rightarrow$ hierarchy in partitioning
  - Nested parallelism
  - inter-node partitioning algorithm (INPA)
  - inter-device partitioning algorithm (IDPA)
  - IDPA is nested inside INPA

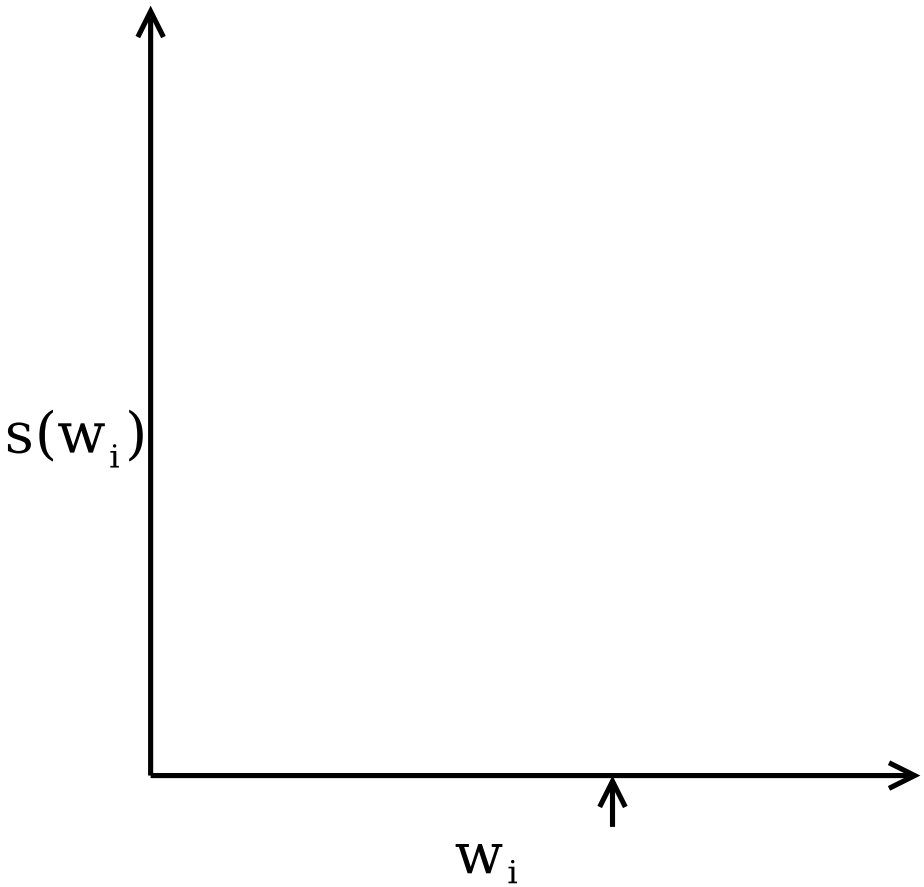
## Node Models



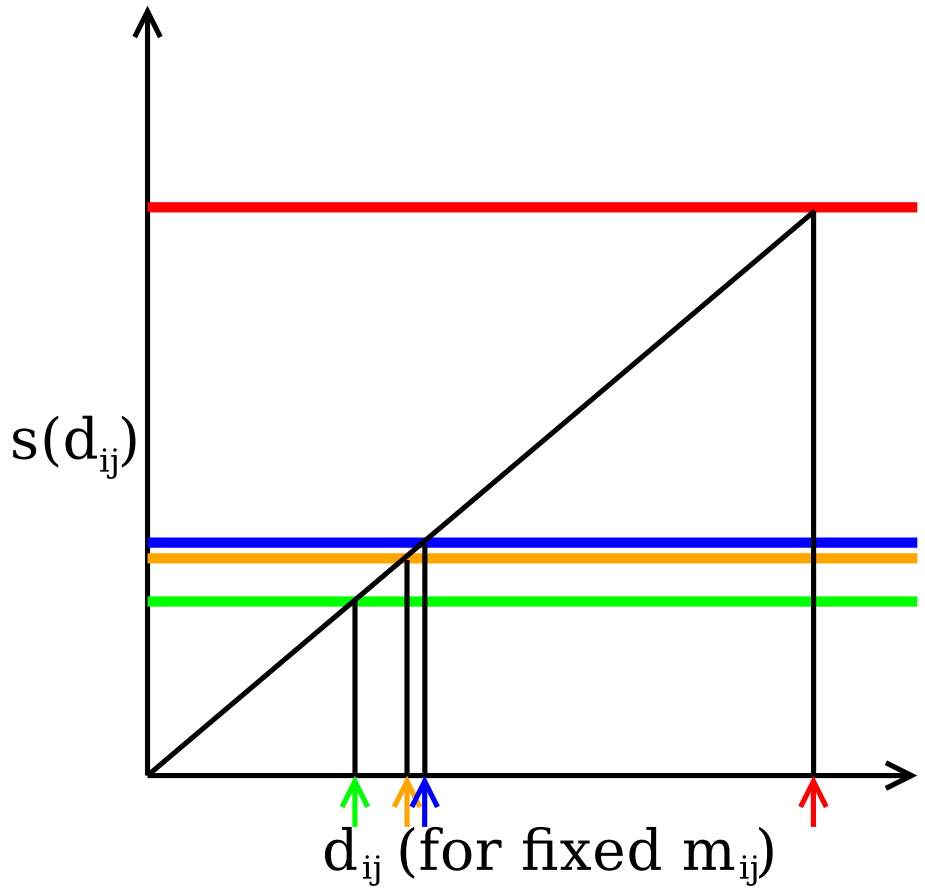

## Device Models for Node *i*



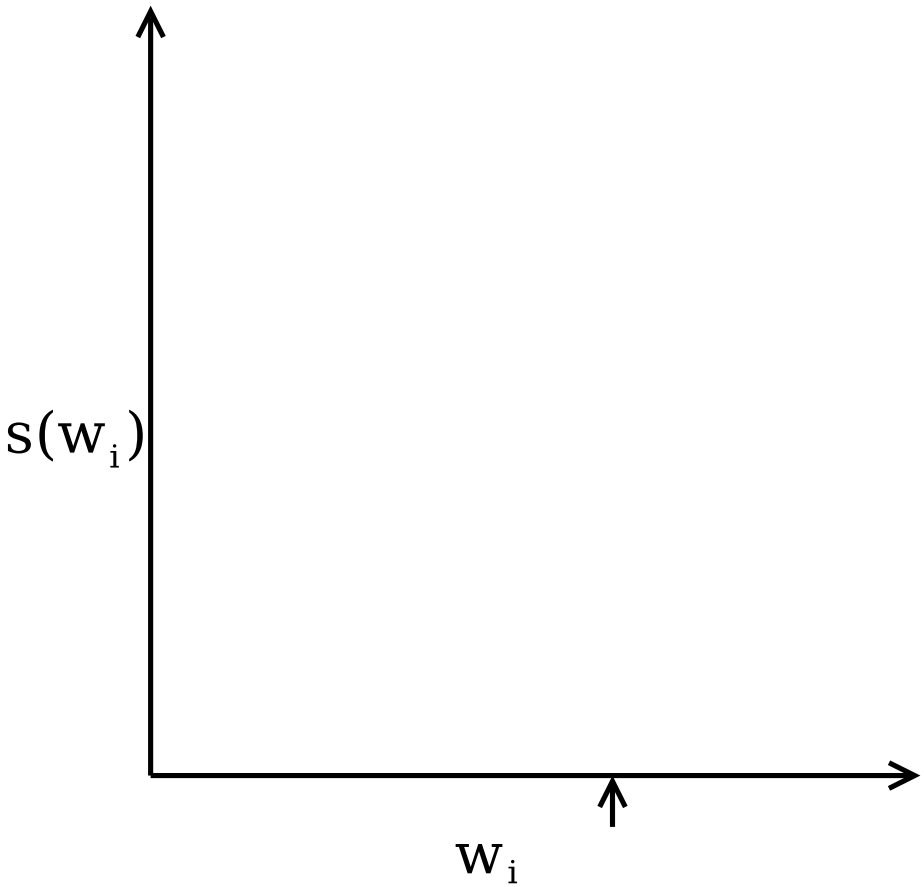
## Node Models



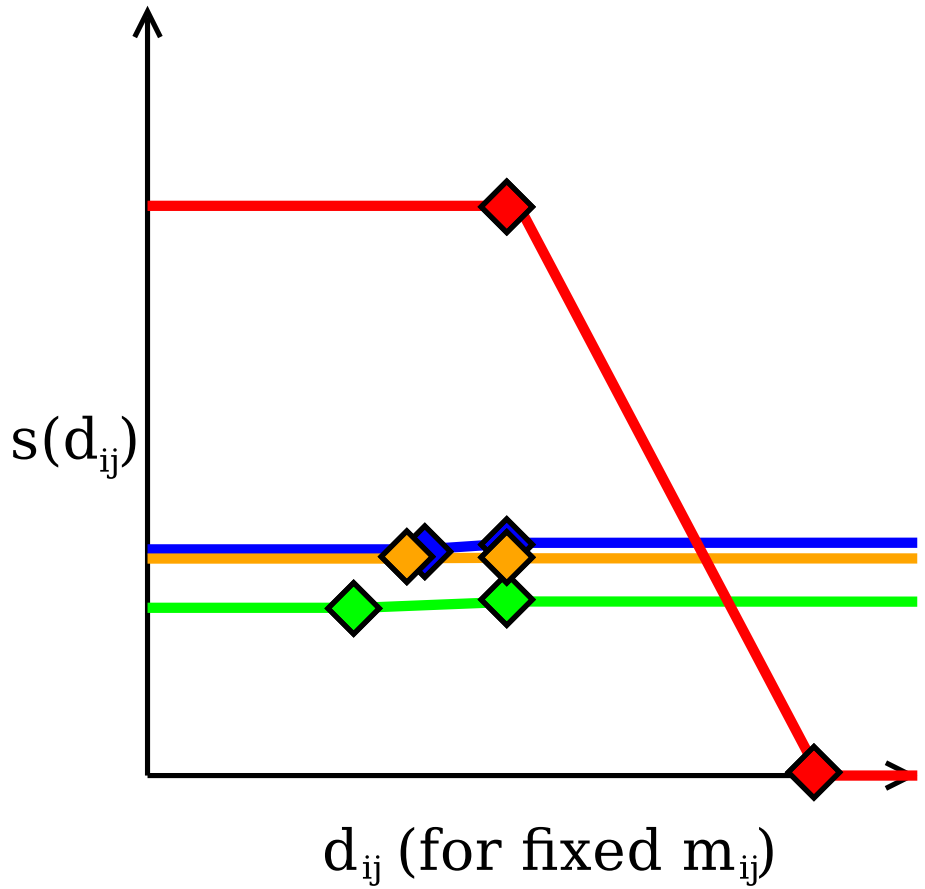

## Device Models for Node $i$



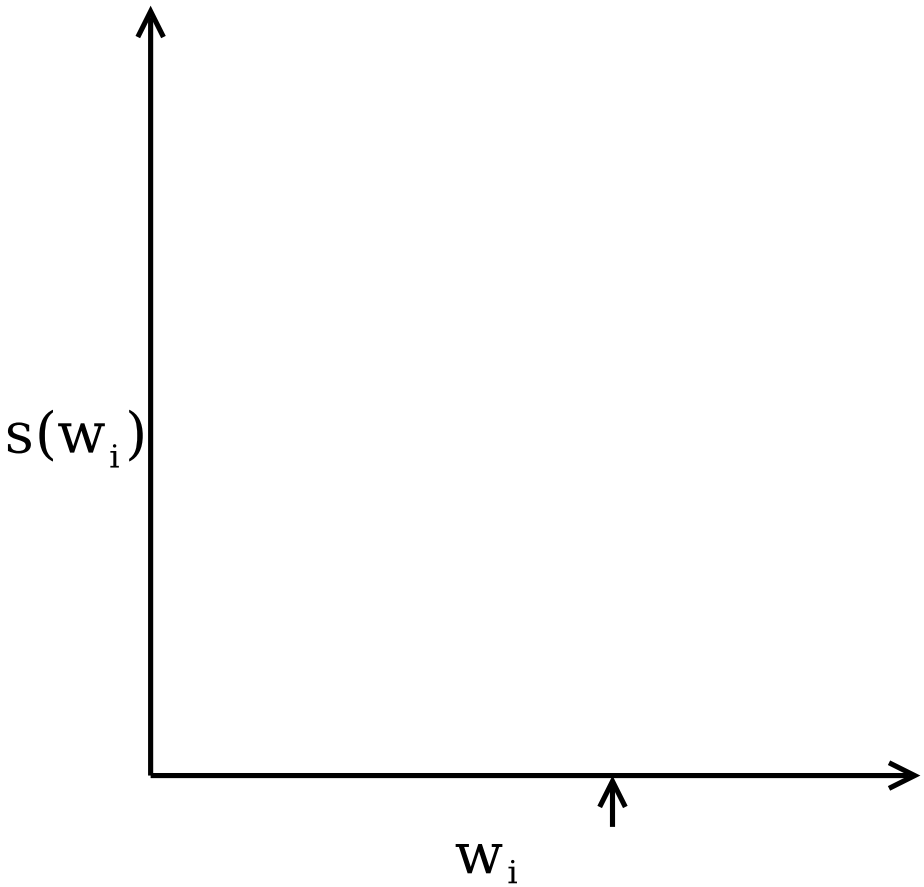
## Node Models



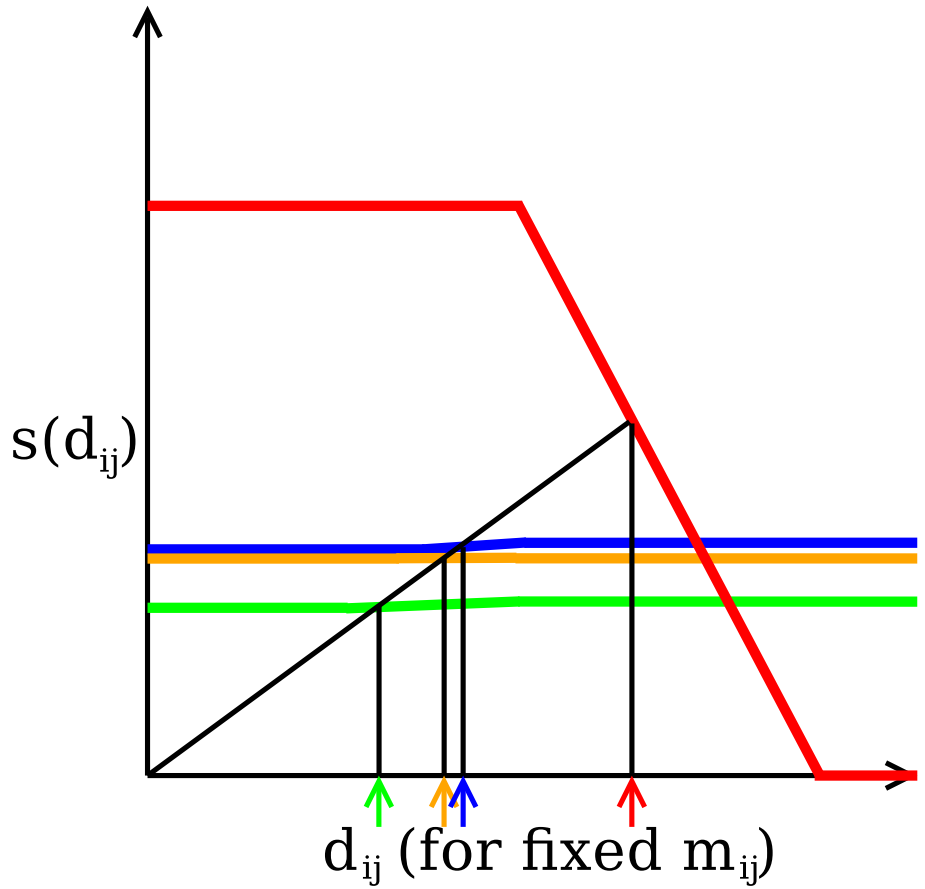

## Device Models for Node *i*



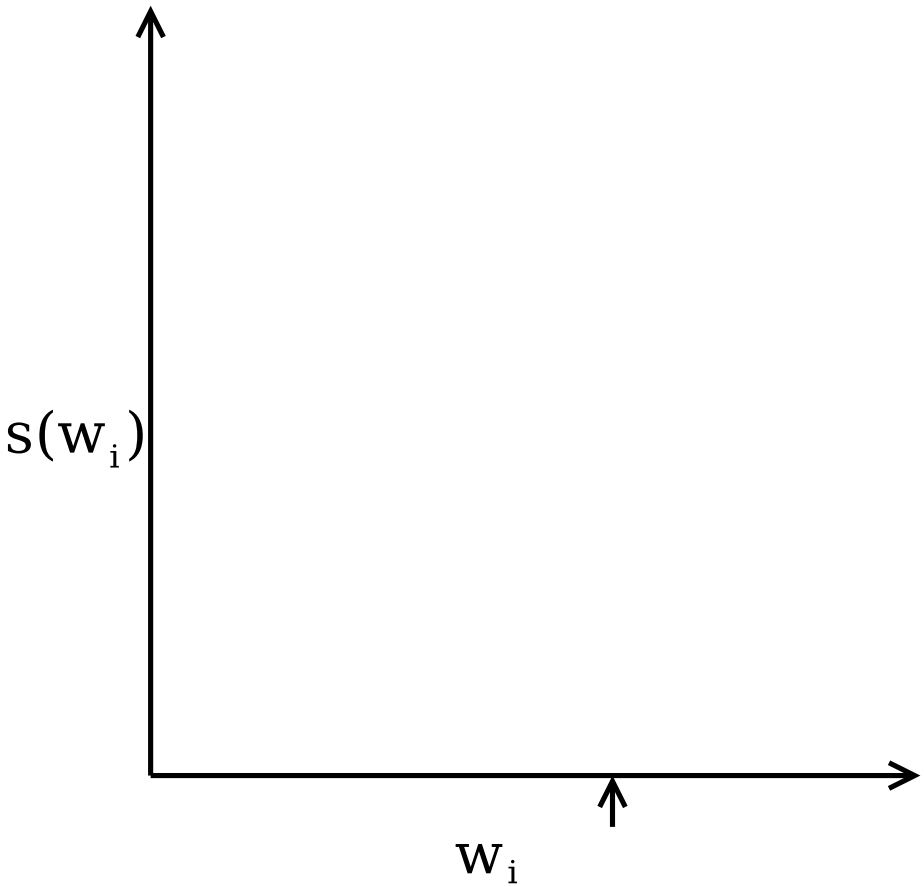
## Node Models



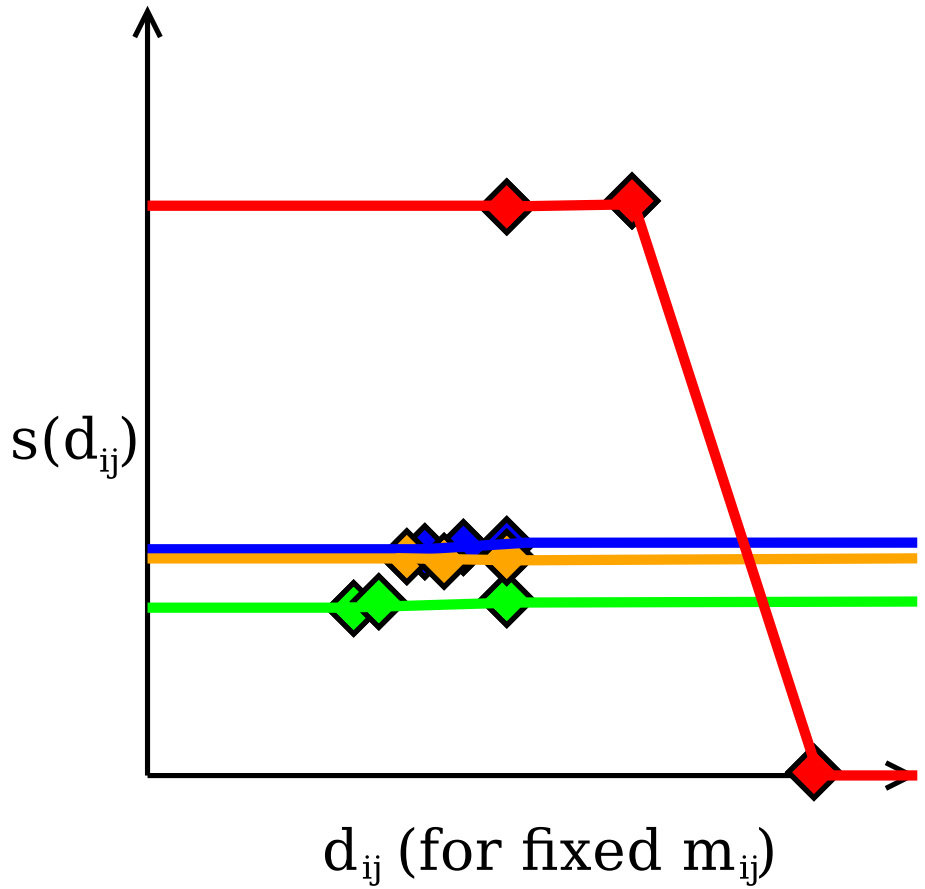

## Device Models for Node *i*



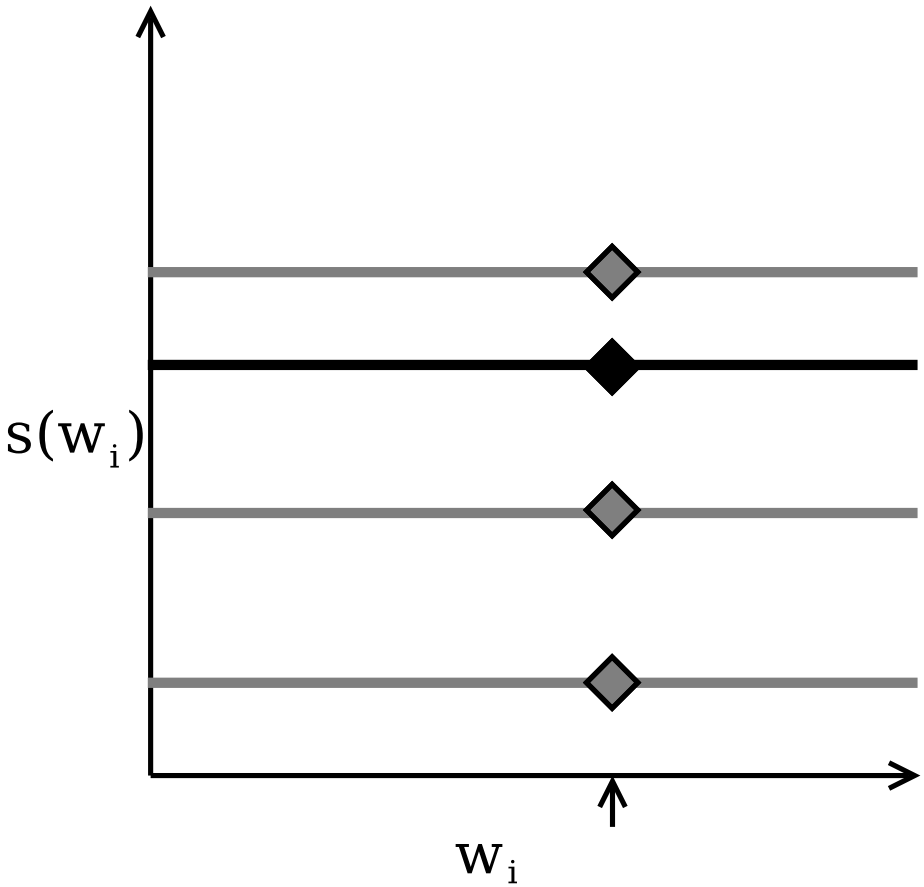
## Node Models



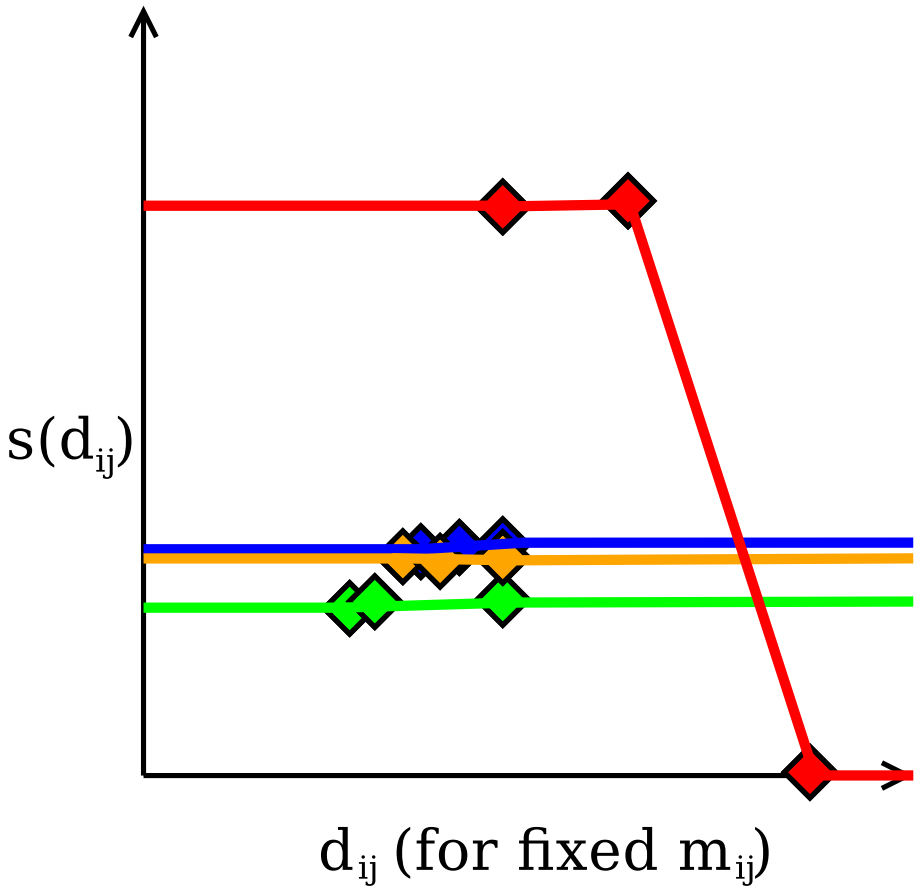

## Device Models for Node *i*



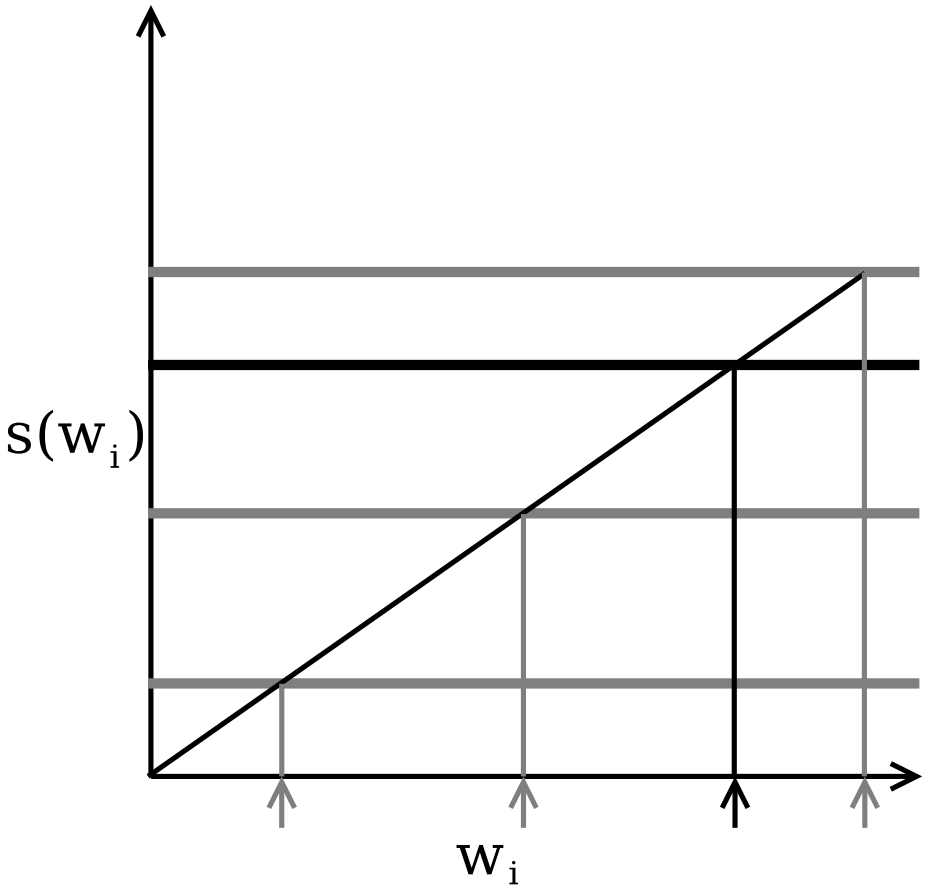
## Node Models



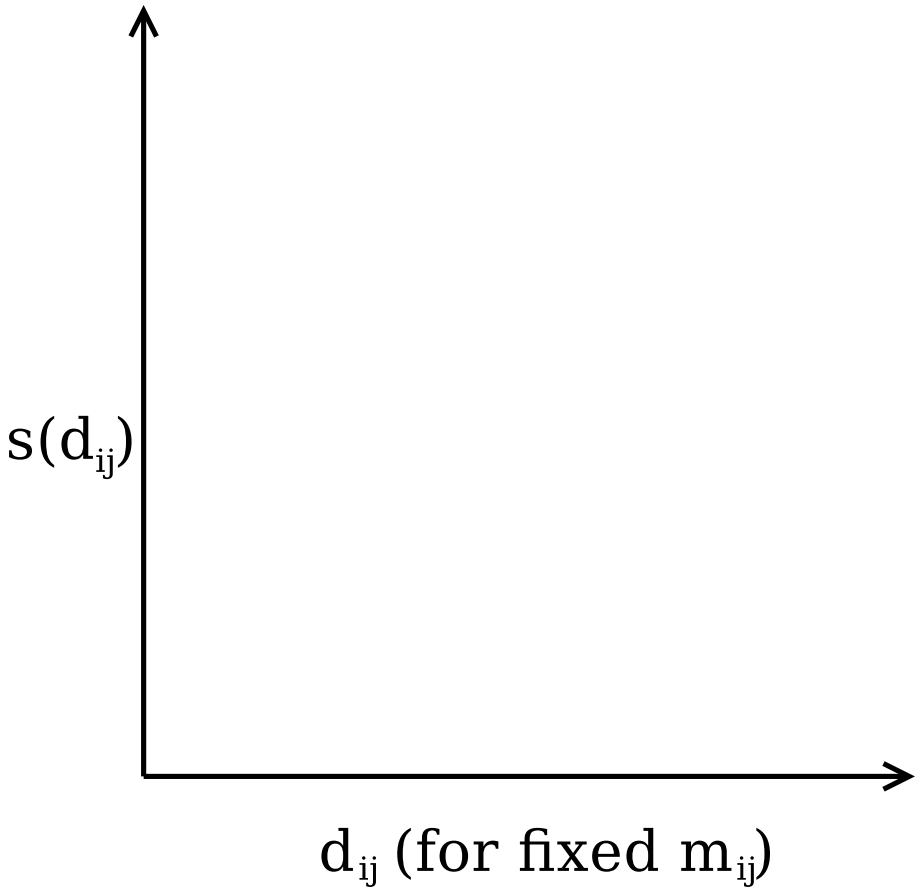

## Device Models for Node *i*



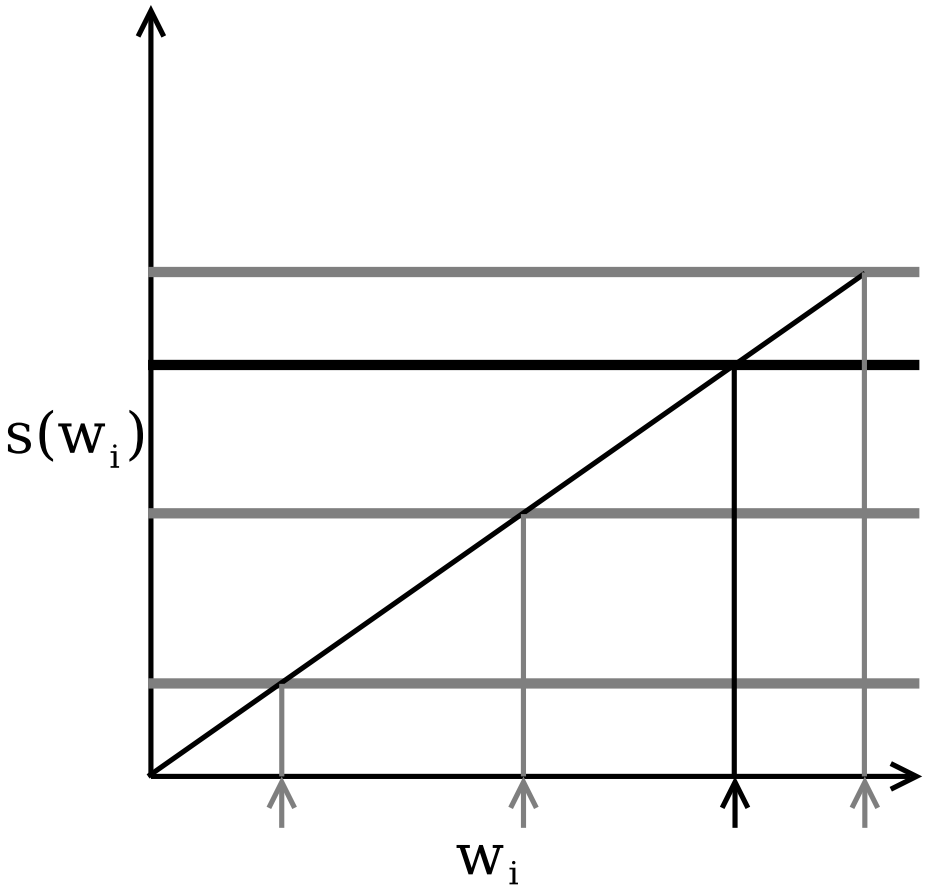
## Node Models



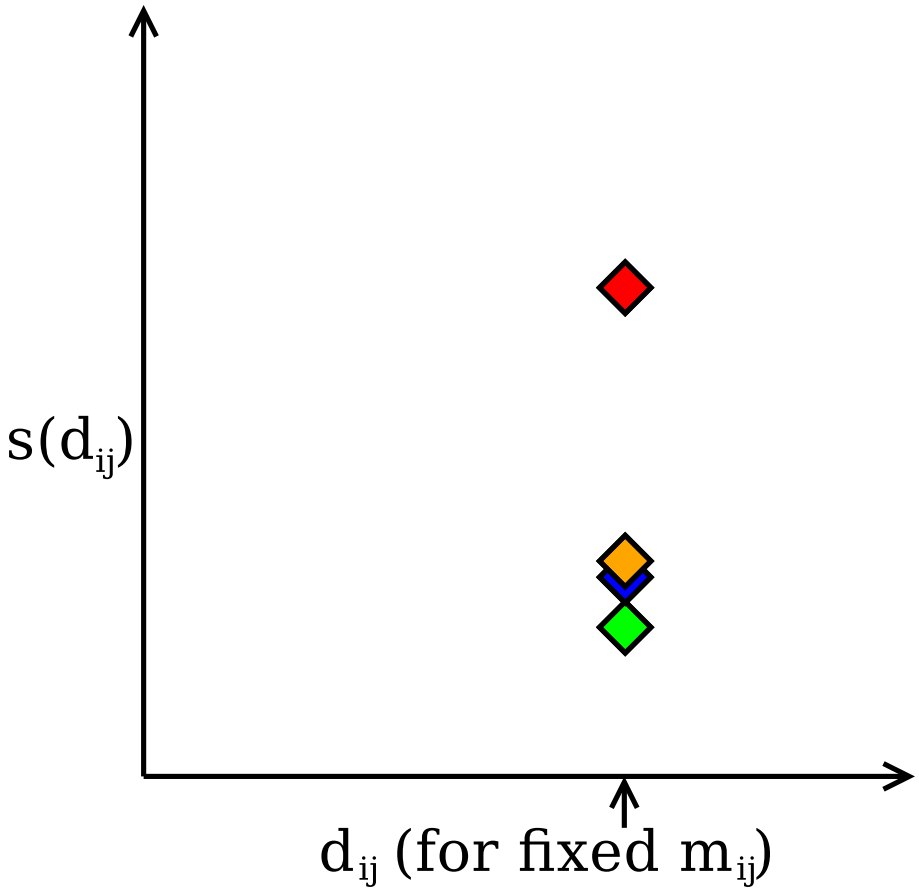

## Device Models for Node $i$



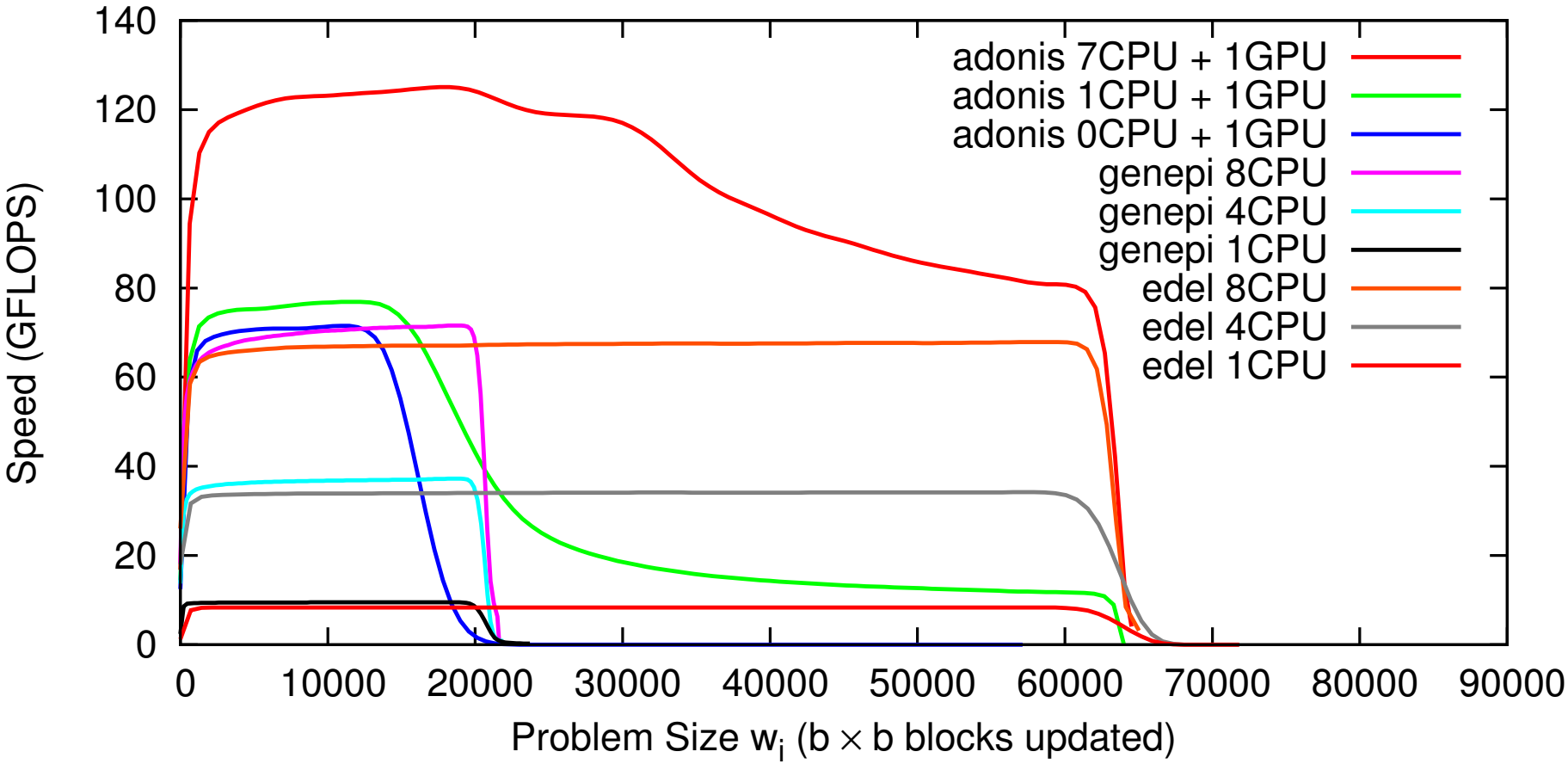
## Node Models



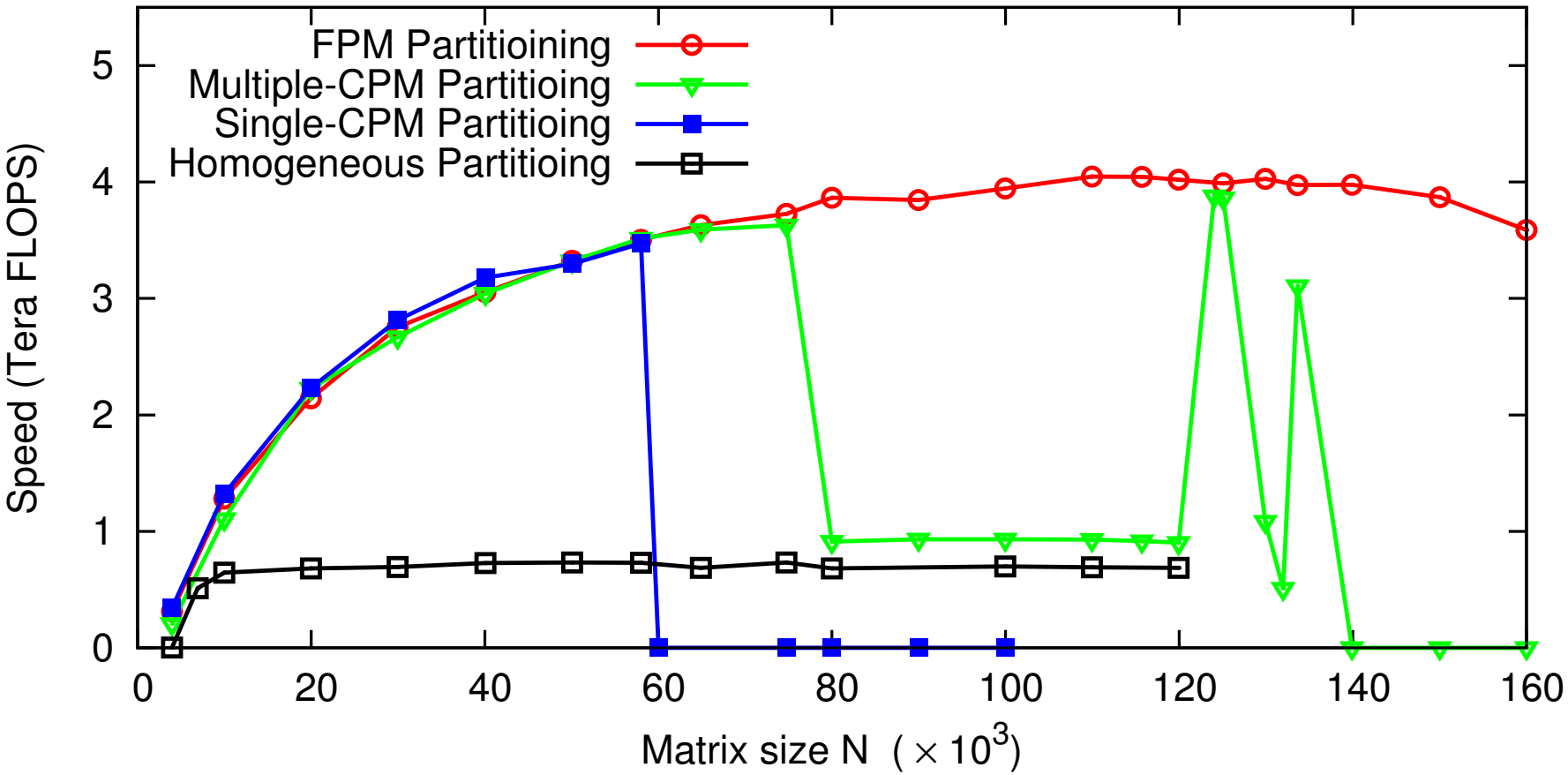

## Device Models for Node $i$



# Experimental Results: Reference full models



# Experimental Results



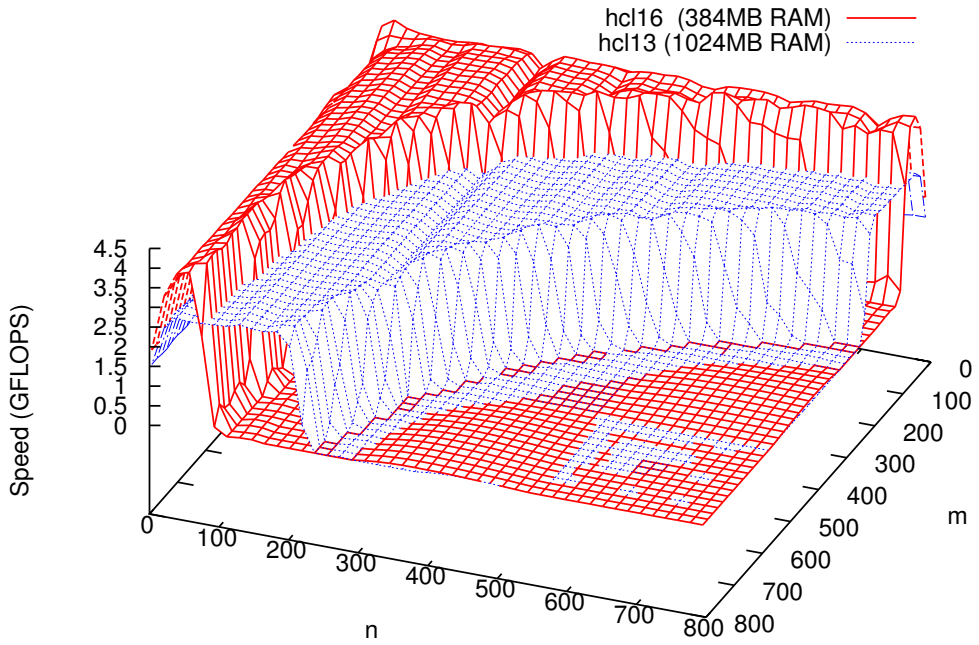

90 nodes, 432 CPUs, 12 GPUs from Grid5000 Grenoble site

# Current Work: Two-Dimensional Partitioning

## Two-Dimensional Matrix Partitioning Based on 2D Models

- Numerical solution
- 3-step column based algorithm

Both algorithms use column based partitioning. Arrangement of processors into a 2D grid is an input.



# Numerical 2D partitioning algorithm

Solve as a constrained non-linear minimisation problem

Function to be minimised:

$$\sum_{j=1}^{q} \sum_{i=1}^{p_j} t_{ij}$$

with $q + 1$ constraints:

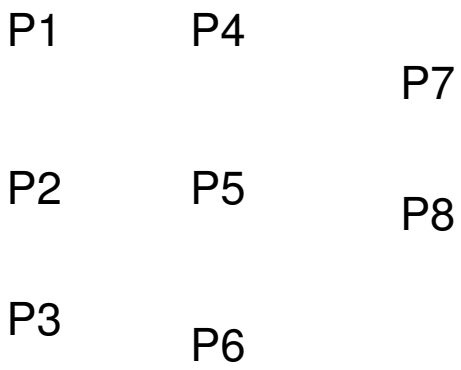
$$n_1 + n_2 + \ldots + n_q = N$$

$$m_{1j} + m_{2j} + \ldots + m_{p_j} = M, \text{ for } 1 \leq j \leq q$$

- Implemented with the NLopt library.
- However, convergence and reliability issues still need to be solved
- Possibly posing the minimisation function and constraints differently will aid convergence.

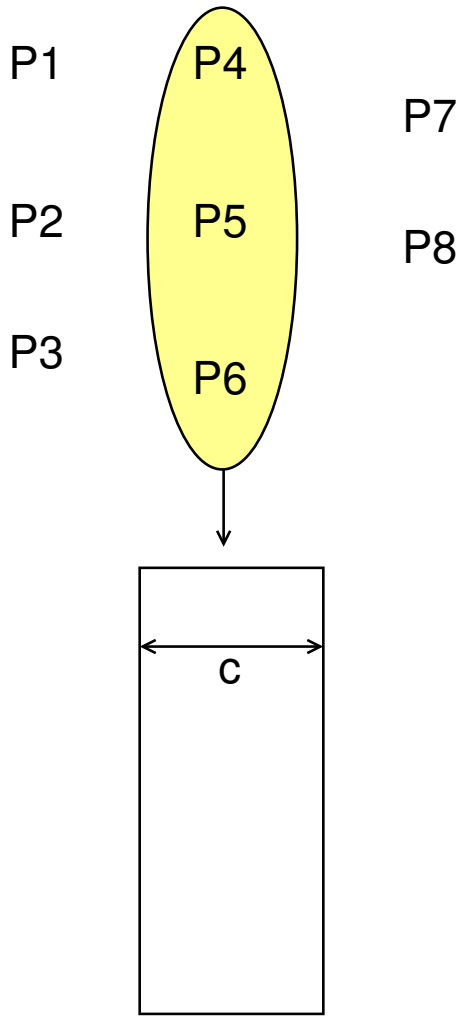
# 3-step column based algorithm

1. Extract 1D model from 2D with $n = c$ for $1 \leq c \leq N$.
   Partition to find optimum time, add point $s_{\text{col}}(c)$ to model.
2. Use 1D partitioning find optimum column widths.
3. Find optimum heights for each processor within the columns.



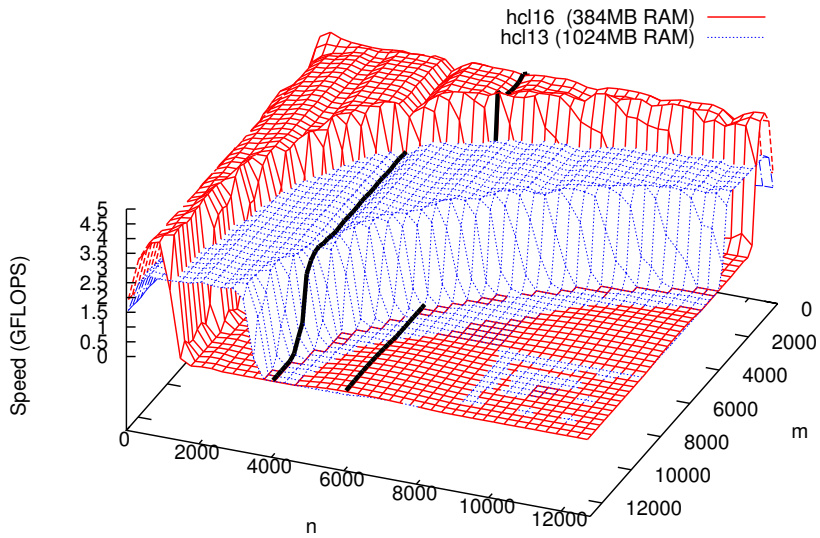
# 3-step column based algorithm

1. Extract 1D model from 2D with $n = c$ for $1 \leq c \leq N$.
   Partition to find optimum time, add point $s_{\text{col}}(c)$ to model.
2. Use 1D partitioning find optimum column widths.
3. Find optimum heights for each processor within the columns.



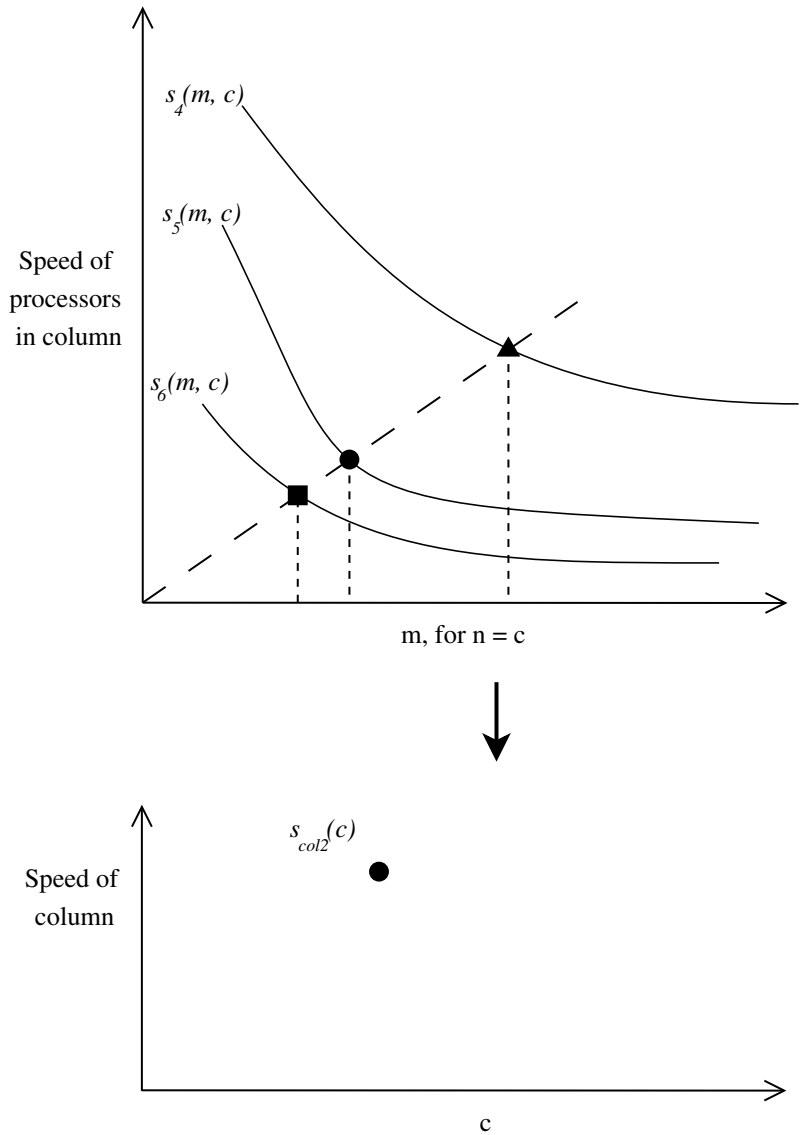
# 3-step column based algorithm

1. Extract 1D model from 2D with $n = c$ for $1 \leq c \leq N$.
   Partition to find optimum time, add point $s_{\text{col}}(c)$ to model.
2. Use 1D partitioning find optimum column widths.
3. Find optimum heights for each processor within the columns.



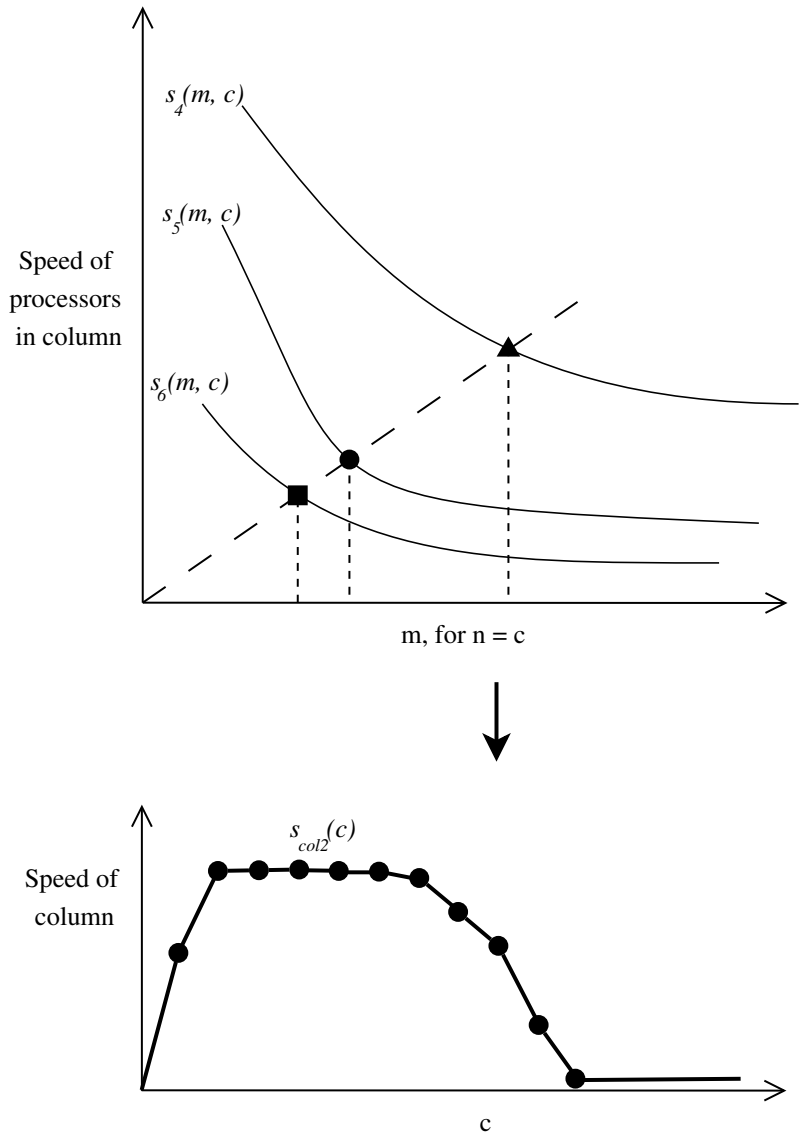
# 3-step column based algorithm

1. Extract 1D model from 2D with $n = c$ for $1 \leq c \leq N$.
   Partition to find optimum time, add point $s_{\text{col}}(c)$ to model.
2. Use 1D partitioning find optimum column widths.
3. Find optimum heights for each processor within the columns.



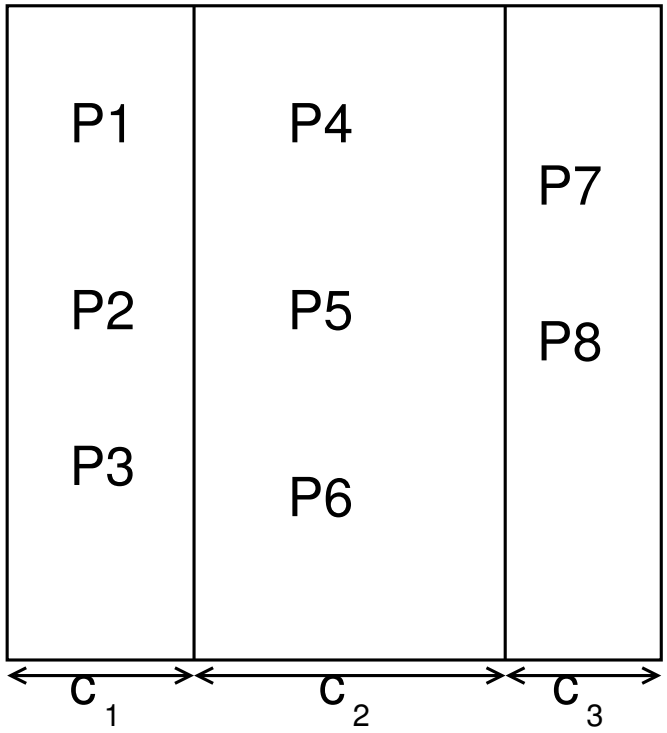
# 3-step column based algorithm

1. Extract 1D model from 2D with $n = c$ for $1 \leq c \leq N$.
   Partition to find optimum time, add point $s_{\text{col}}(c)$ to model.
2. Use 1D partitioning find optimum column widths.
3. Find optimum heights for each processor within the columns.



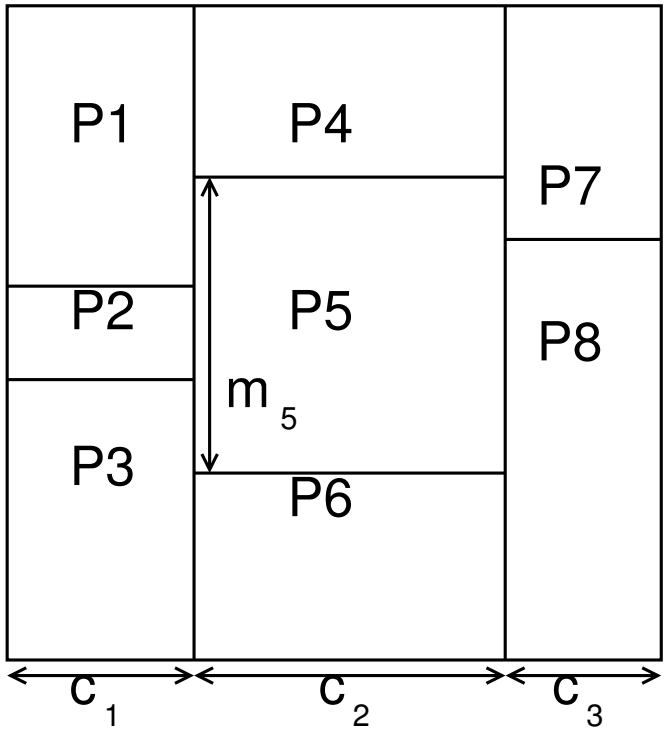
# 3-step column based algorithm

1. Extract 1D model from 2D with $n = c$ for $1 \leq c \leq N$.
   Partition to find optimum time, add point $s_{\text{col}}(c)$ to model.
2. Use 1D partitioning find optimum column widths.
3. Find optimum heights for each processor within the columns.



# 3-step column based algorithm

1. Extract 1D model from 2D with $n = c$ for $1 \leq c \leq N$.
   Partition to find optimum time, add point $s_{\text{col}}(c)$ to model.
2. Use 1D partitioning find optimum column widths.
3. Find optimum heights for each processor within the columns.



## Applications

- Signal Processing Systems group, INESC-ID, Lisbon, Portugal
  - Using FPMs to partition database requests on heterogeneous platform.
  - Extended FPMs to overlap communications, $\times 4$ speedup for FFT.
- Division of Scientific Computing, Uppsala University, Sweden
  - Upcoming collaboration to load balance multiphase flow simulations.

## Experimental Platforms

- Grid'5000, 10 sites, 1260 nodes, France
- HCL Cluster, local 16 node experimental cluster.
- SiPS Cluster, 4 node CPU+GPU, Lisbon, Portugal

# Software

## FuPerMod

- FuPerMod package developed at HCL
- Based on system and mathematical software: C/C++, MPI, Autotools, GNU Scientific Library, Boost C++ libraries, BLAS, CUDA Toolkit
- Contains all presented algorithms.
- Designed for easy integration with existing heterogeneous applications.

# Publications


- Lastovetsky, A., and R. Reddy, **"Distributed Data Partitioning for Heterogeneous Processors Based on Partial Estimation of their Functional Performance Models"**, HeteroPar'2009, Delft, Netherlands, LNCS, vol. 6043, Springer, pp. 91-101, 2010.
- Lastovetsky, A., and R. Reddy, **"Two-dimensional Matrix Partitioning for Parallel Computing on Heterogeneous Processors Based on their Functional Performance Models"**, HeteroPar'2009, Delft, Netherlands, LNCS, vol. 6043, Springer, pp. 112-121, 2010.
- Clarke, D., A. Lastovetsky, and V. Rychkov, **"Dynamic Load Balancing of Parallel Computational Iterative Routines on Platforms with Memory Heterogeneity"**, Europar 2010 / Heteropar'2010, Ischia-Naples, Italy, LNCS 6586, Springer, pp. 41-50, 2011.
- Clarke, D., A. Lastovetsky, and V. Rychkov, **"Dynamic Load Balancing of Parallel Computational Iterative Routines on Highly Heterogeneous HPC Platforms"**, PPL, vol. 21, issue 2: World Scientific, pp. 195-217, 06/2011.
- Rychkov, V., D. Clarke, and A. Lastovetsky, **"Using Multidimensional Solvers for Optimal Data Partitioning on Dedicated Heterogeneous HPC Platforms "**, PaCT-2011, LNCS 6873, Kazan, Russia, Springer, pp. 332-346, 19/09/2011.
- Lastovetsky, A., R. Reddy, V. Rychkov, and D. Clarke, **"Design and implementation of self-adaptable parallel algorithms for scientific computing on highly heterogeneous HPC platforms"**, PARCO (under review).
- Clarke, D., A. Lastovetsky, and V. Rychkov, **"Column-Based Matrix Partitioning for Parallel Matrix Multiplication on Heterogeneous Processors Based on Functional Performance Models"**, HeteroPar'2011 , Bordeaux, France, LNCS 7155, Springer, pp. 450-459, 2012.
- Clarke, D., A. Ilic, A. Lastovetsky, L. Sousa, **"Hierarchical Partitioning Algorithm for Scientific Computing on Highly Heterogeneous CPU + GPU Clusters"** EuroPar'2012 (under review).