Hierarchical Partitioning Algorithm for Scientific Computing on Highly Heterogeneous CPU + GPU Clusters

David Clarke[1] Aleksandar Ilic[2]
Alexey Lastovetsky[1] Leonel Sousa[2]

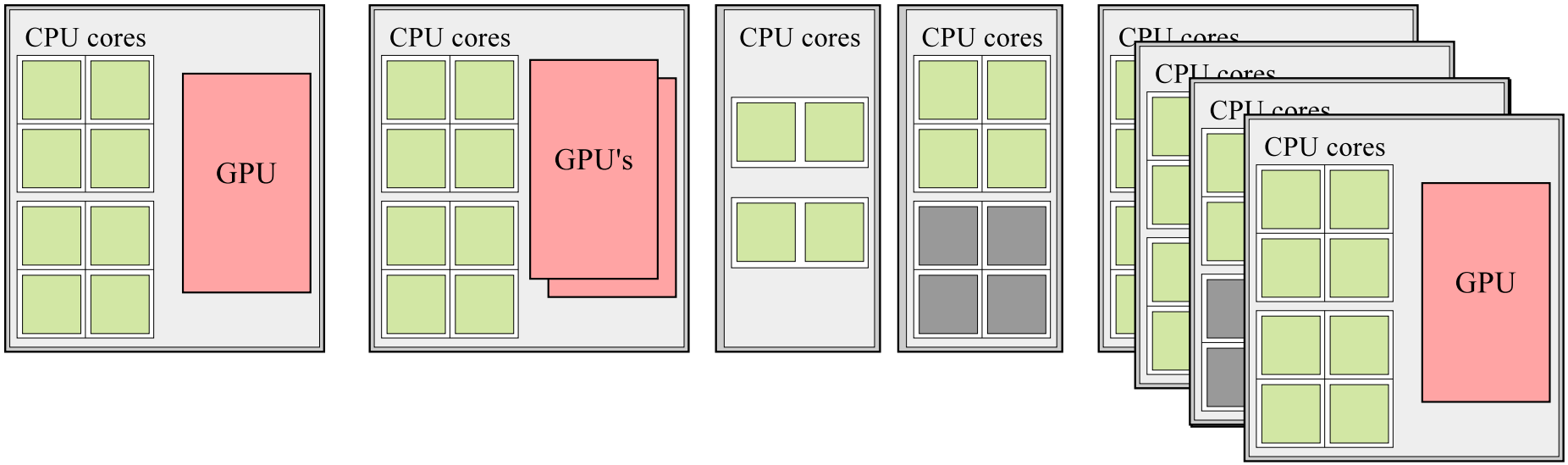
[1] CSI, University College Dublin, Ireland

[2] INESC-ID, IST/TULisbon, Lisbon, Portugal

Euro-Par 2012

INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa


Target Hierarchical Heterogeneous Platform



- Heterogeneity between devices within a node
- Heterogeneity between nodes
- eg. Grid'5000 Grenoble site.

Problem Statement: Matrix Partitioning

Want to perform matrix algebra in parallel on hierarchical platform.
For example:

- Matrix multiplication
- LU decomposition
- Jacobi iterative method
- ...

How to optimally partition matrix?

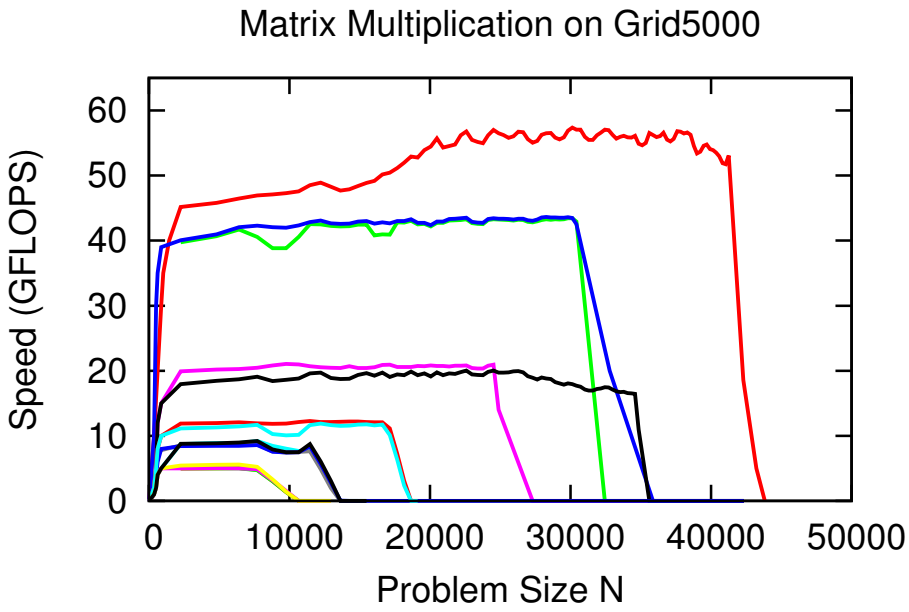
- Partition matrix between nodes
- Sub-partition between devices within a node
- To achieve load balancing, partition with respect to device and node speed.
- Minimise total volume of communication.

Traditional load Balancing

- Traditionally, processor performance is defined by a constant number.
- Computational units are partitioned as $d_i = N \times \frac{s_i}{\sum_{j=1}^{p} s_j}$.
- Speed is computed from clock speed, number of cores or by performing a benchmark.

Traditional load Balancing

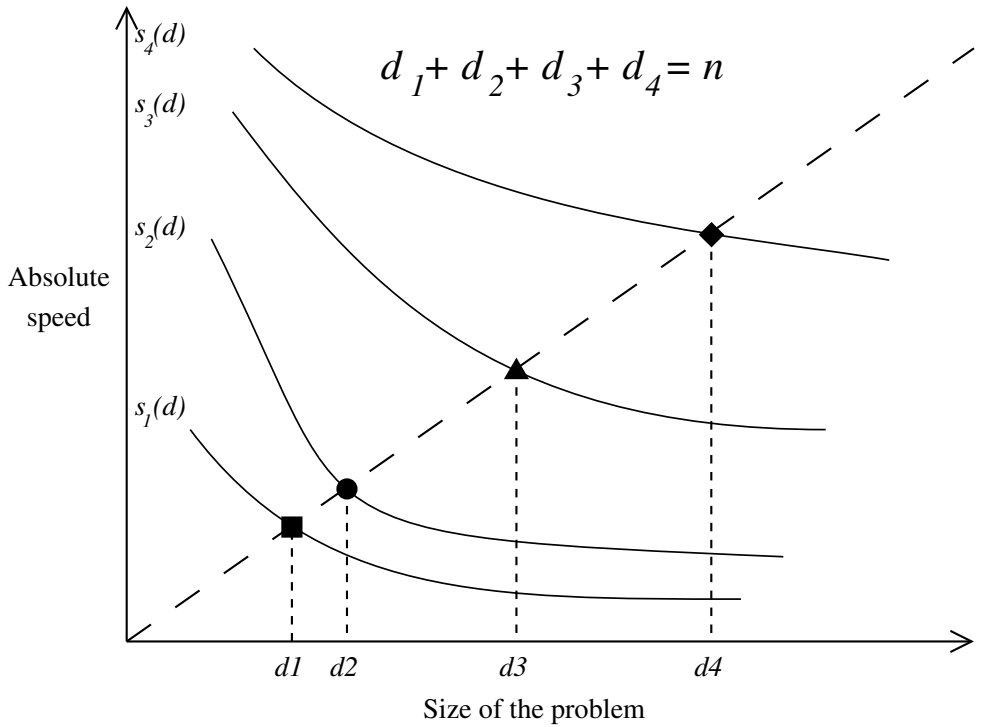
- Traditionally, processor performance is defined by a constant number.
- Computational units are partitioned as $d_i = N \times \frac{s_i}{\sum_{j=1}^{p} s_j}$.
- Speed is computed from clock speed, number of cores or by performing a benchmark.



- In reality, speed is a function of problem size.
- Algorithms based on constant performance models are only applicable for limited problem sizes.

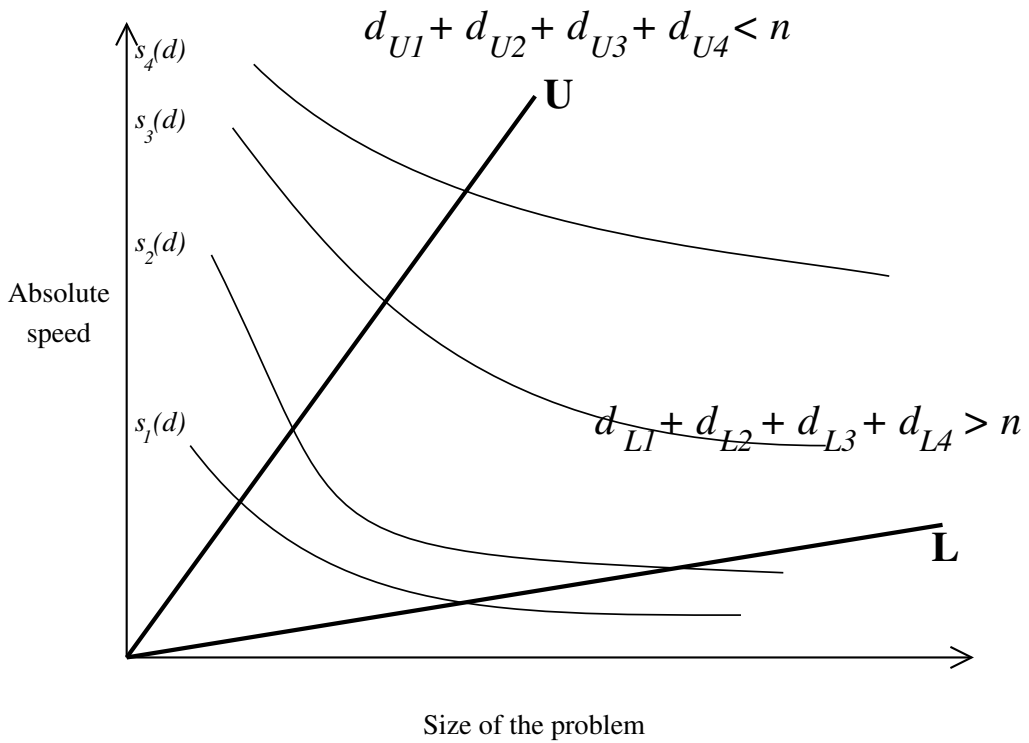
Partitioning with functional performance models

- Want all devices to compute assigned workload d_i within same time.
- Points $\left(d_i, s_i(d_i)\right)$ lie on a line passing through the origin when $\frac{d_i}{s_i(d_i)} = constant$.
- Total problem size determines the slope.
- Algorithm iteratively bisects solution space to find values d_i.



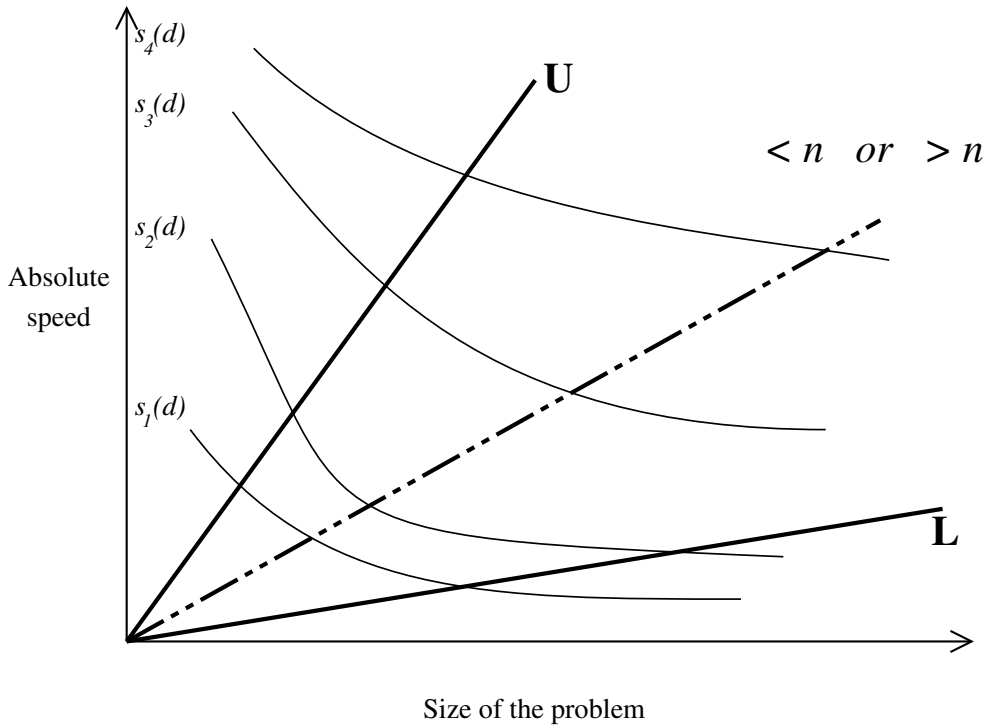
Partitioning with functional performance models

- Want all devices to compute assigned workload d_i within same time.
- Points $\left(d_i, s_i(d_i)\right)$ lie on a line passing through the origin when $\frac{d_i}{s_i(d_i)} = constant$.
- Total problem size determines the slope.
- Algorithm iteratively bisects solution space to find values d_i.



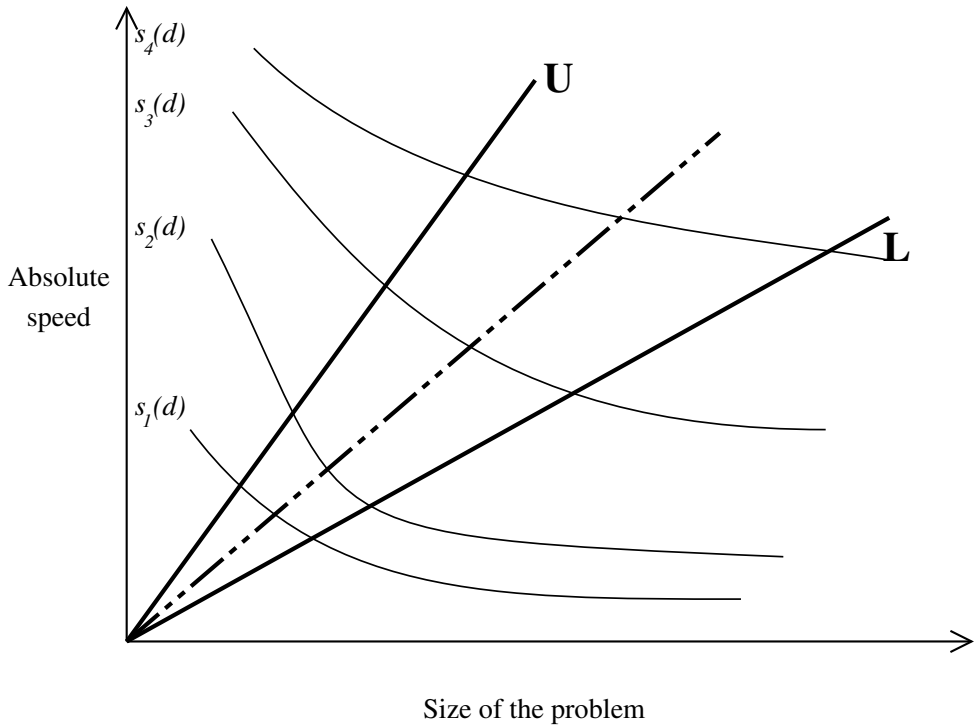
Partitioning with functional performance models

- Want all devices to compute assigned workload d_i within same time.
- Points $\left(d_i, s_i(d_i)\right)$ lie on a line passing through the origin when $\frac{d_i}{s_i(d_i)} = constant$.
- Total problem size determines the slope.
- Algorithm iteratively bisects solution space to find values d_i.



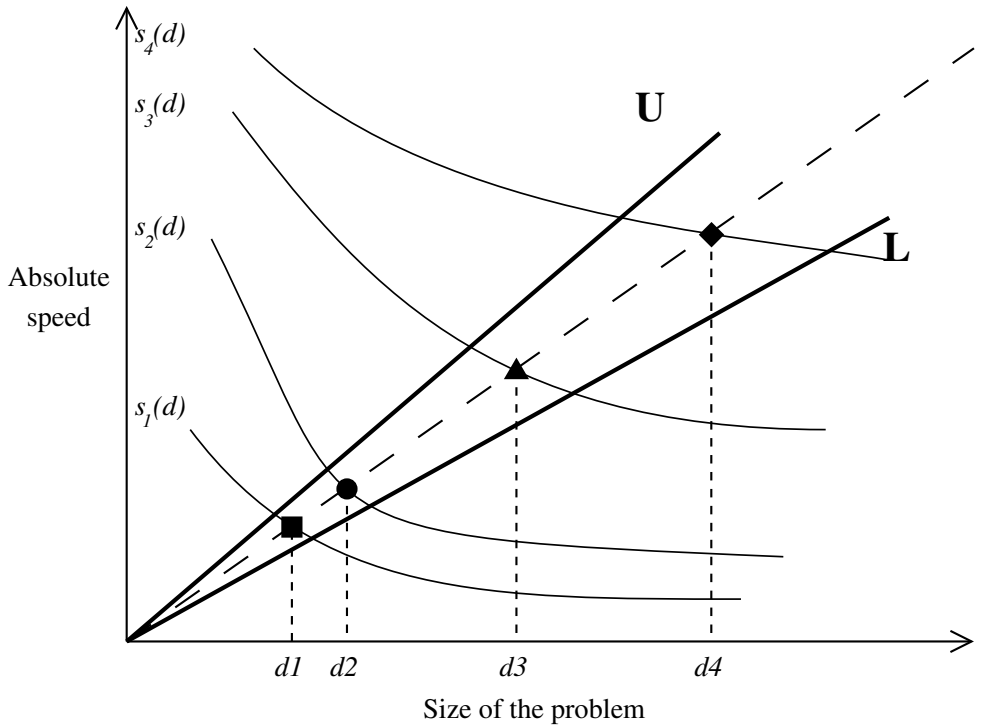
Partitioning with functional performance models

- Want all devices to compute assigned workload d_i within same time.
- Points $\left(d_i, s_i(d_i)\right)$ lie on a line passing through the origin when $\frac{d_i}{s_i(d_i)} = constant$.
- Total problem size determines the slope.
- Algorithm iteratively bisects solution space to find values d_i.



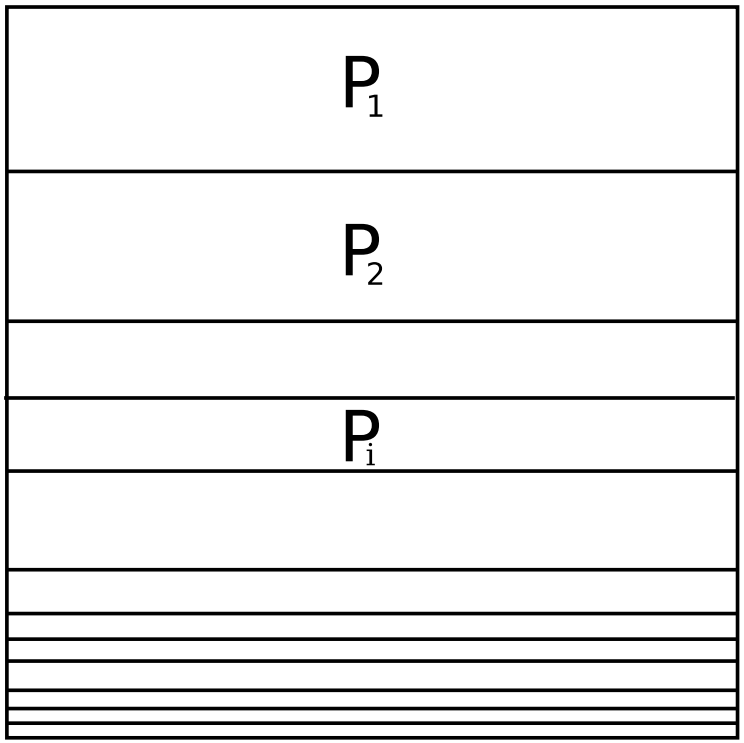
Partitioning with functional performance models

- Want all devices to compute assigned workload d_i within same time.
- Points $\left(d_i, s_i(d_i)\right)$ lie on a line passing through the origin when $\frac{d_i}{s_i(d_i)} = constant$.
- Total problem size determines the slope.
- Algorithm iteratively bisects solution space to find values d_i.

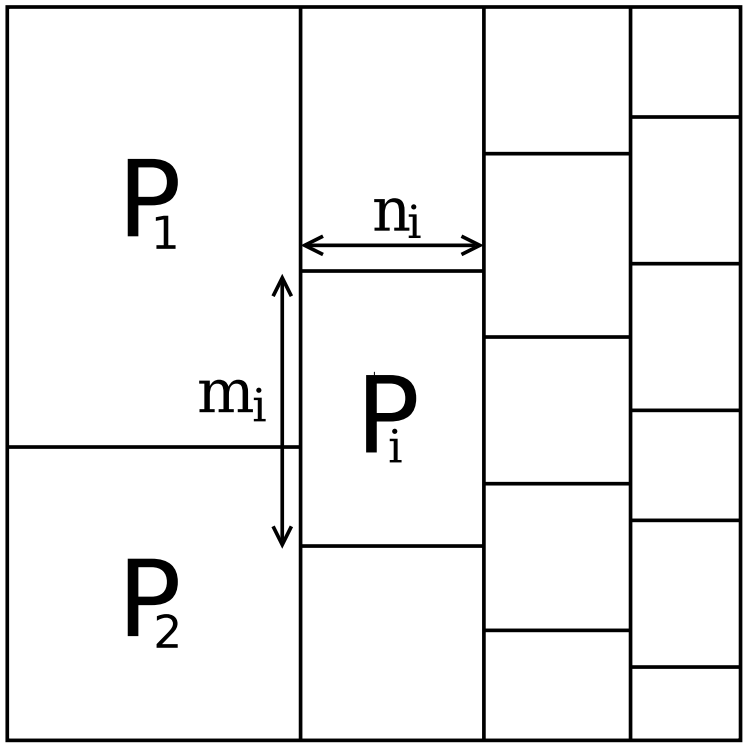


Matrix Partitioning

Simple Partitioning



2D Partitioning

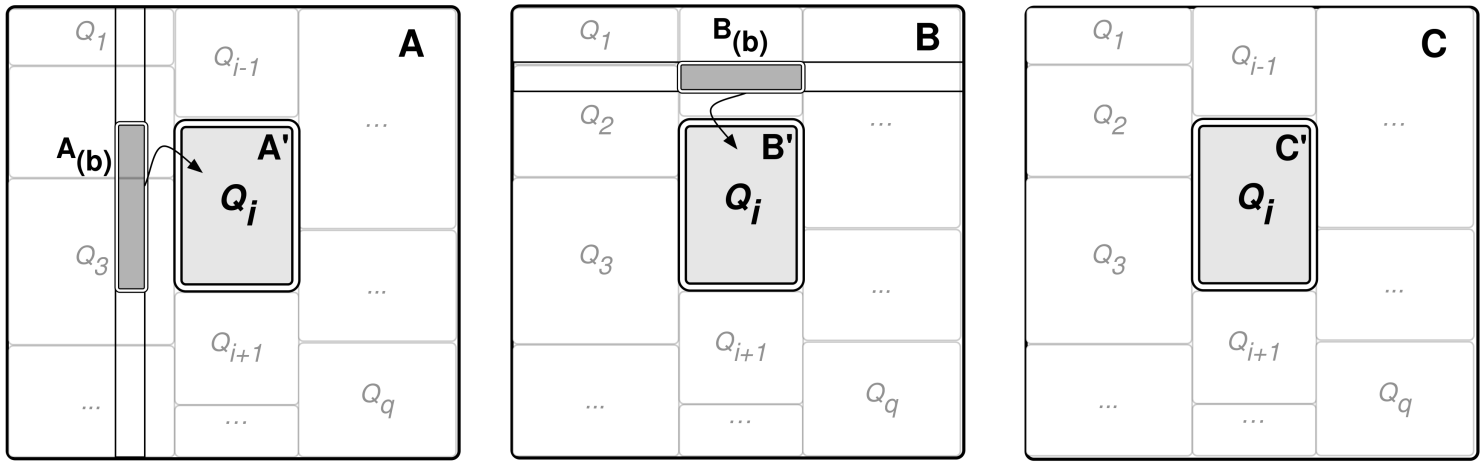


2D matrix partitioning from 1D partitioning algorithm

- Height m_i and width n_i combined into one parameter, area $w_i = m_i \times n_i$.
- Square areas are benchmarked $m = n = \sqrt{w}$.
- 1D partitioning algorithm finds area rectangles.
- Communication minimising algorithm computes ordering and shape of these rectangles.

Minimising Total Communication Volume

- Application specific
- For example: Matrix multiplication
  communication minimised by minimising $\sum_i^p (m_i + n_i)$



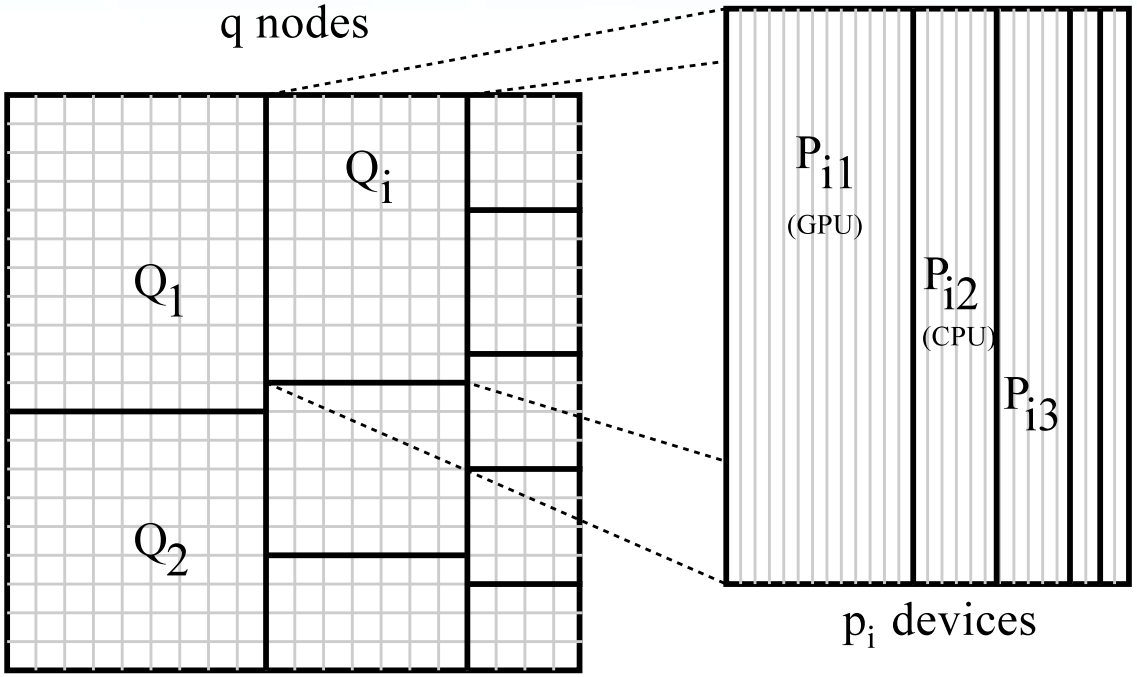
- Call communication minimising algorithm* (CMA) to compute:
  - Optimum number of columns
  - Optimum number of processors in each column

* Beaumont, O., Boudet, V., Rastello, F., Robert, Y.: Matrix Multiplication on Heterogeneous Platforms. IEEE Trans. Parallel Distrib. Syst. 12(10), 1033–1051 (2001)

Hierarchical Partitioning Algorithm

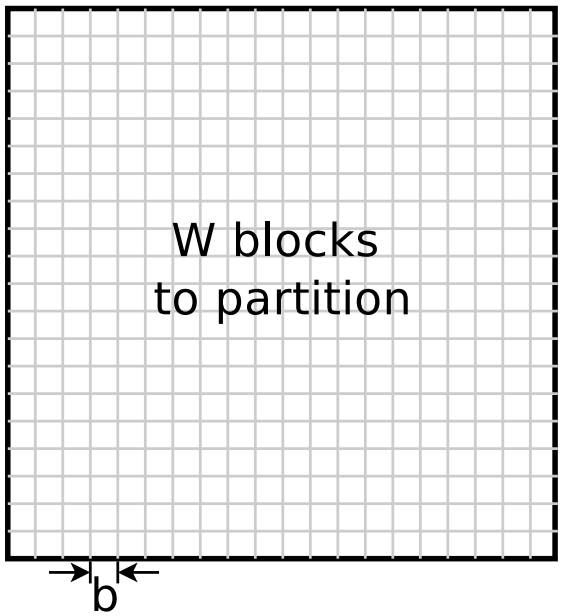
- Dynamic algorithm - no a priori information about performance required.
- Inputs:
  - Problem size
  - Number of nodes
  - Number of devices per node
  - Device type (eg. cpu, gpu, . . .).
- Link computational kernel to be benchmarked for each device.
- Initially distribution is partitioned evenly between nodes and between devices within a node
- Algorithm converges towards optimum solution

Hierarchical Partitioning Algorithm



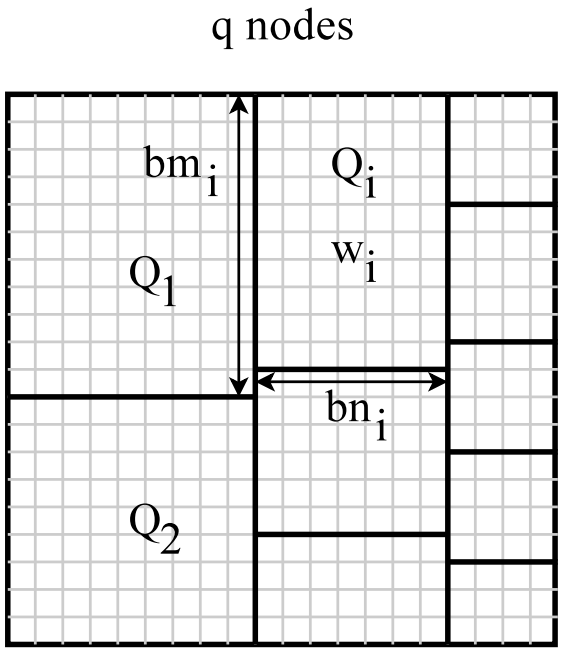
- q nodes, $Q_1, \ldots, Q_q$.
- node Q_i has p_i devices, $P_{i1}, \ldots, P_{ip_i}$
- Hierarchy in platform $\rightarrow$ hierarchy in partitioning
  - Nested parallelism
  - inter-node partitioning algorithm (INPA)
  - inter-device partitioning algorithm (IDPA)
  - IDPA is nested inside INPA

Hierarchical Partitioning Algorithm



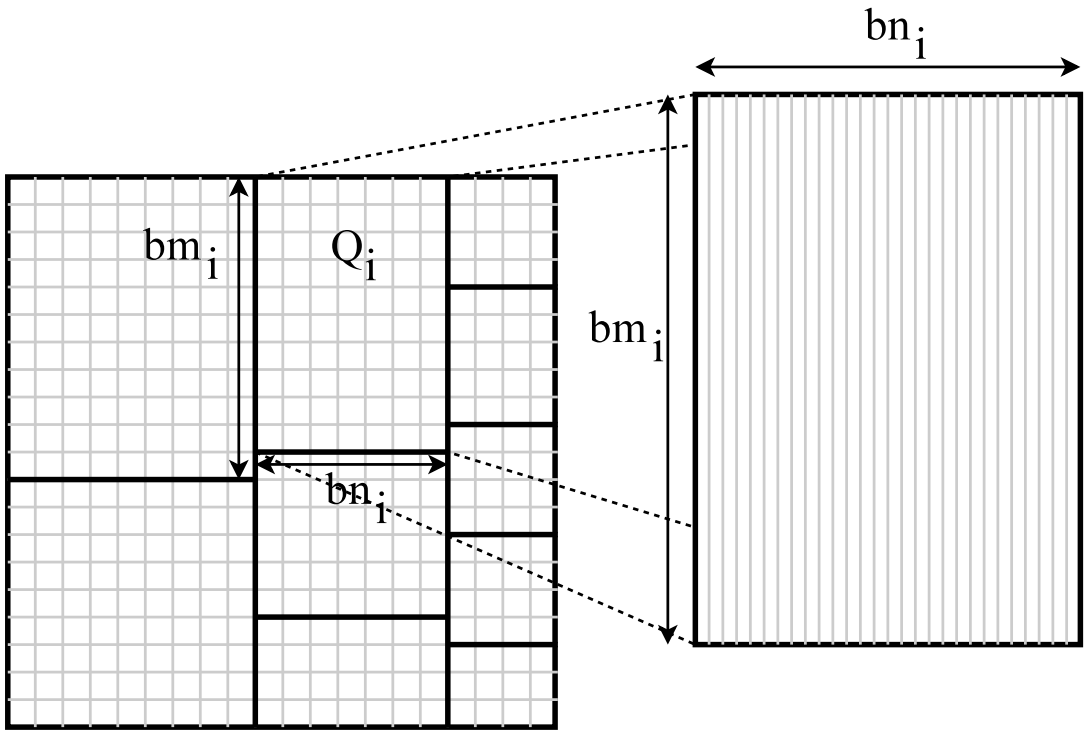
- W computational units to partition between nodes
- inter-node partitioning algorithm (INPA) creates node-FPM's and computes $w_1, \ldots, w_q$ so that $w_1 + \ldots + w_q = W$.

Hierarchical Partitioning Algorithm



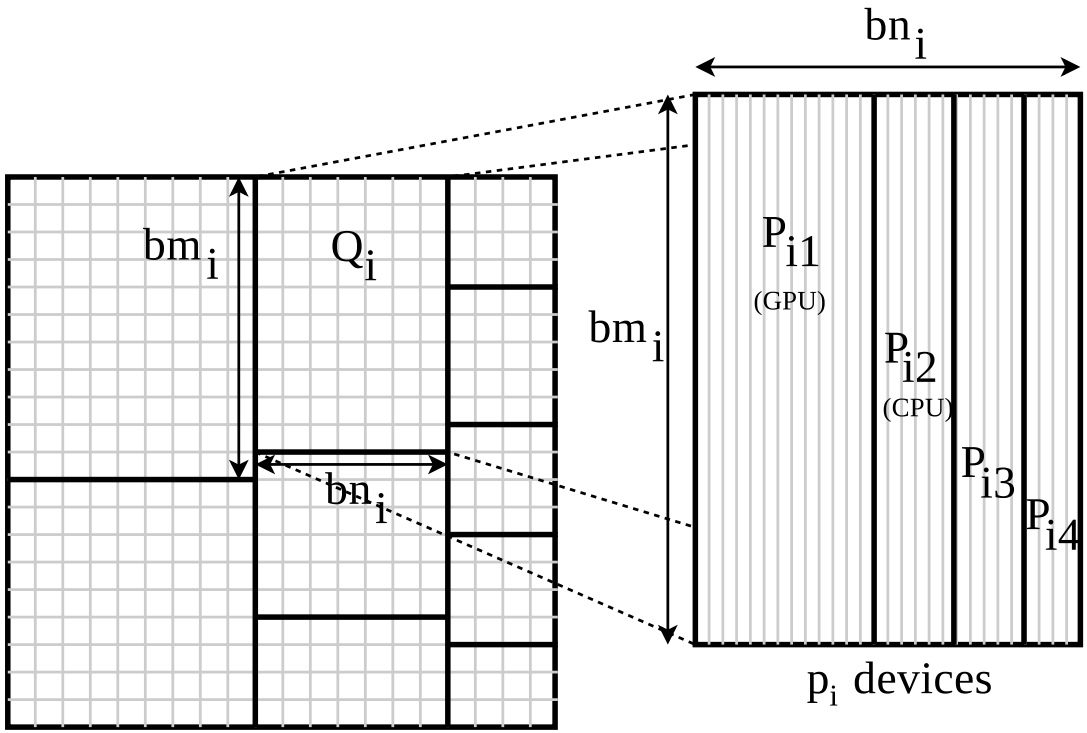
- Communication minimising algorithm has input: $w_1, \ldots, w_q$ and output: $(m_1, n_1), \ldots, (m_q, n_q)$ such that $m_i \times n_i = w_i$ and matrix is completely tiled.

Hierarchical Partitioning Algorithm

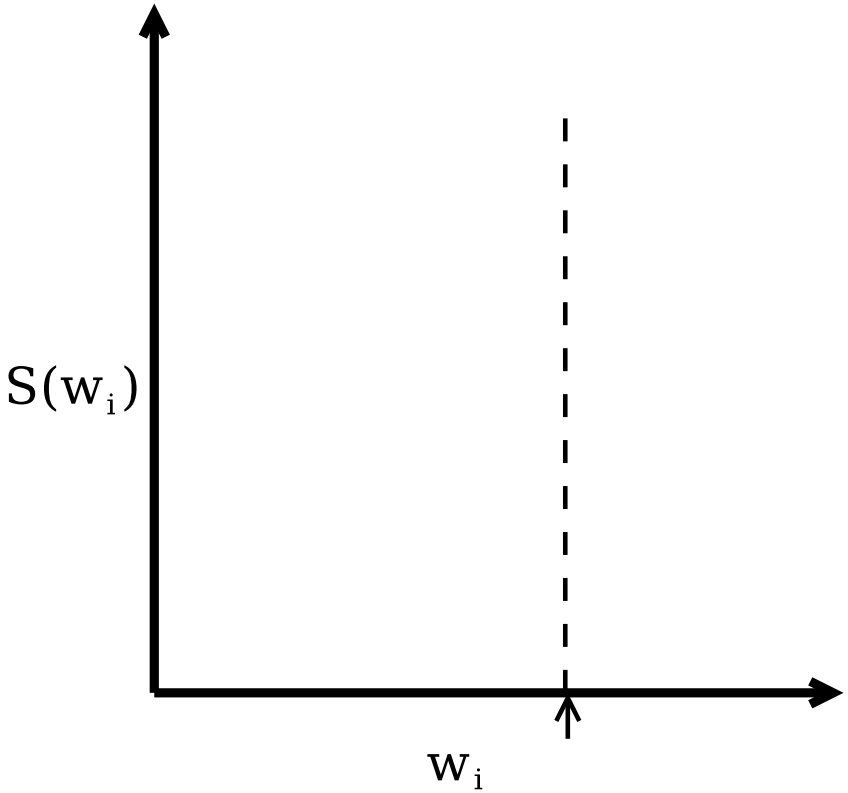


- inter-device partitioning algorithm (IDPA) creates device-FPM's and computes $d_{i1}, \ldots, d_{ip}$, such that $d_{i1} + \ldots + d_{ip} = bn_i$

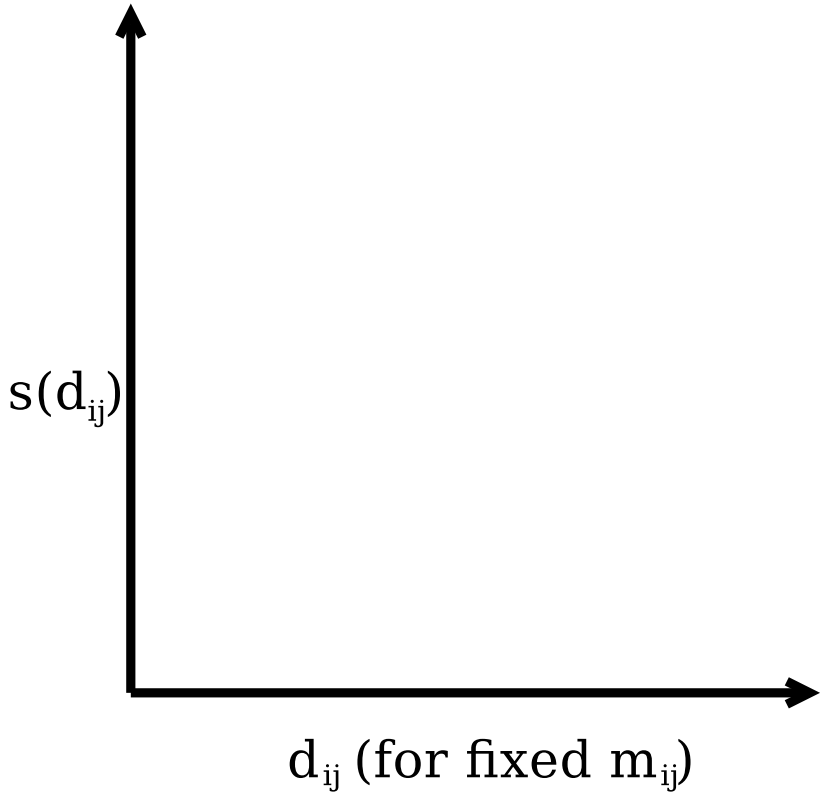
Hierarchical Partitioning Algorithm



- inter-device partitioning algorithm (IDPA) creates device-FPM's and computes $d_{i1}, \ldots, d_{ip}$,
  such that $d_{i1} + \ldots + d_{ip} = bn_i$



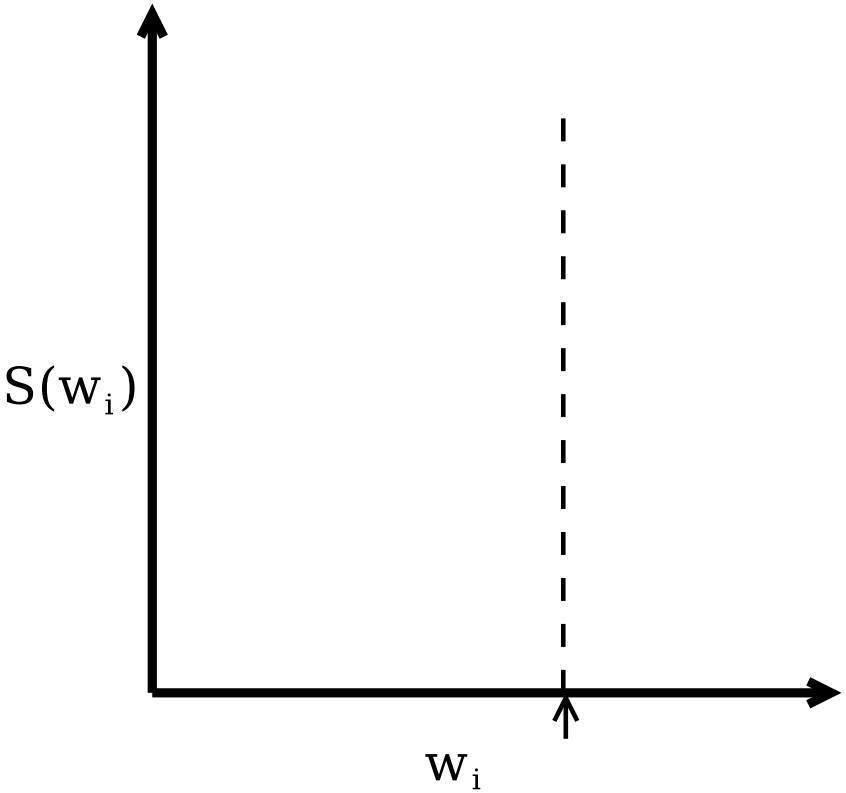
models of nodes



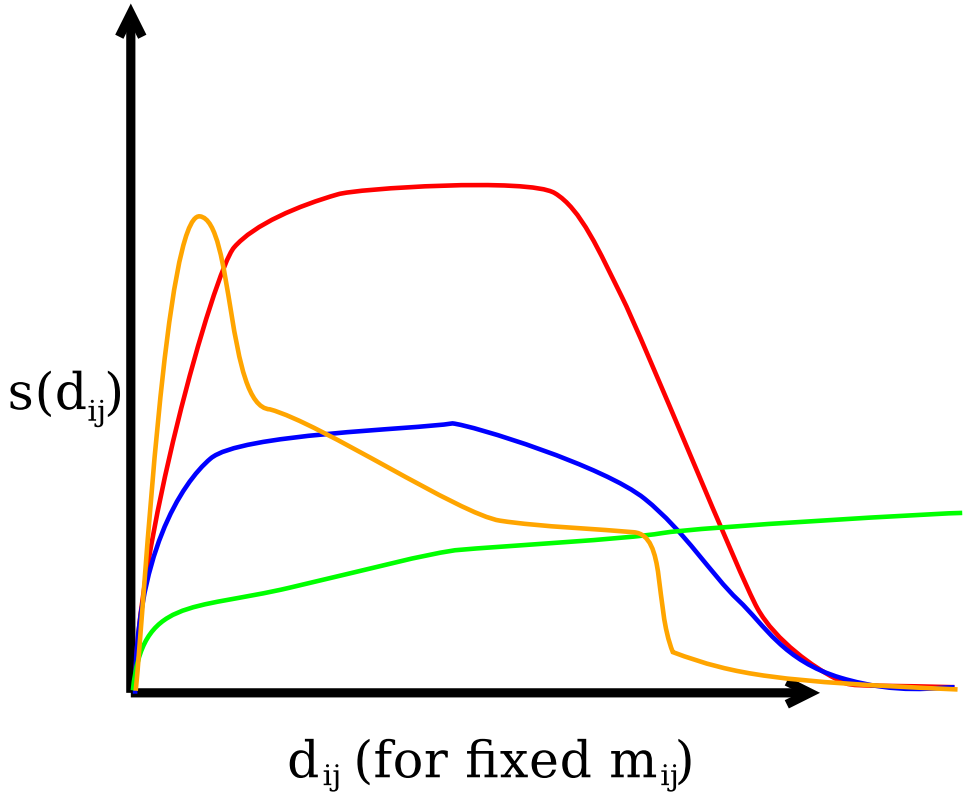
models of devices
(from 1 node)

$$w_i = m_i \times n_i$$

$$\sum_{j=1}^{p} d_{ij} = b \times n_i$$



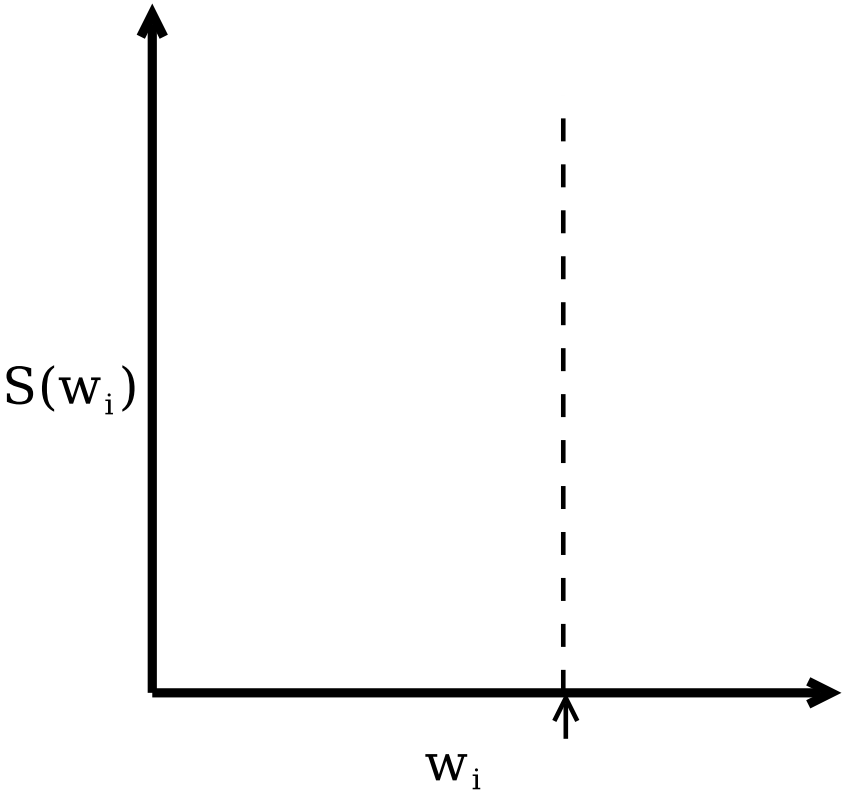
models of nodes



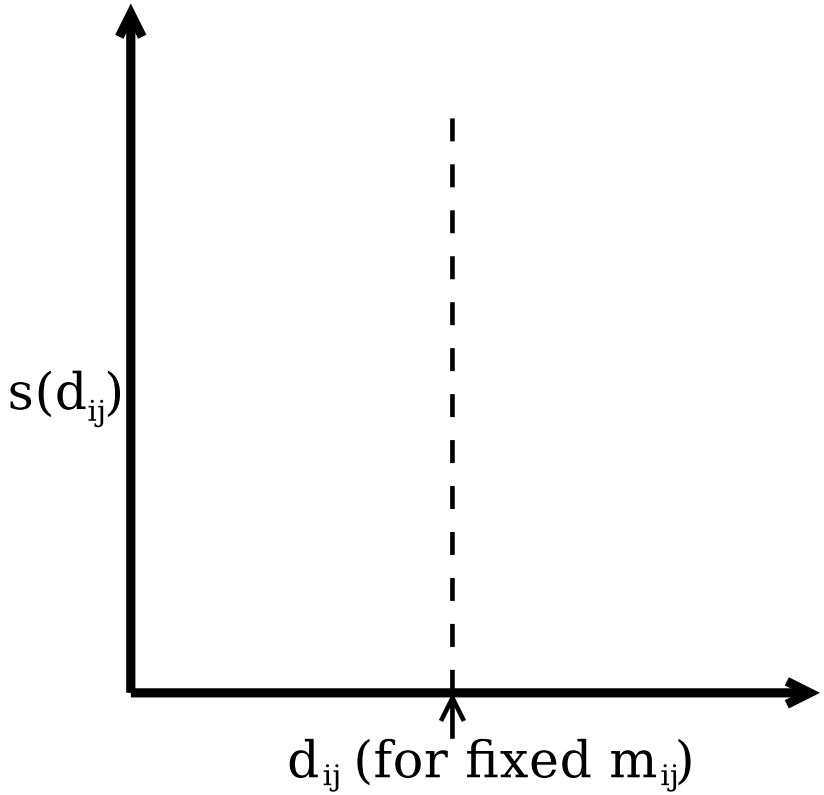
models of devices
(from 1 node)

$$w_i = m_i \times n_i$$

$$\sum_{j=1}^{p} d_{ij} = b \times n_i$$



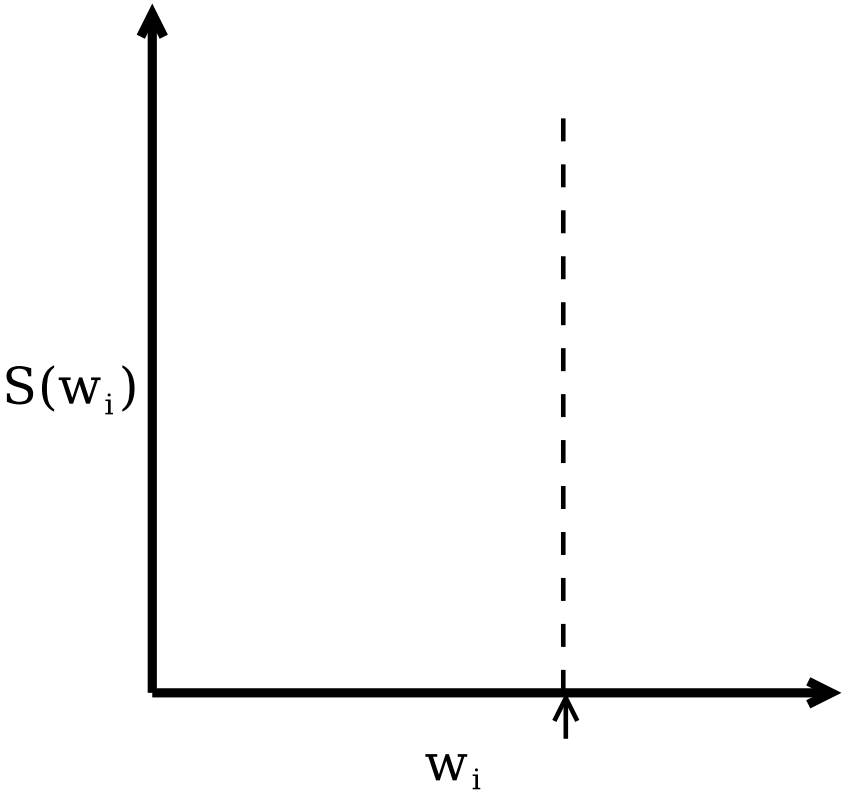
models of nodes



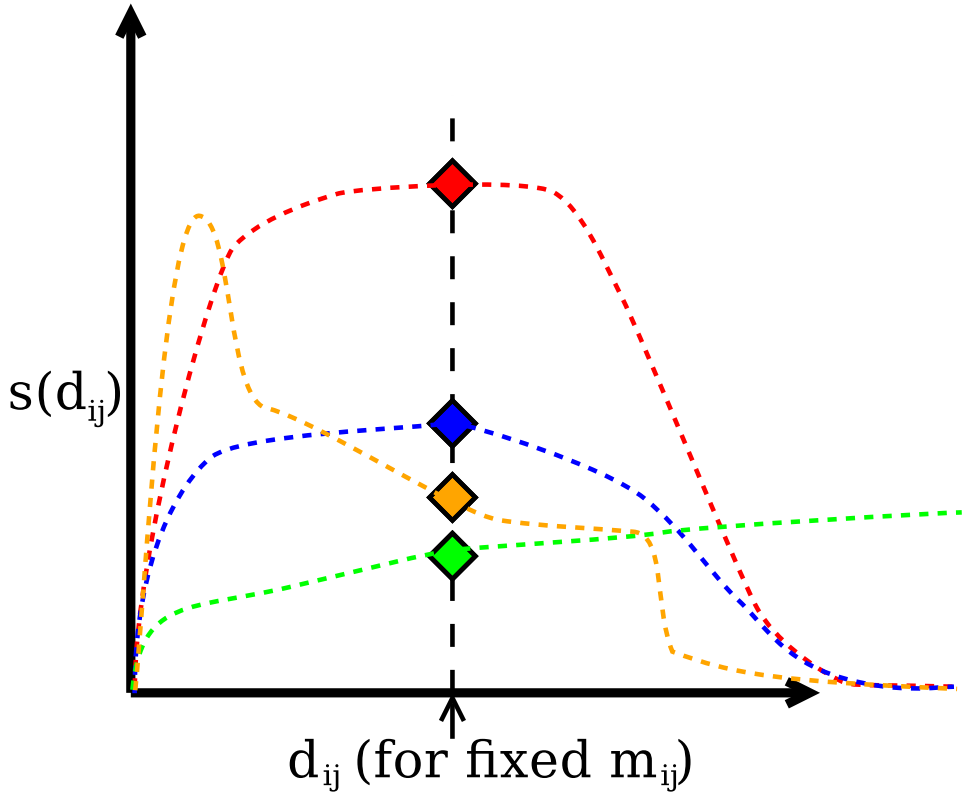
models of devices
(from 1 node)

$$w_i = m_i \times n_i$$

$$\sum_{j=1}^{p} d_{ij} = b \times n_i$$



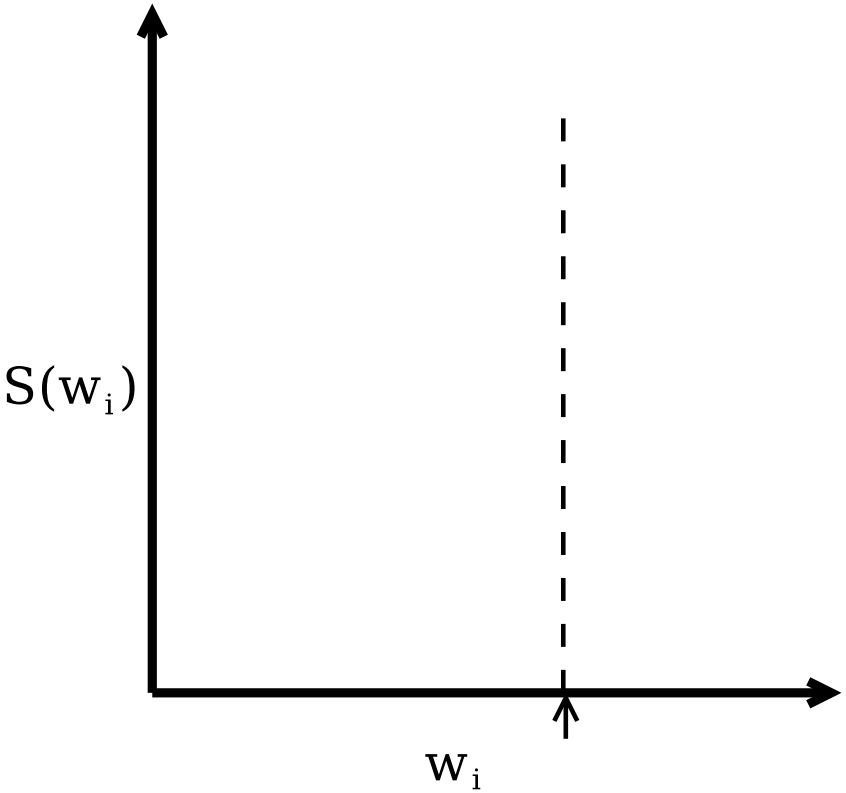
models of nodes



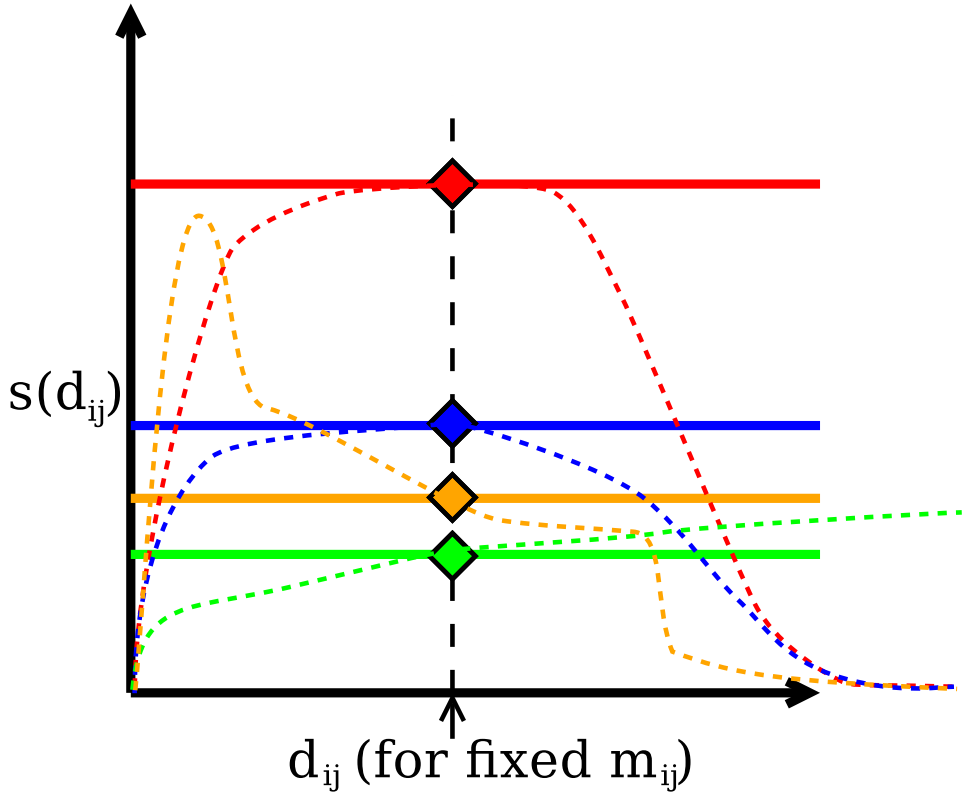
models of devices
(from 1 node)

$$w_i = m_i \times n_i$$

$$\sum_{j=1}^{p} d_{ij} = b \times n_i$$



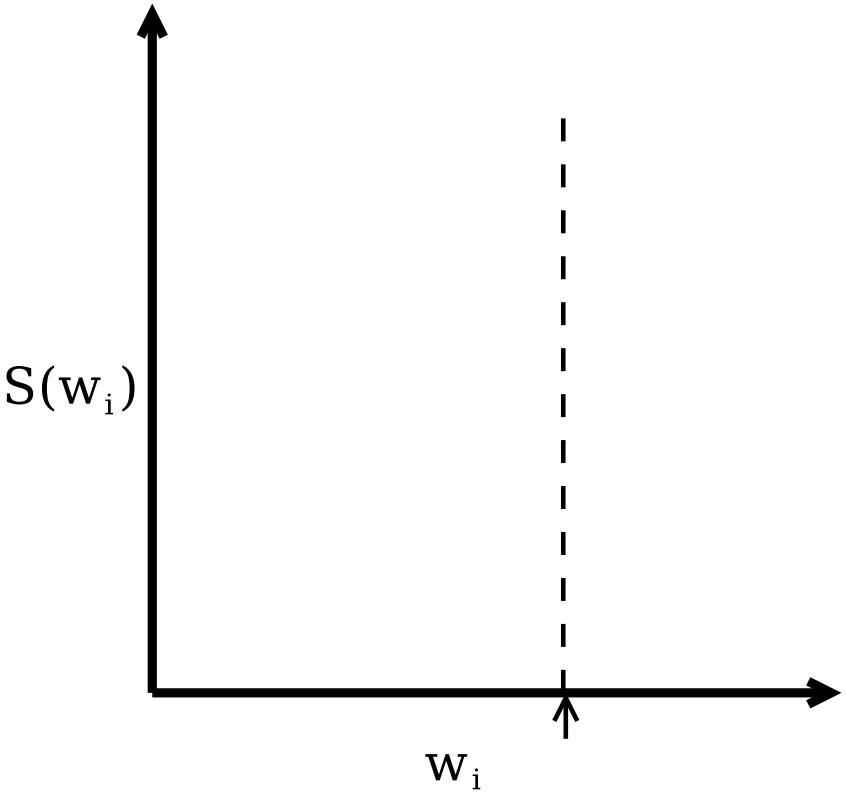
models of nodes



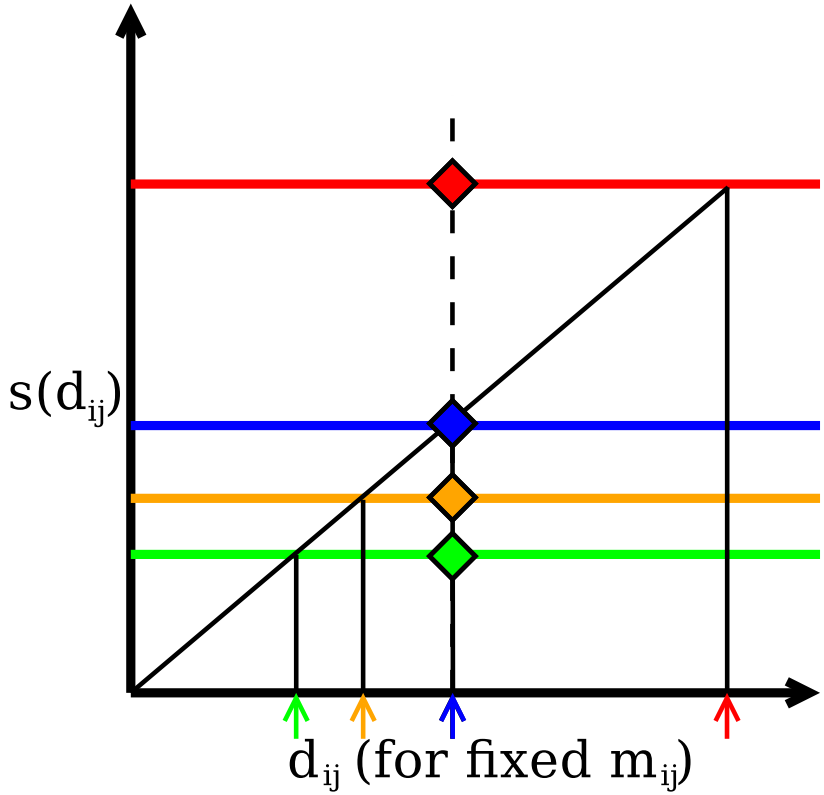
models of devices
(from 1 node)

$$w_i = m_i \times n_i$$

$$\sum_{j=1}^{p} d_{ij} = b \times n_i$$



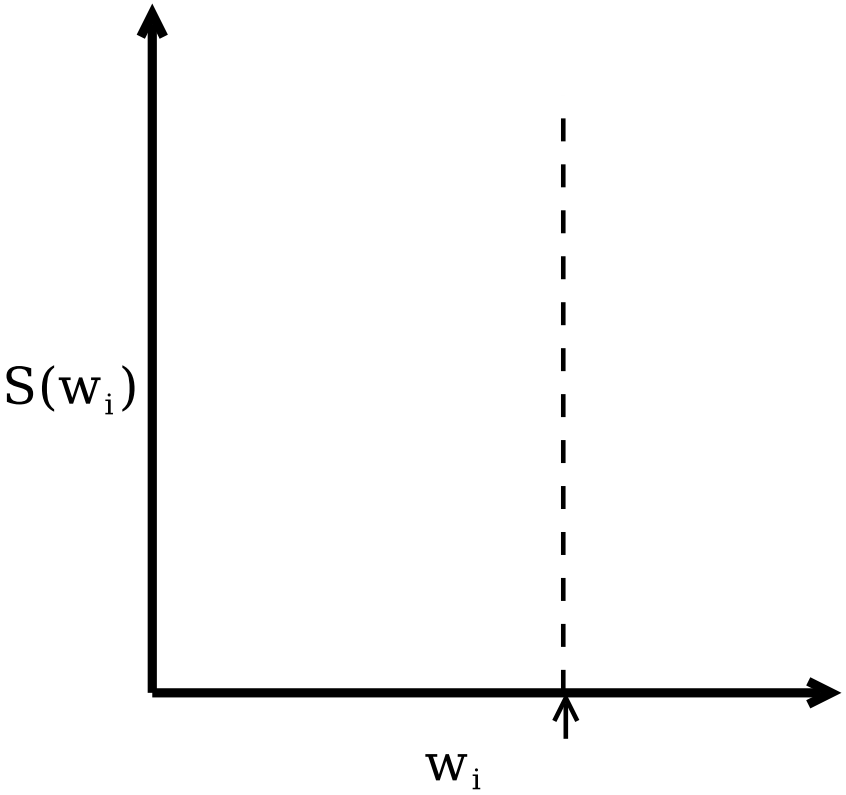
models of nodes



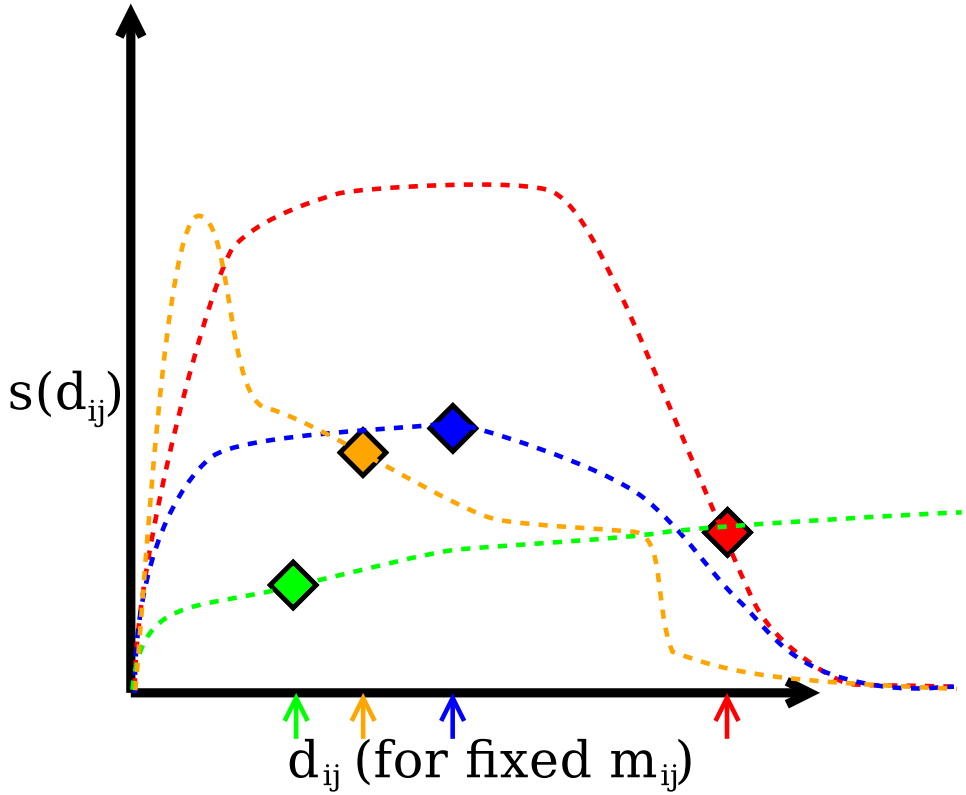
models of devices
(from 1 node)

$$w_i = m_i \times n_i$$

$$\sum_{j=1}^{p} d_{ij} = b \times n_i$$



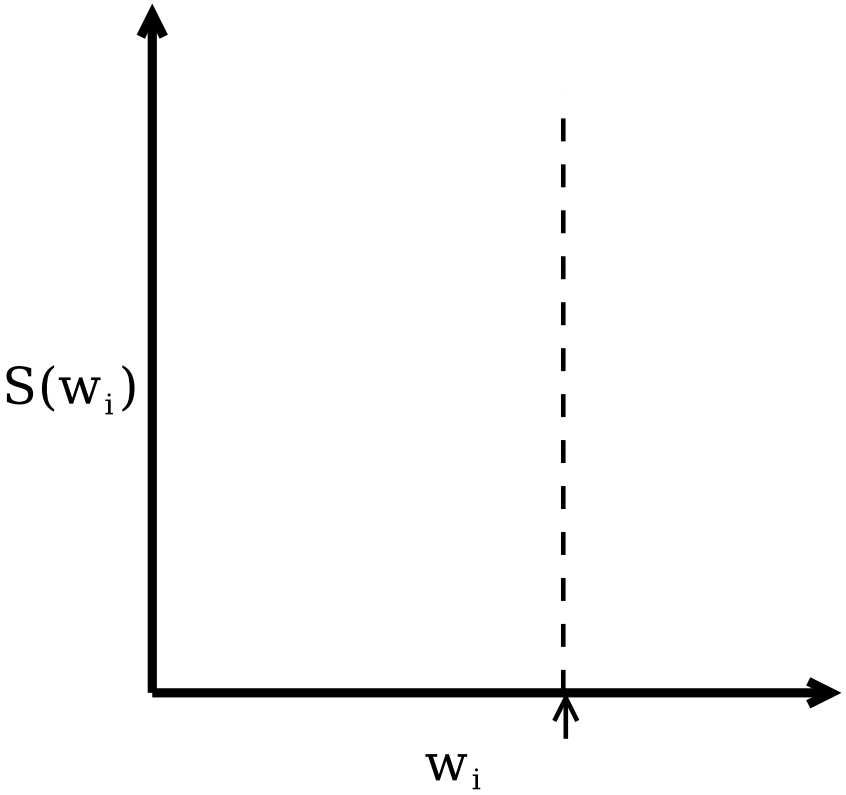
models of nodes



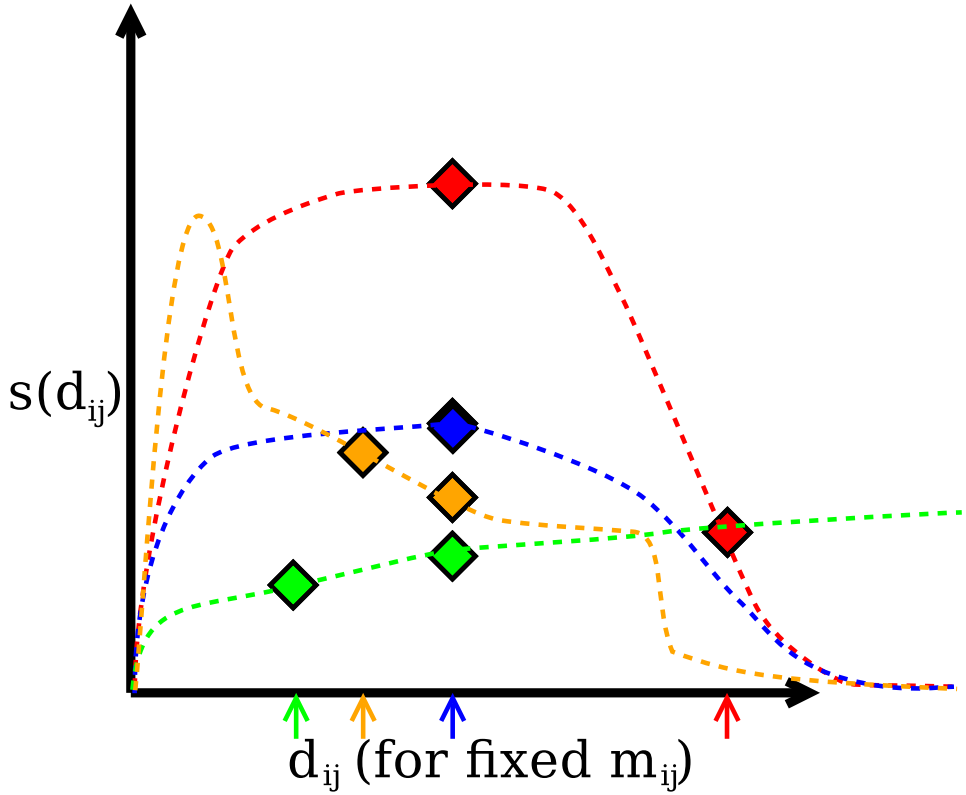
models of devices
(from 1 node)

$$w_i = m_i \times n_i$$

$$\sum_{j=1}^{p} d_{ij} = b \times n_i$$



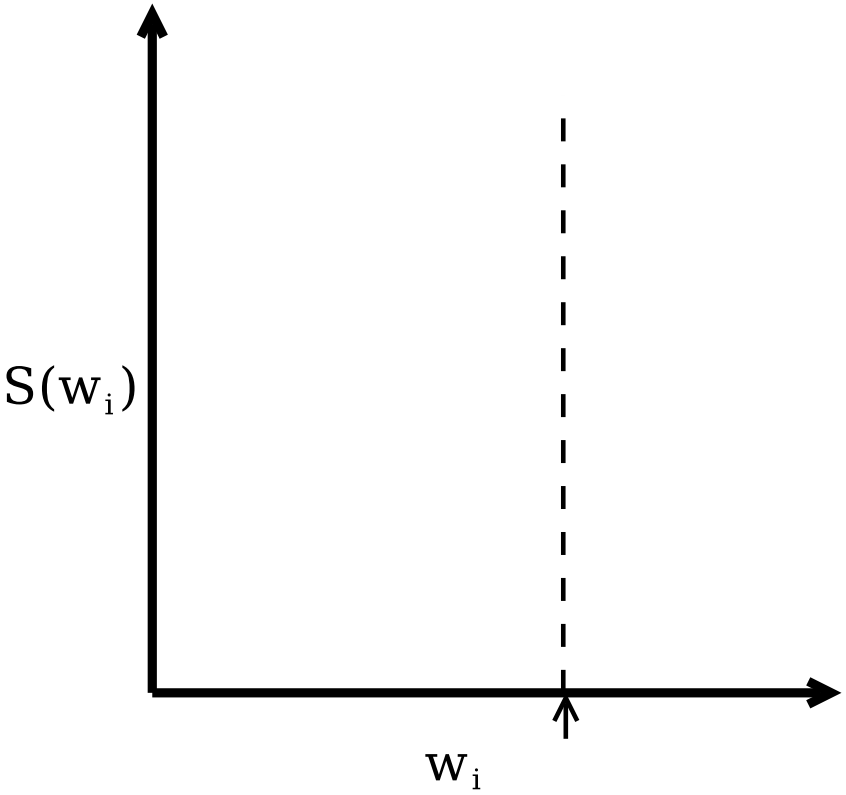
models of nodes



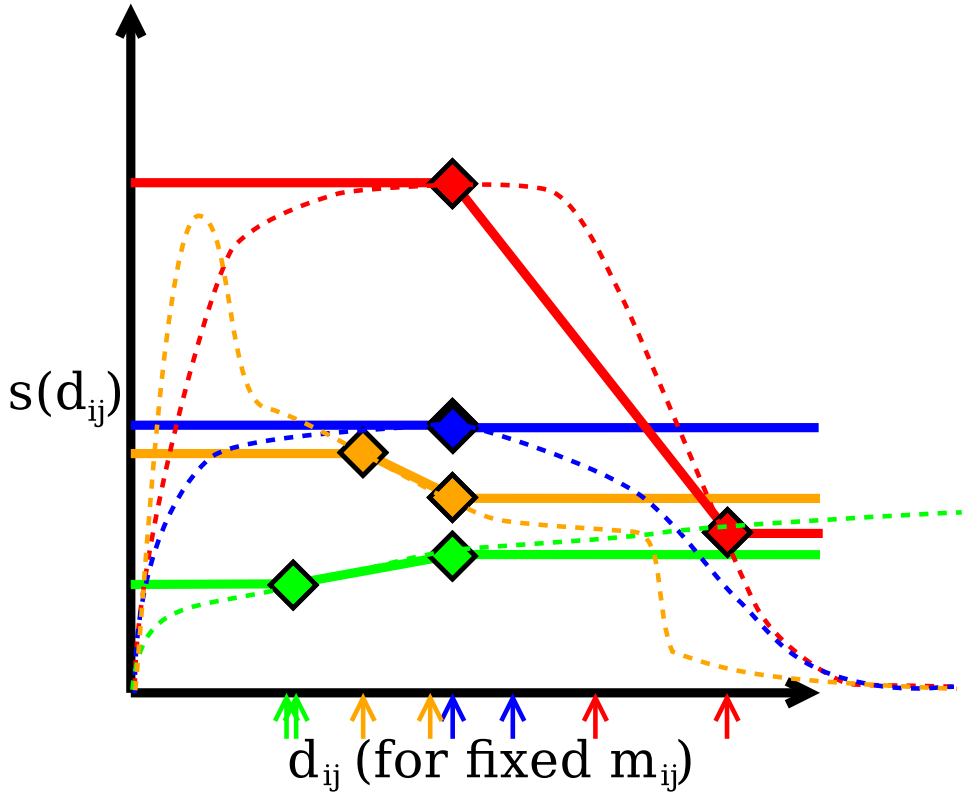
models of devices
(from 1 node)

$$w_i = m_i \times n_i$$

$$\sum_{j=1}^{p} d_{ij} = b \times n_i$$



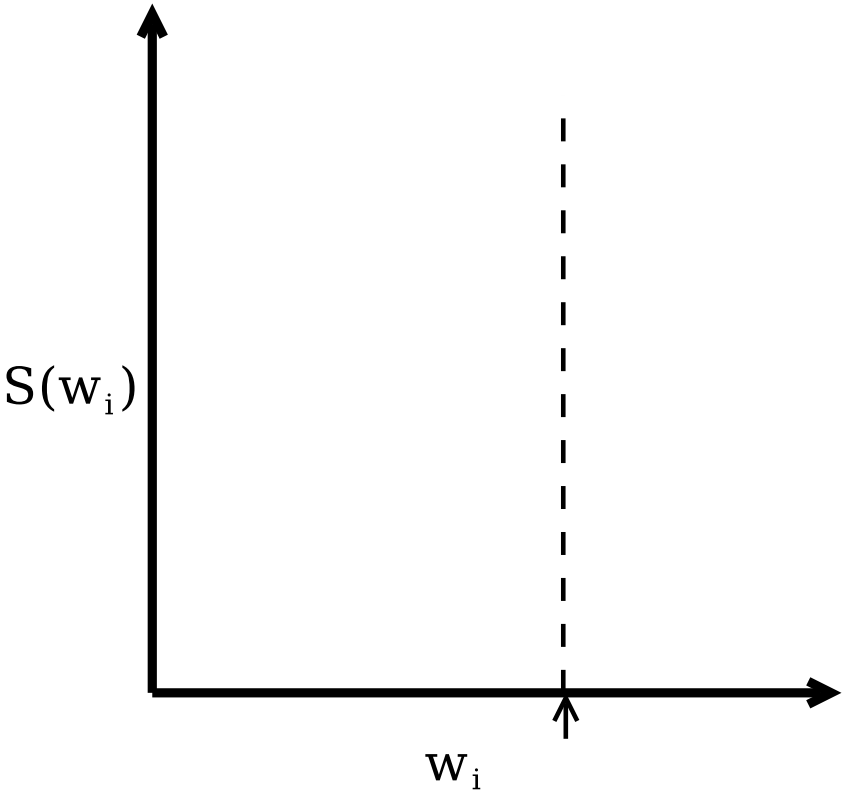
models of nodes



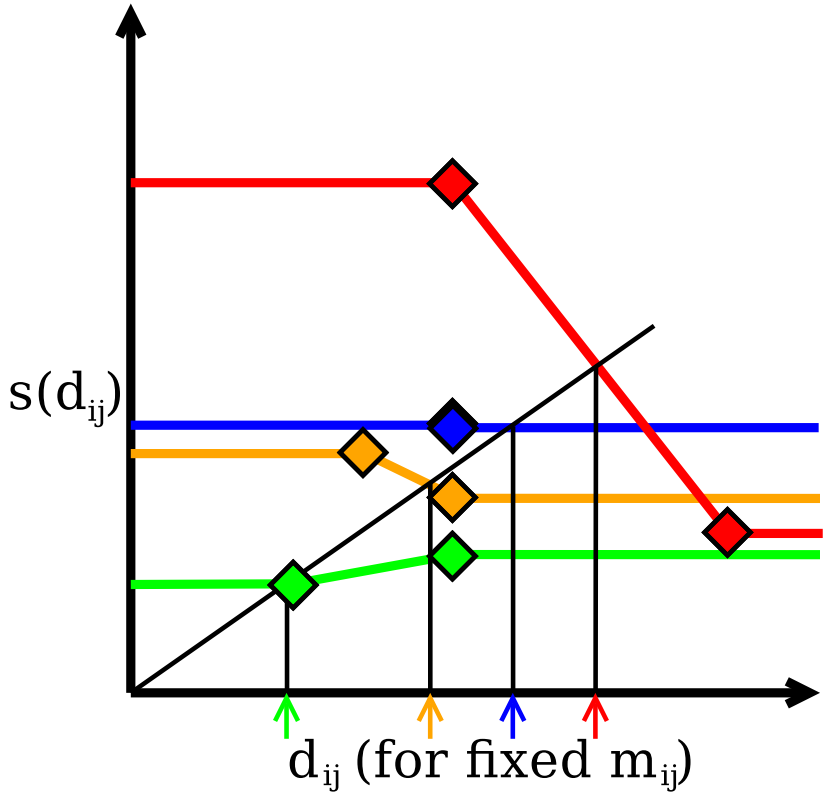
models of devices
(from 1 node)

$$w_i = m_i \times n_i$$

$$\sum_{j=1}^{p} d_{ij} = b \times n_i$$



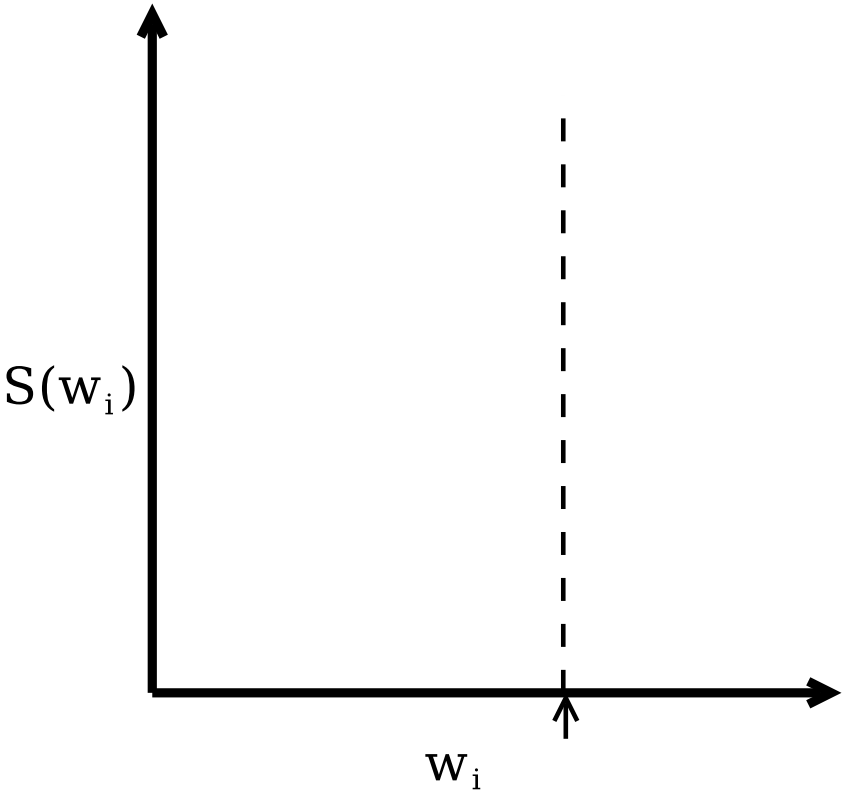
models of nodes



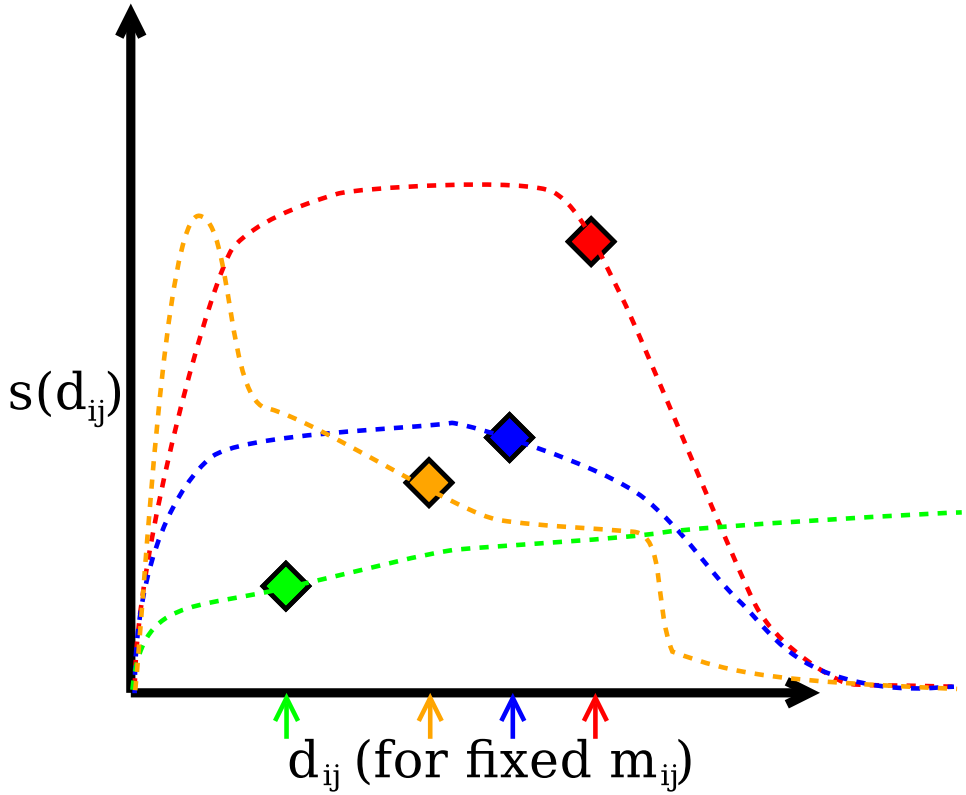
models of devices
(from 1 node)

$$w_i = m_i \times n_i$$

$$\sum_{j=1}^{p} d_{ij} = b \times n_i$$



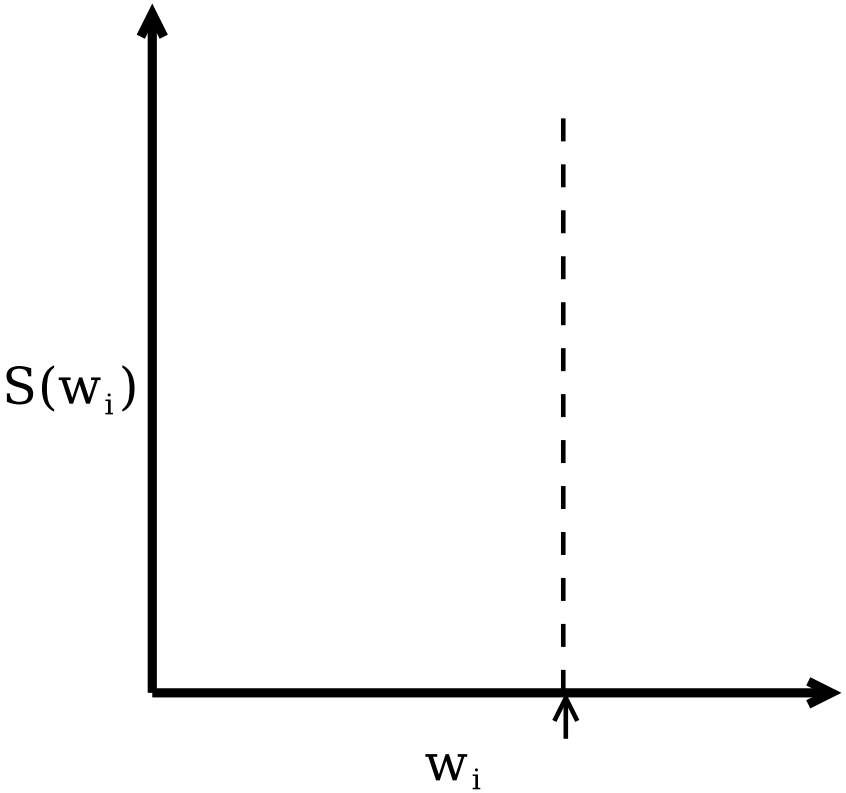
models of nodes



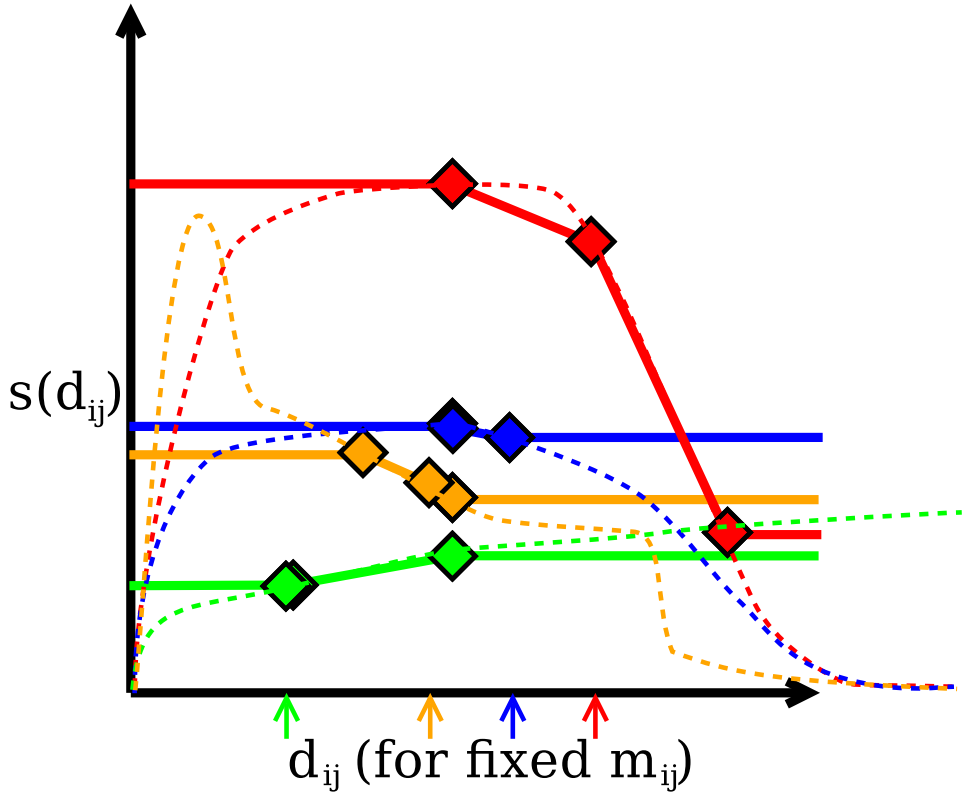
models of devices
(from 1 node)

$$w_i = m_i \times n_i$$

$$\sum_{j=1}^{p} d_{ij} = b \times n_i$$



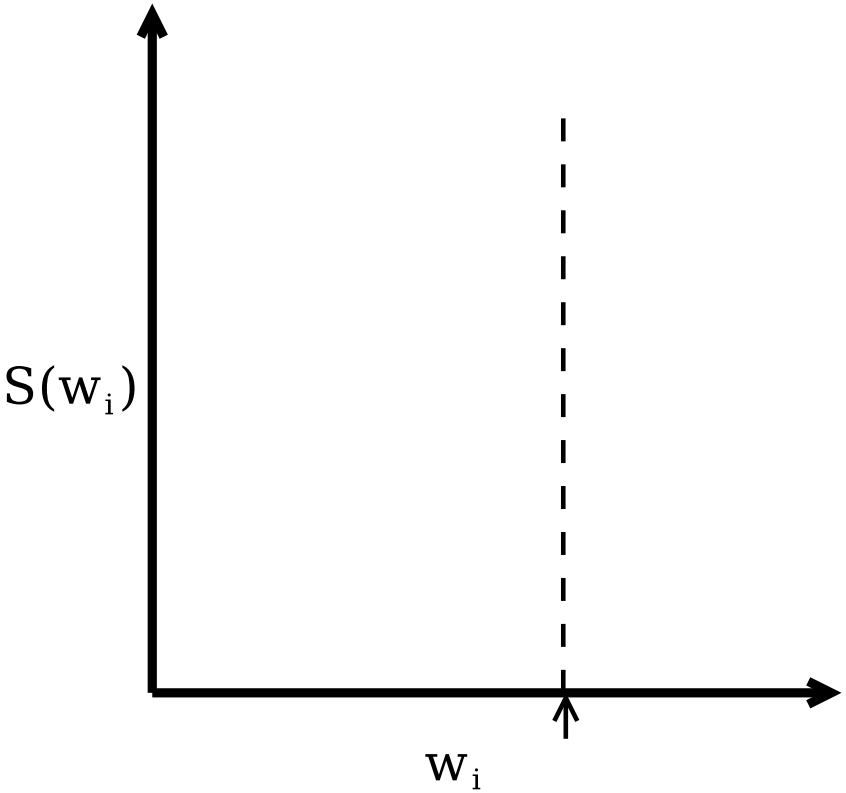
models of nodes



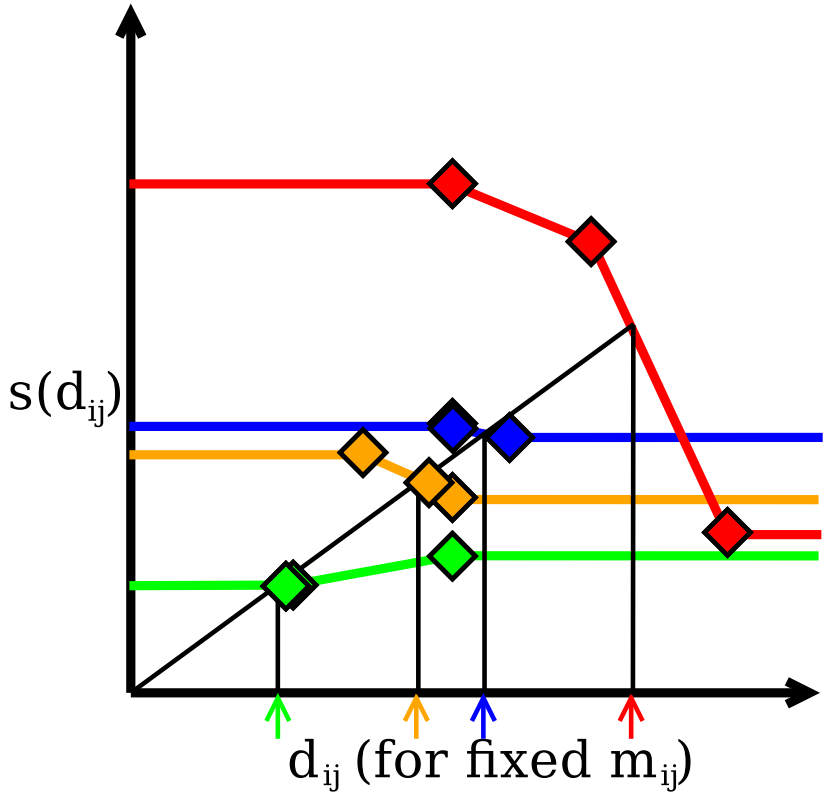
models of devices
(from 1 node)

$$w_i = m_i \times n_i$$

$$\sum_{j=1}^{p} d_{ij} = b \times n_i$$



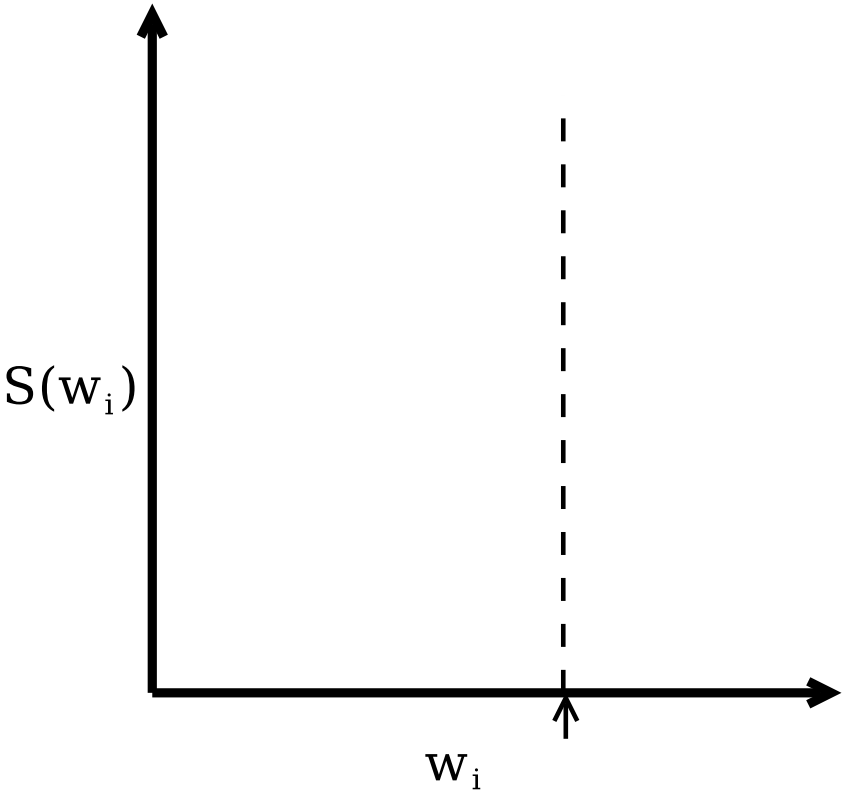
models of nodes



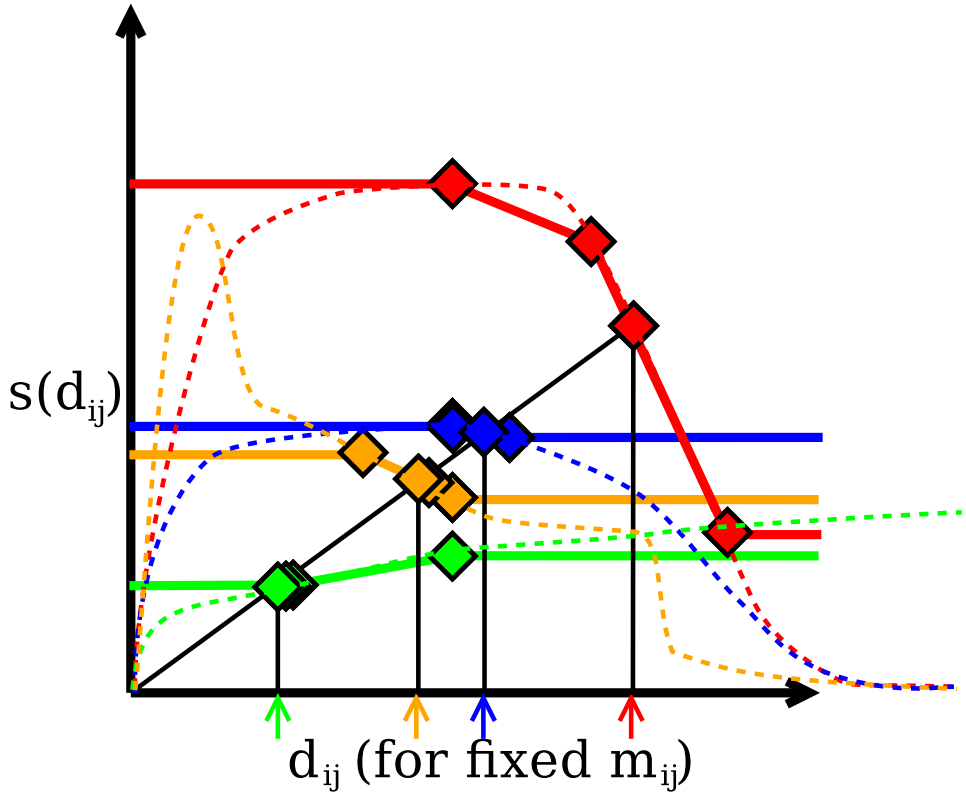
models of devices
(from 1 node)

$$w_i = m_i \times n_i$$

$$\sum_{j=1}^{p} d_{ij} = b \times n_i$$



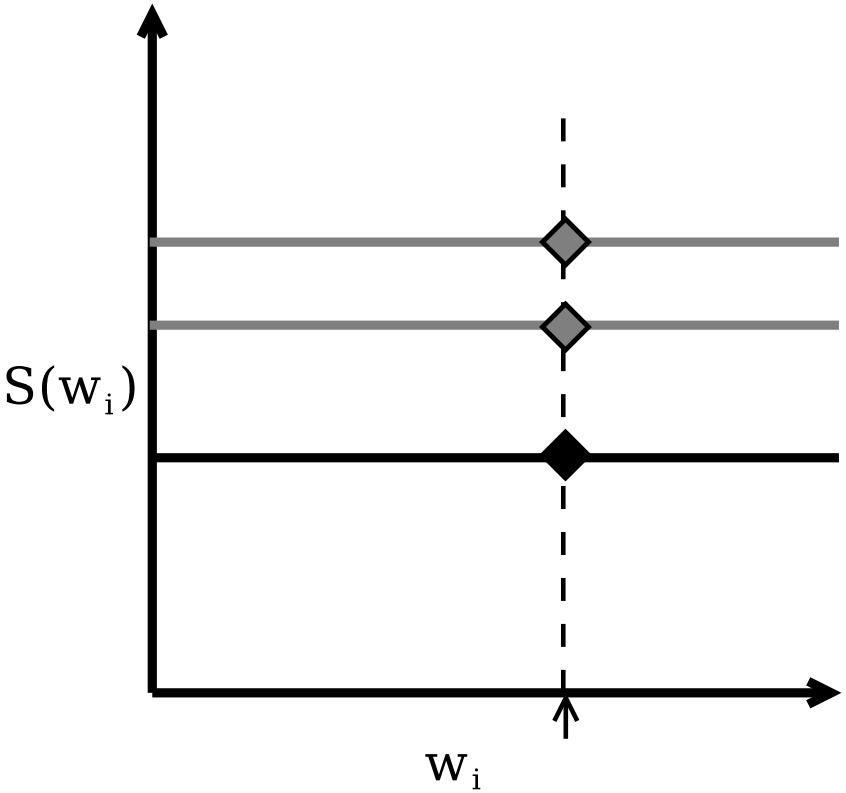
models of nodes



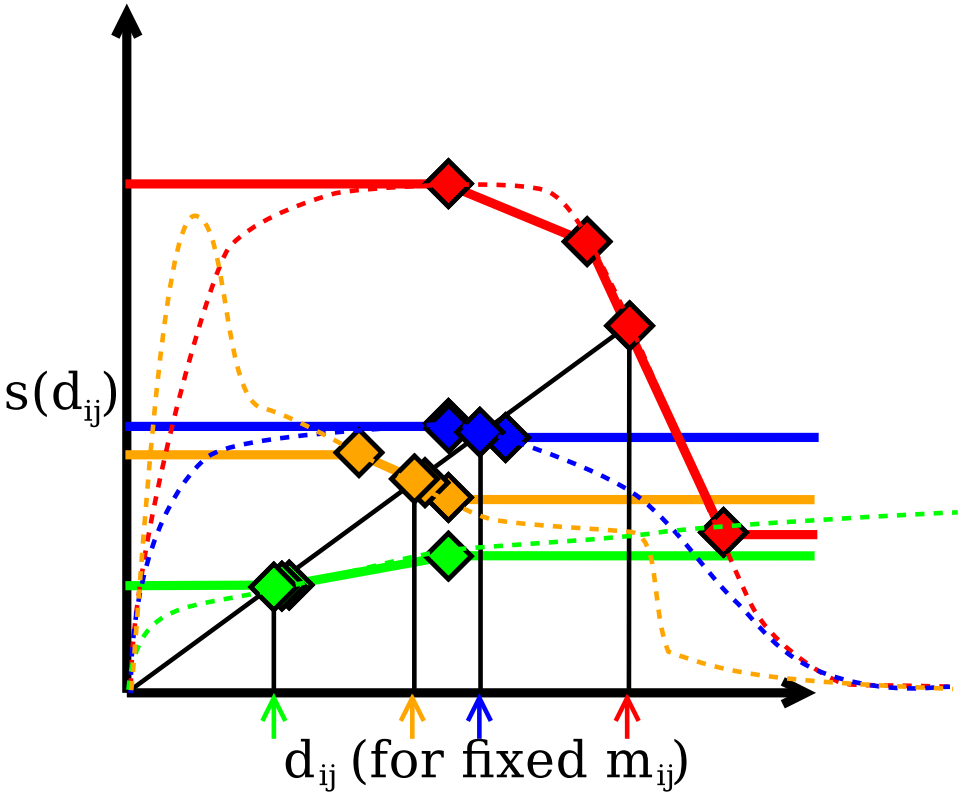
models of devices
(from 1 node)

$$w_i = m_i \times n_i$$

$$\sum_{j=1}^{p} d_{ij} = b \times n_i$$



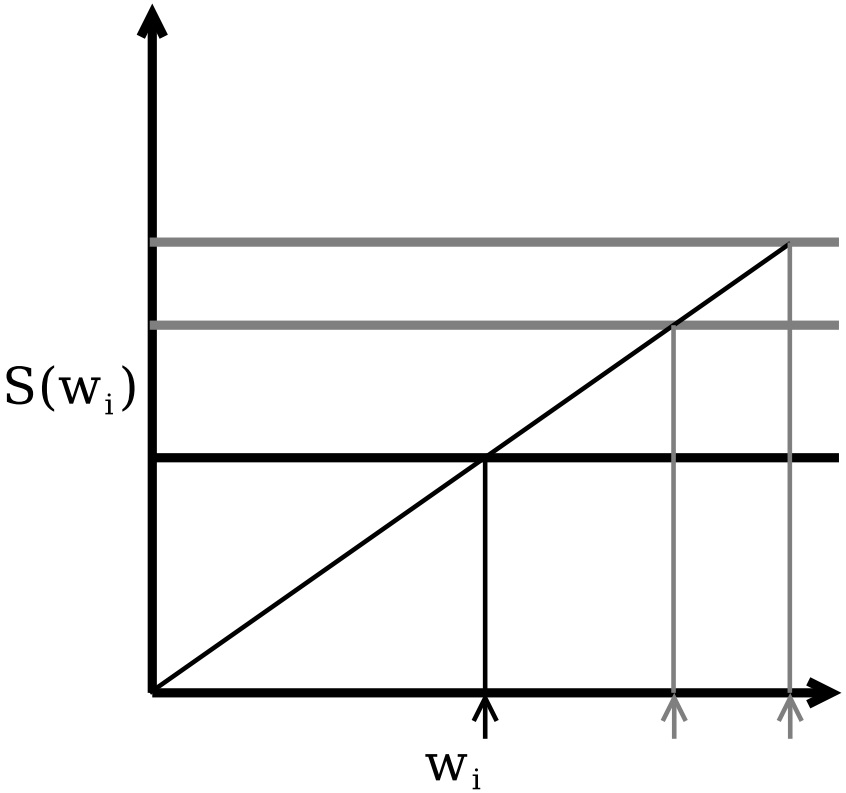
models of nodes



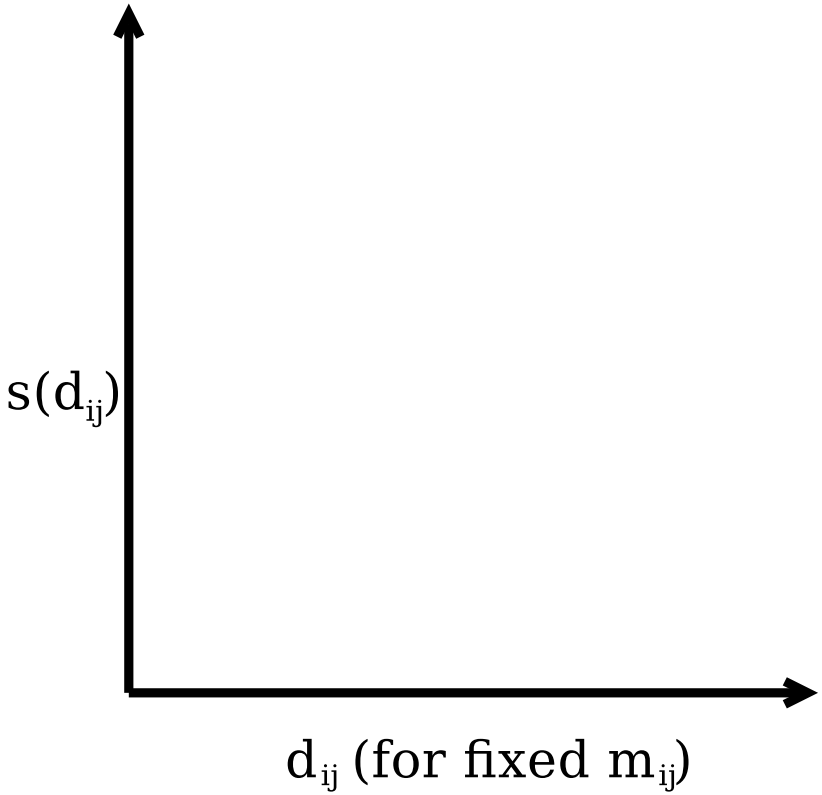
models of devices
(from 1 node)

$$w_i = m_i \times n_i$$

$$\sum_{j=1}^{p} d_{ij} = b \times n_i$$



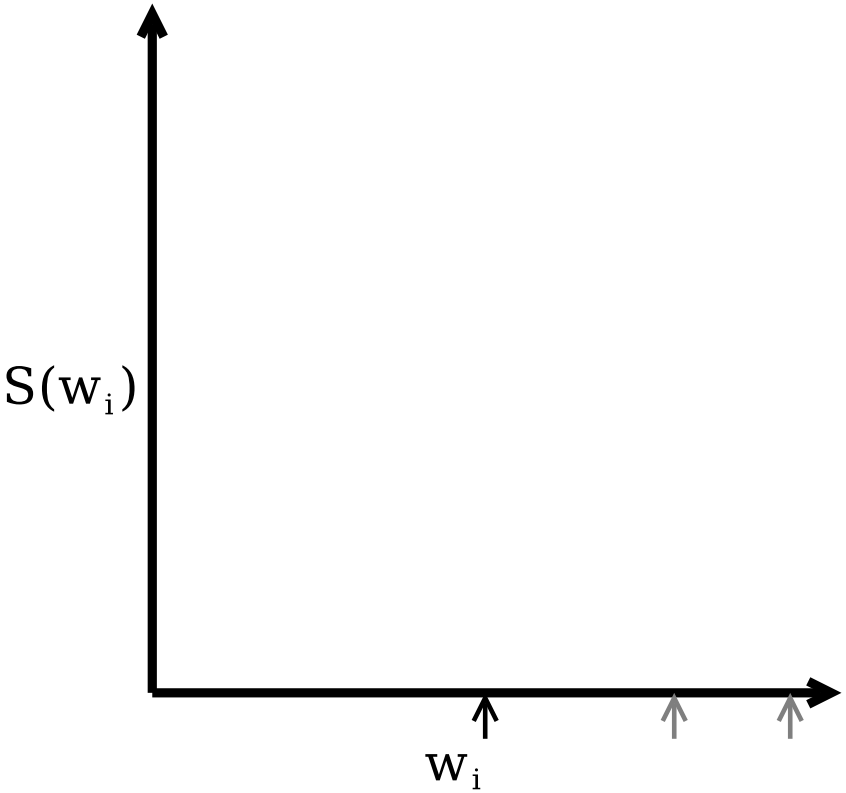
models of nodes



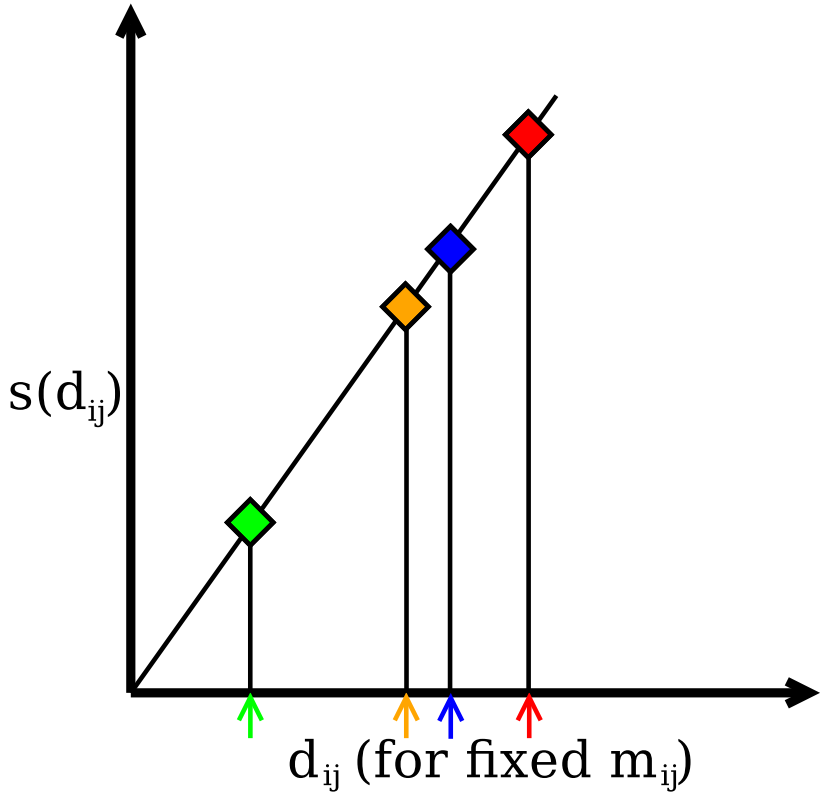
models of devices
(from 1 node)

$$w_i = m_i \times n_i$$

$$\sum_{j=1}^{p} d_{ij} = b \times n_i$$



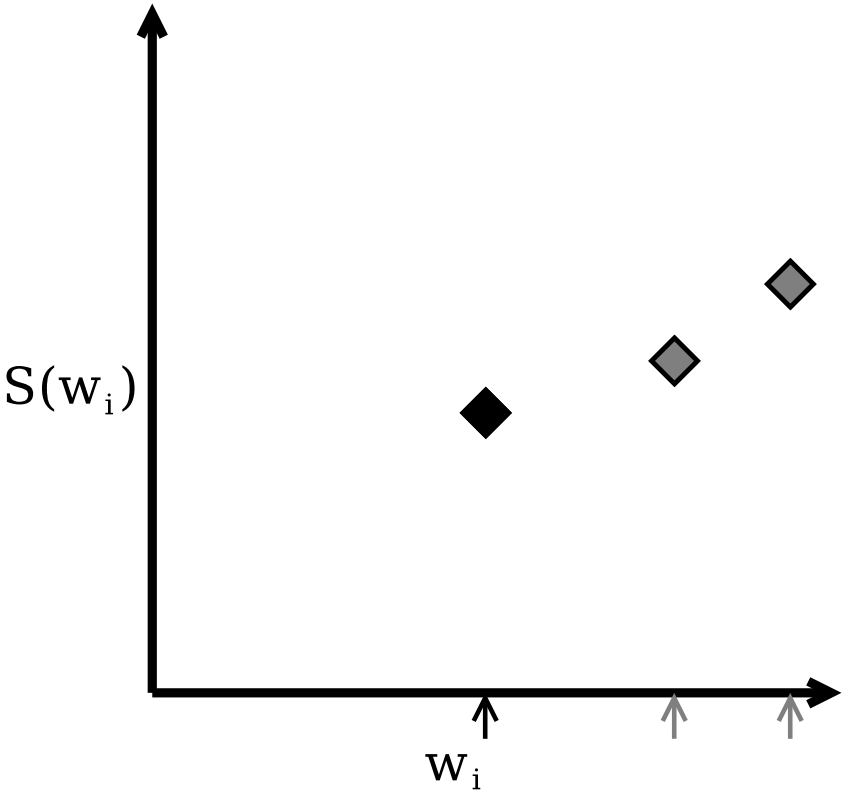
models of nodes



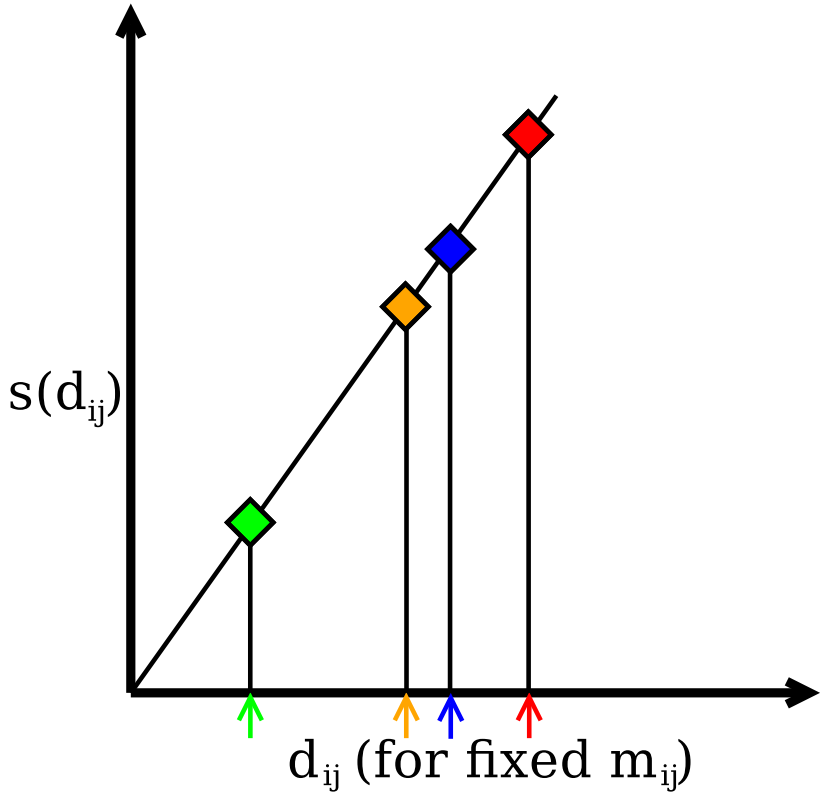
models of devices
(from 1 node)

$$w_i = m_i \times n_i$$

$$\sum_{j=1}^{p} d_{ij} = b \times n_i$$



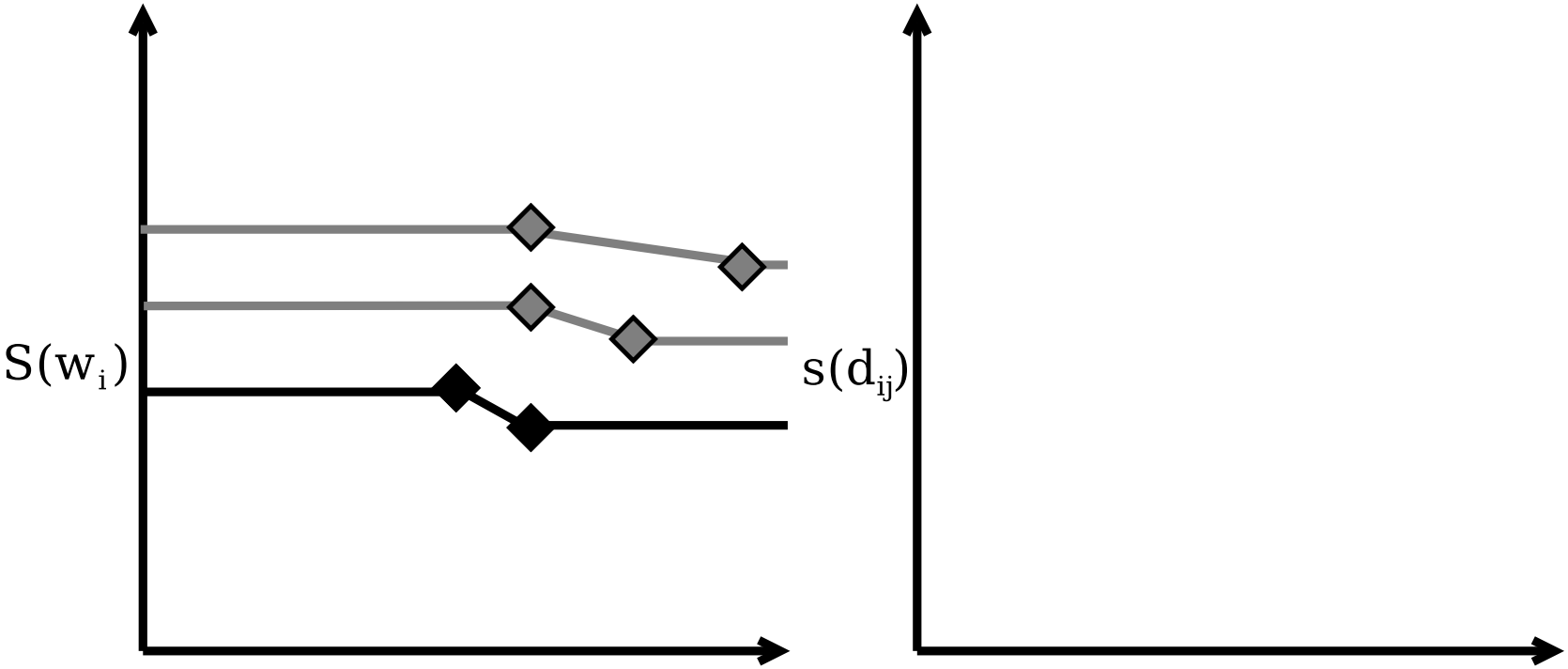
models of nodes



models of devices
(from 1 node)

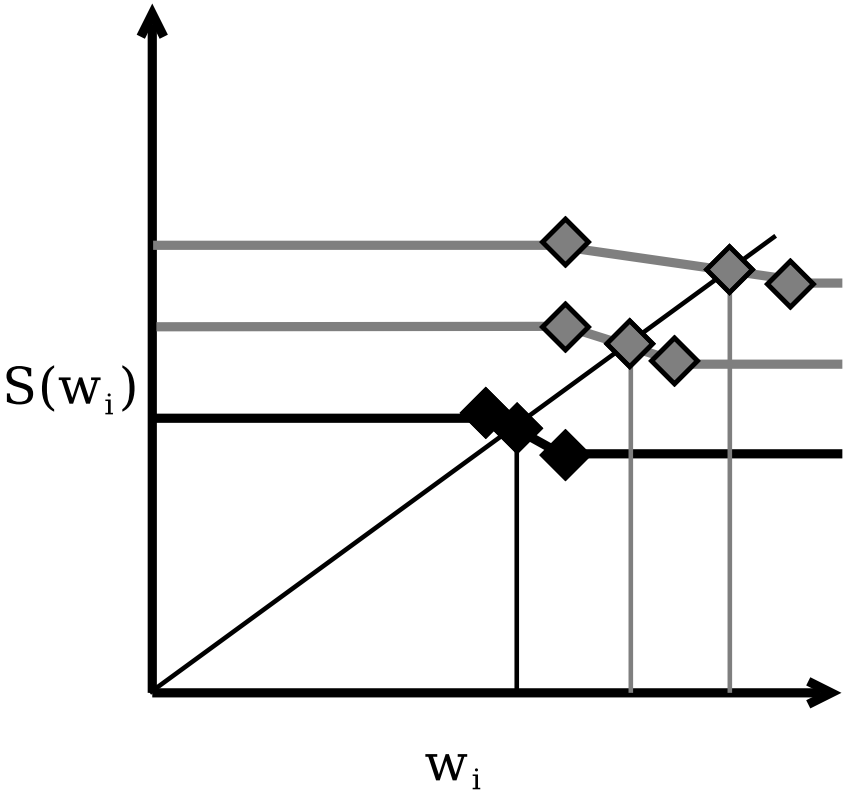
$$w_i = m_i \times n_i$$

$$\sum_{j=1}^{p} d_{ij} = b \times n_i$$

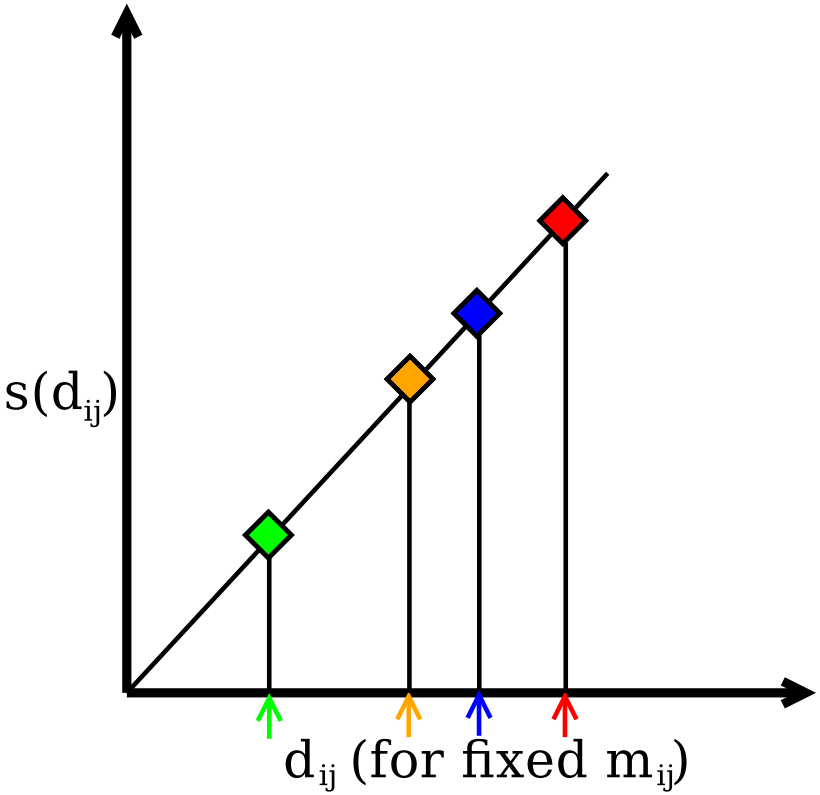


$$w_i = m_i \times n_i$$

$$\sum_{j=1}^{p} d_{ij} = b \times n_i$$



models of nodes

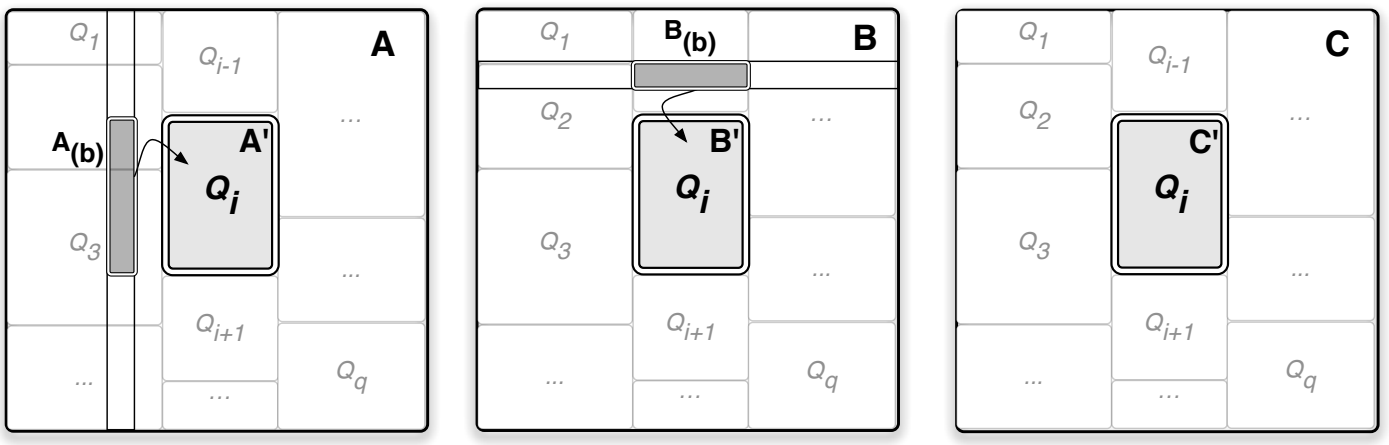


models of devices
(from 1 node)

$$w_i = m_i \times n_i$$

$$\sum_{j=1}^{p} d_{ij} = b \times n_i$$

Parallel Matrix Multiplication



(a)

(b)

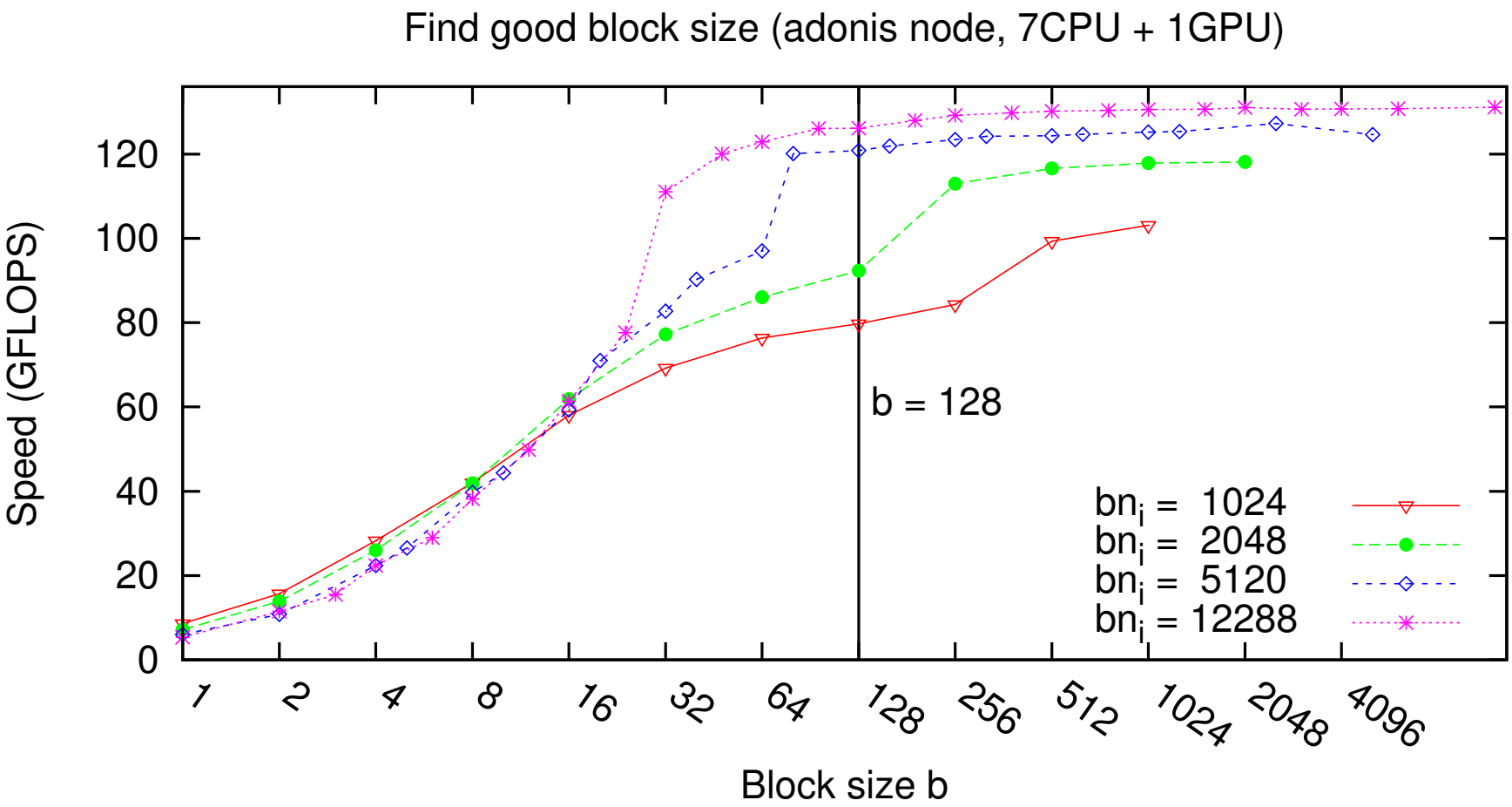
Experimental Setup

90 Nodes from Grid'5000 Grenoble site

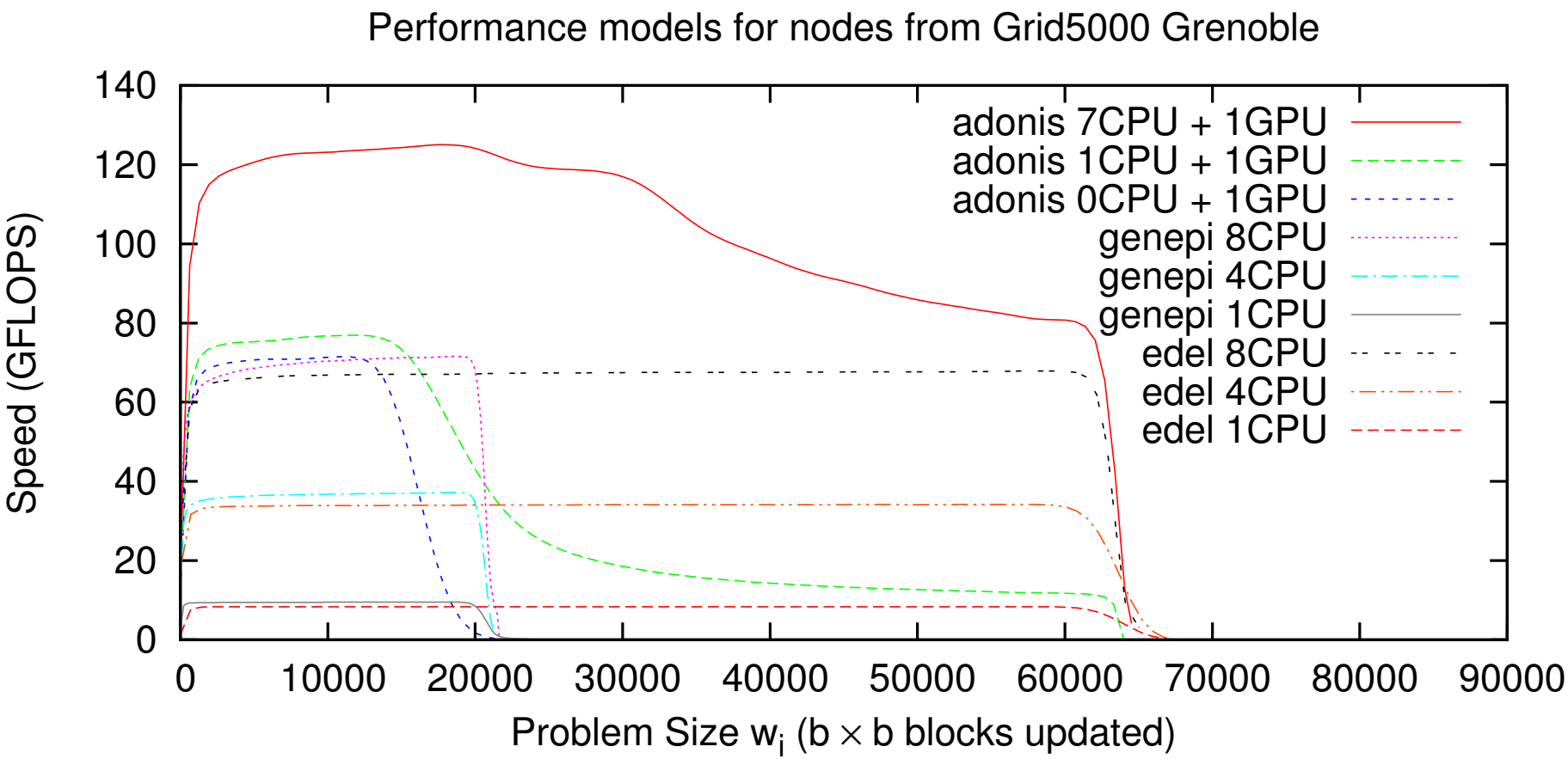
Cores:	0	1	2	3	4	5	6	7	8	Nodes	Cores	GPUs	Hardware
Adonis	2	1	1	1	1	1	2	3	0	12	48	12	2.27/2.4GHz Xeon, 24GB
Edel	0	6	4	4	4	8	8	8	8	50	250	0	2.27GHz Xeon, 24GB
Genepi	0	3	3	3	3	4	4	4	4	28	134	0	2.5GHz Xeon, 8GB
Total										90	432	12	

- All nodes connected with InfiniBand communication network.
- High performance BLAS libraries: Intel MKL for CPU, CUBLAS for GPU devices.
- Open MPI for inter node communication.
- OpenMP for inter-device parallelism.

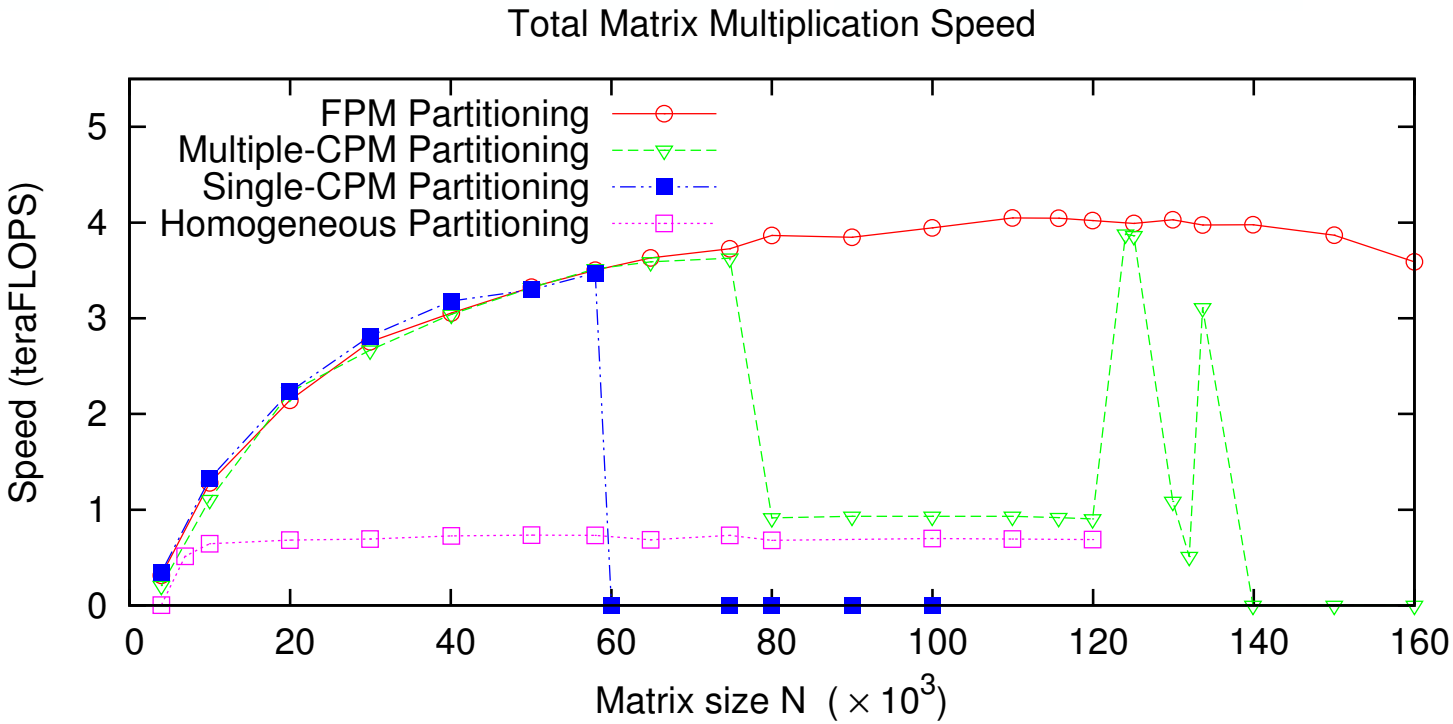
Experimental Results



Experimental Results



Experimental Results



- Functional performance model (FPM): the proposed algorithm
- Multiple constant performance models (CPM): Redistribute based on previous benchmark.
- Single-CPM: One benchmark is preformed.
- Homogeneous distribution: Partitioned evenly between nodes, then evenly between devices within each node.