Design and implementation of parallel algorithms for highly heterogeneous HPC platforms

Dave Clarke, Alexey Lastovetsky, Ravi Reddy, Vladimir Rychkov

School of Computer Science and Informatics

University College Dublin

Alexey.Lastovetsky@ucd.ie

http://hcl.ucd.ie

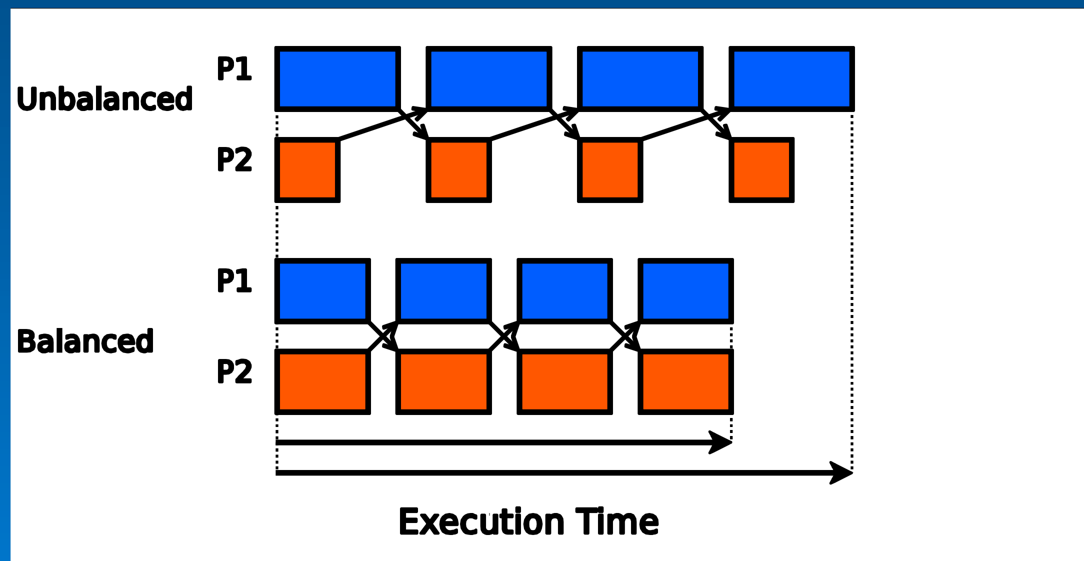
CCGSC 2010, Ashville, September 2010

Motivation

- Traditional mainstream parallel computing systems
  - Used to be homogeneous
    - » At least, at the application level
  - Parallel algorithms
    - » Try to distribute computations evenly
- New trend in mainstream parallel computing systems
  - Heterogeneous processing devices
    - » Heterogeneous cores, accelerators (GPUs)
    - » Heterogeneous clusters
    - » Clusters of clusters

Motivation (ctd)

- New heterogeneous parallel algorithms needed
  - To distribute computations between heterogeneous processing devices unevenly
    - » Ideally, in proportion to their speed

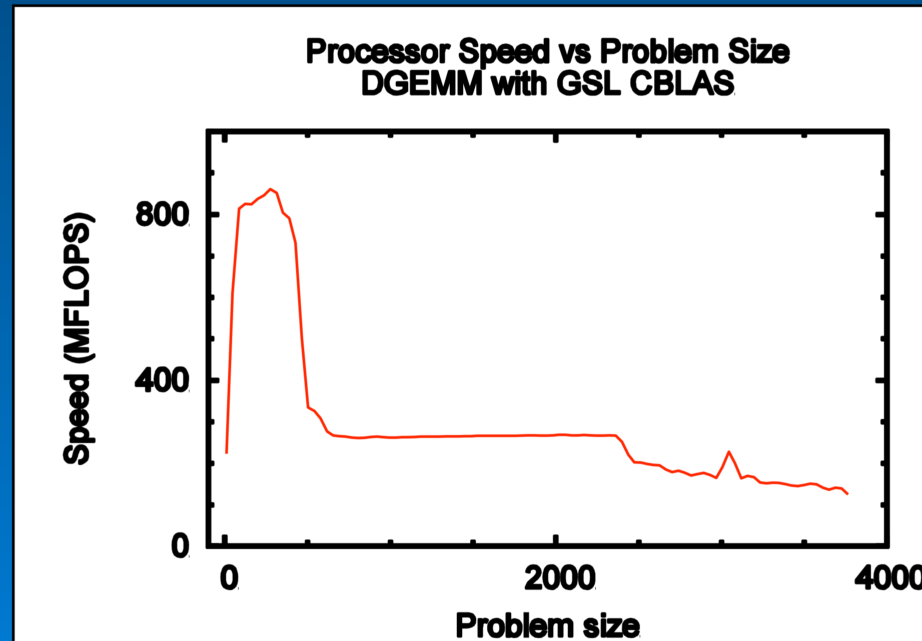


Motivation (ctd)

- Since mid 90s, fundamental heterogeneous parallel algorithms for scientific computing have been designed
  - Introduced a new type of parameters representing the performance of processors
  - Significantly outperformed their homogeneous counterparts
    - » Heterogeneous clusters of workstations (main target platform)
    - » Given the performance parameters are accurate
- Can we use these algorithms for the new platforms?
  - Not quite
- Why?
  - The performance parameters are constants
    - » Assuming the (relative) speed of the processors does not depend on the sizes of computational tasks

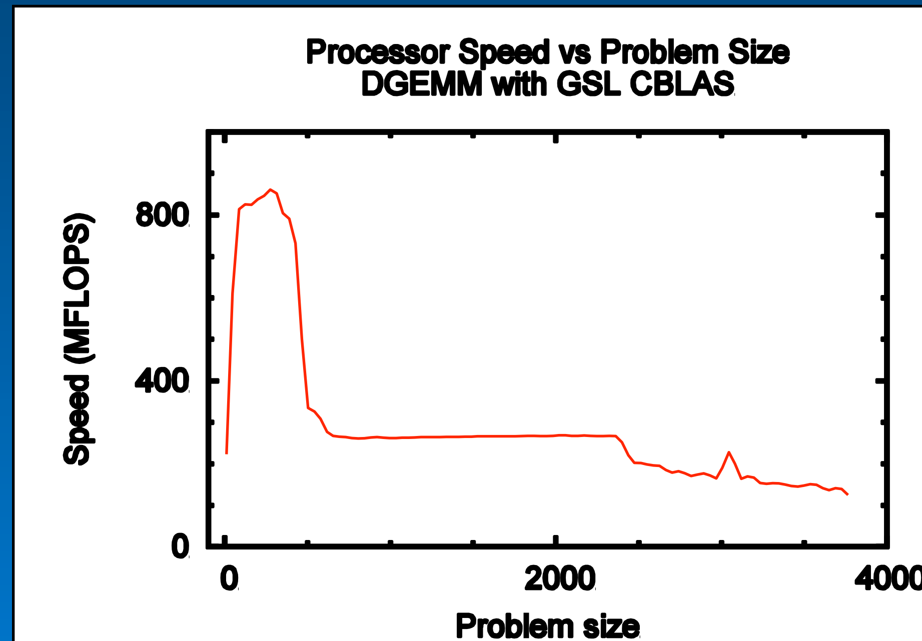
Motivation (ctd)

- Constant performance models (CPMs) are sufficiently accurate if
  - All processors are general-purpose of traditional architecture, and
  - Same code used for local computations on all processors, and
  - Computational task assigned to each processor is small enough to fit into main memory and big enough not to fully fit into cache.



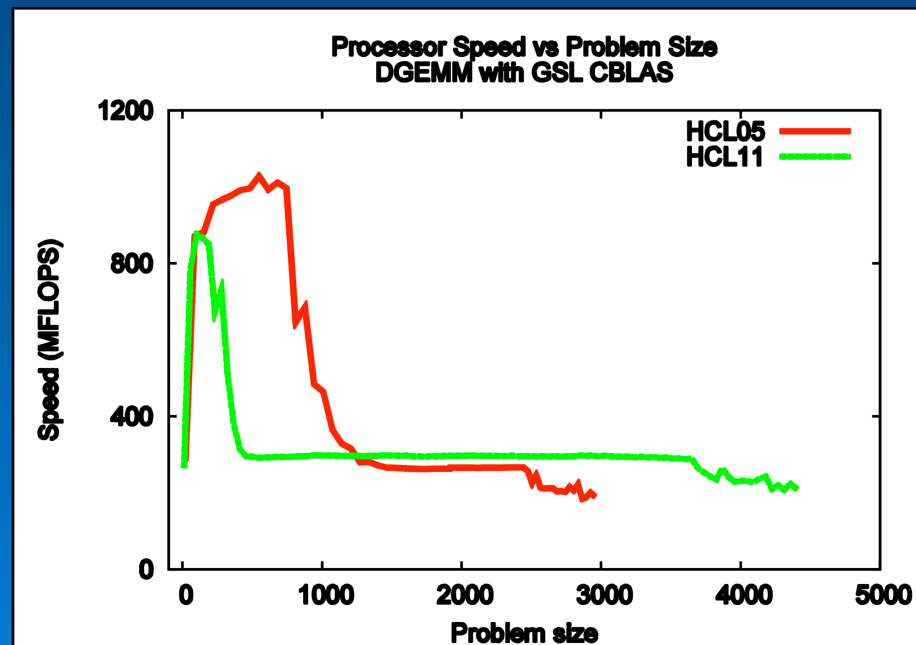
Motivation (ctd)

- The assumption of constant speed may not be accurate if
  - Some tasks either fitting into cache or not fitting into main memory, or
  - Some processing units are not traditional (GPUs, specialised cores), or
  - Different processors use different codes for local computations



Motivation (ctd)

- Applicability of CPMs and CPM-based algorithms
  - The more different P_1 and P_2, the smaller will be the range of sizes R_{12} where their relative speed can be accurately approximated by a constant
  - If the number of significantly different PUs is large enough, then region $\cap R_{ij}$ of applicability of CPM-based algorithms can be very small



Functional performance model (FPM)

- CPM-based algorithms
  - Very restricted for highly heterogeneous platforms
  - Never cover the full range of problem sizes
- Solution:
  - Use FPM to define the performance of processing units
    - » The absolute speed of processor is represented by a function of problem size rather by a constant
    - » Natural, simple and general (applicable to any processing unit)
      - No architectural parameters
  - Use FPM to design heterogeneous parallel algorithms

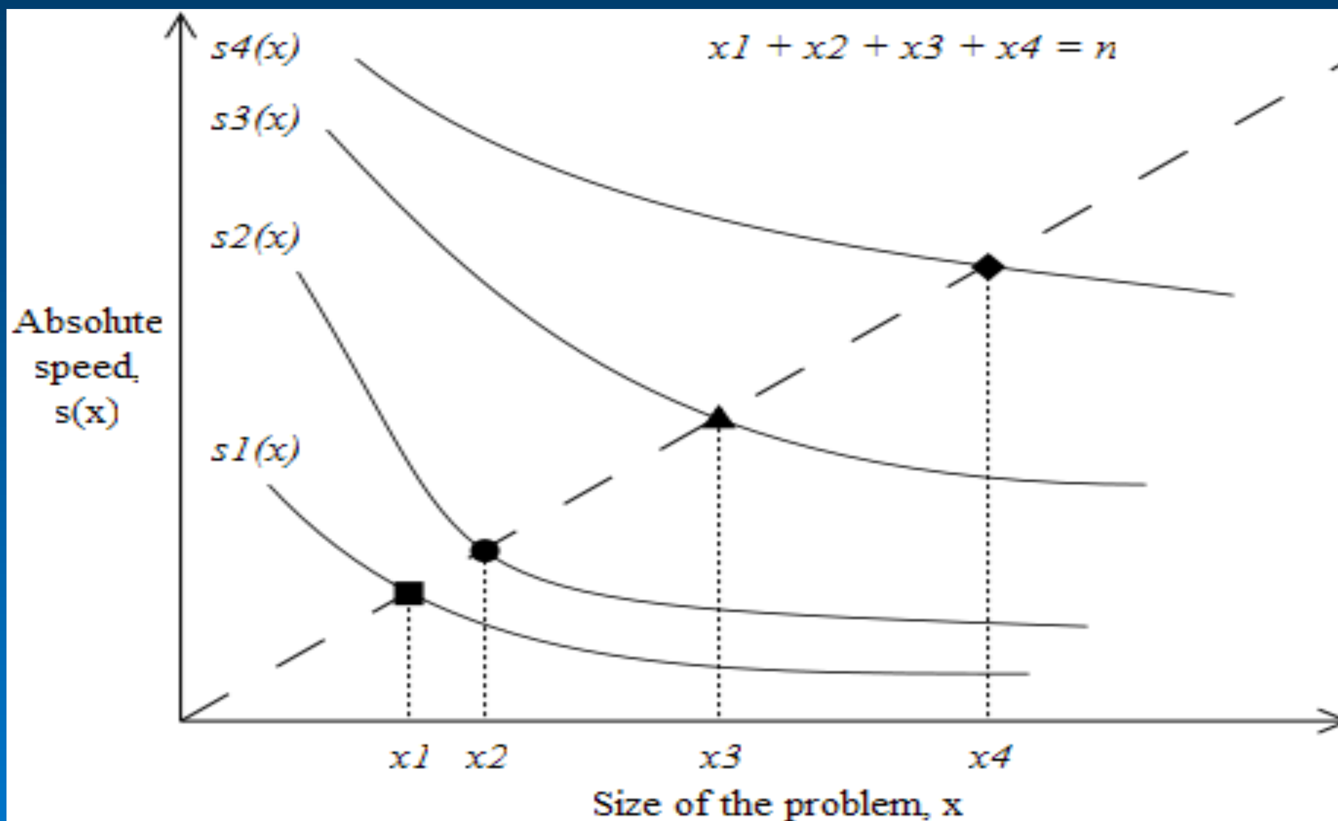
FPM-based algorithms

- We have studied the following problem
  - Given
    - » A set of n elements (say, representing equal computation units)
    - » A well-ordered set of p processors whose speeds are continuous functions of the size of problem, $s_i=f_i(x)$,
  - Partition the set into p sub-sets such that
    - » The partitioning minimizes $\max_i \frac{n_i}{s_i(n_i)}$,

where n_i is the number of elements allocated to processor P_i

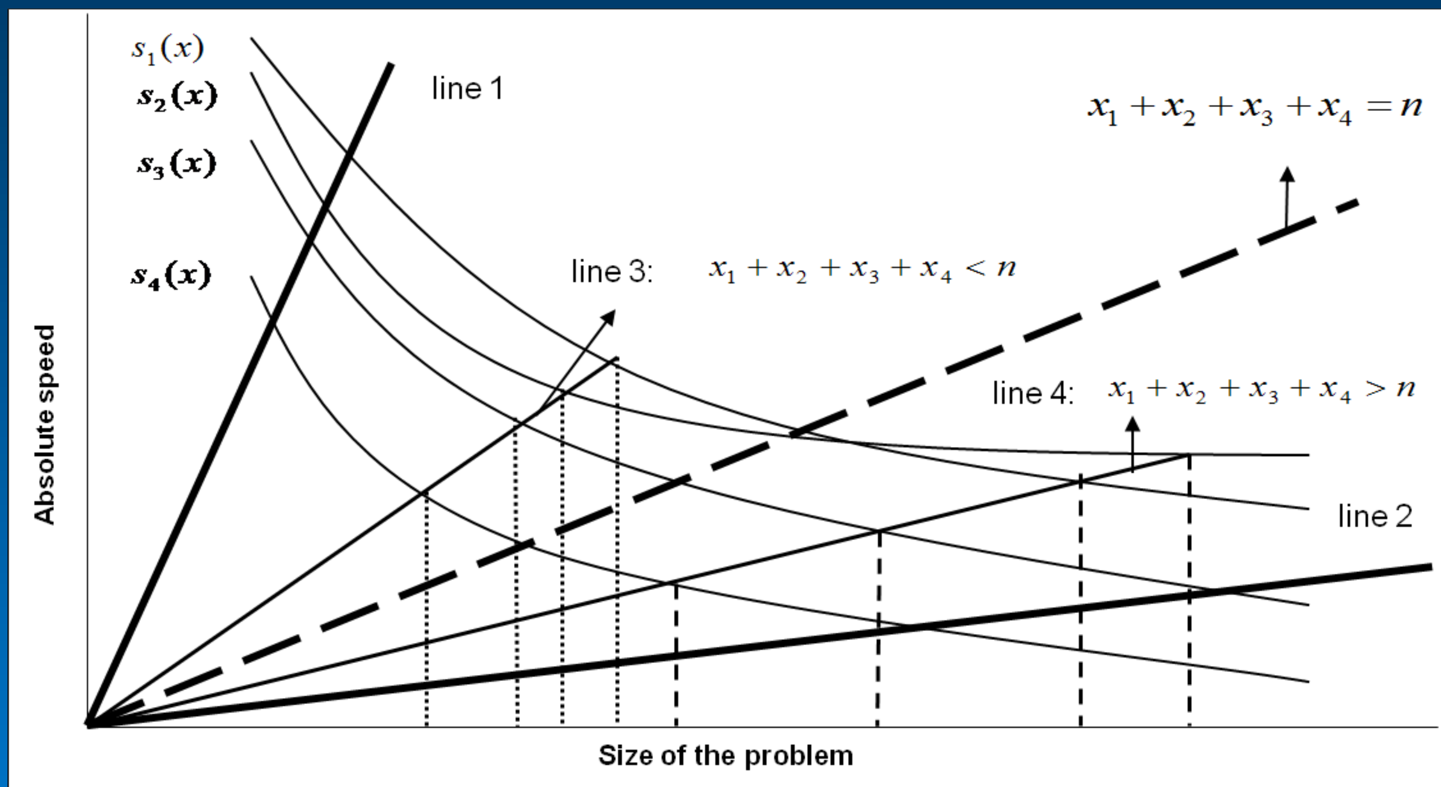
FPM-based algorithms (ctd)

- Partitioning algorithms are based on the observation:



FPM-based algorithms (ctd)

- A typical algorithm works as follows:



$\max_i \frac{n_i}{s_i(n_i)}$

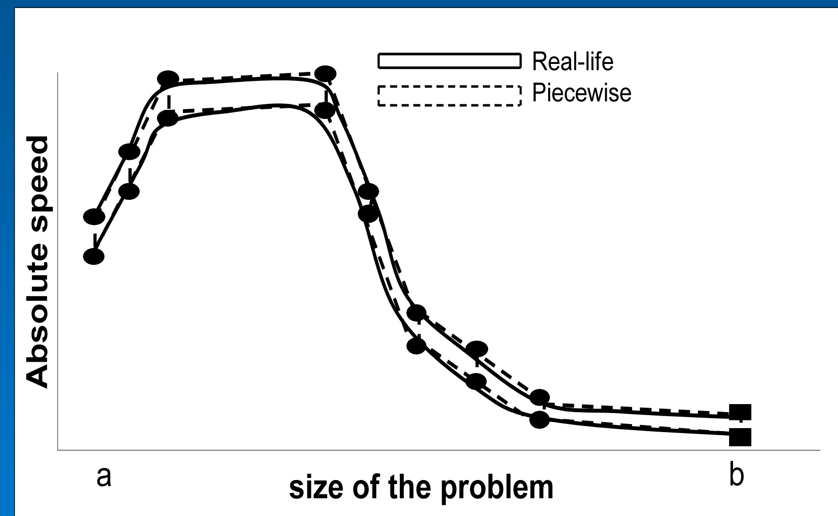
FPM-based algorithms (ctd)

- A number of algorithms have been designed and validated using the FPM-based partitioning
  - Linear algebra
    - » 1D LU factorisation
    - » 2D matrix multiplication
  - Database applications (TPC-H Benchmark)
  - Different platforms
    - » Heterogeneous computational clusters
    - » Multicore and accelerator based desktop systems

$\max_i \frac{n_i}{s_i(n_i)}$

FPM-based algorithms (ctd)

- Implementation issues FPM-based algorithms
  - FPMs of the processing units are input parameters
    - » The efficiency of applications depends on the accuracy and "quality" of the FPMs
    - » In general, FPMs are multi-dimensional surfaces (not just curves)
  - FPM construction issues
    - » Accuracy
    - » Quality
    - » Efficiency



FPM-based algorithms (ctd)

The cost of constructions of FPMs can be very high

=> The FPM-based algorithms using FPMs as input parameters

- ☹ cannot be used in self-adaptable applications
- ☺ still can be used in applications repeatedly running in a stable environment
  - FPMs are constructed once and used multiple times

FPM-based algorithms for self-adaptable applications

- Solution
  - Do not use full pre-defined FPMs for partitioning
    - » Full FPMs are no longer input parameters of the partitioning algorithm
  - Use partial approximations of the FPMs instead, which are
    - » Not predefined
    - » Constructed for each particular problem size during the execution of the partitioning algorithm
    - » Accurate enough for the required accuracy of partitioning
    - » Covering the range of problem sizes just sufficient to solve the partitioning problem of the given size

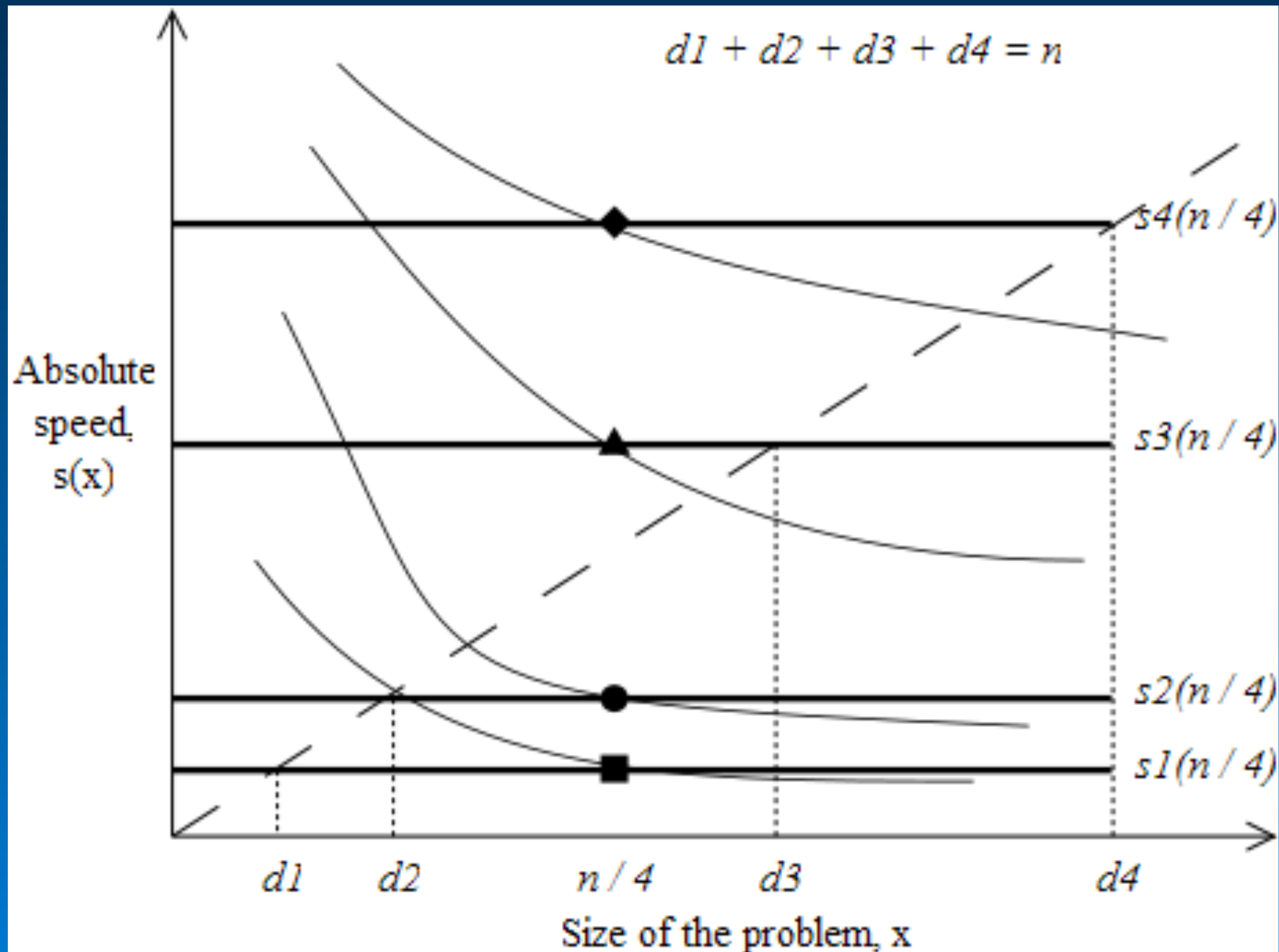
Adaptive FPM-based partitioning algorithm

- We study the following problem
  - Given
    - » A set of n elements (say, representing equal computation units)
    - » A well-ordered set of p processors whose speeds of processing x elements, $s_i=s_i(x)$, can be obtained by measuring the execution time, $t_i(x)$, of a computational kernel, $s_i(x)=x/t_i(x)$
  - Partition the set into p sub-sets such that

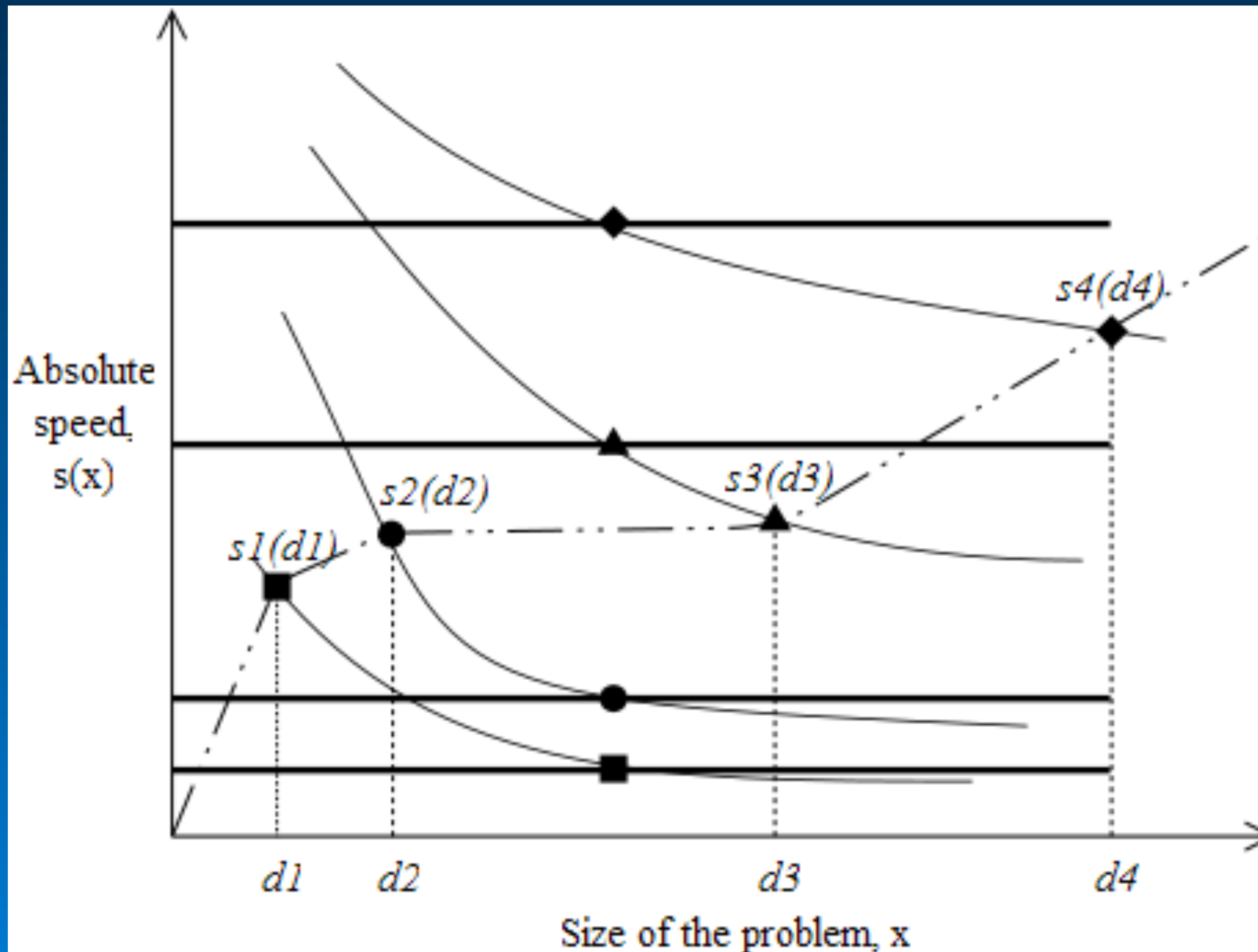
$$\max_{1\le i,j\le p}\left(\left|\frac{t_i(n_i)-t_j(n_j)}{t_i(n_i)}\right|\right)\le\varepsilon$$

where n_i is the number of elements allocated to processor P_i

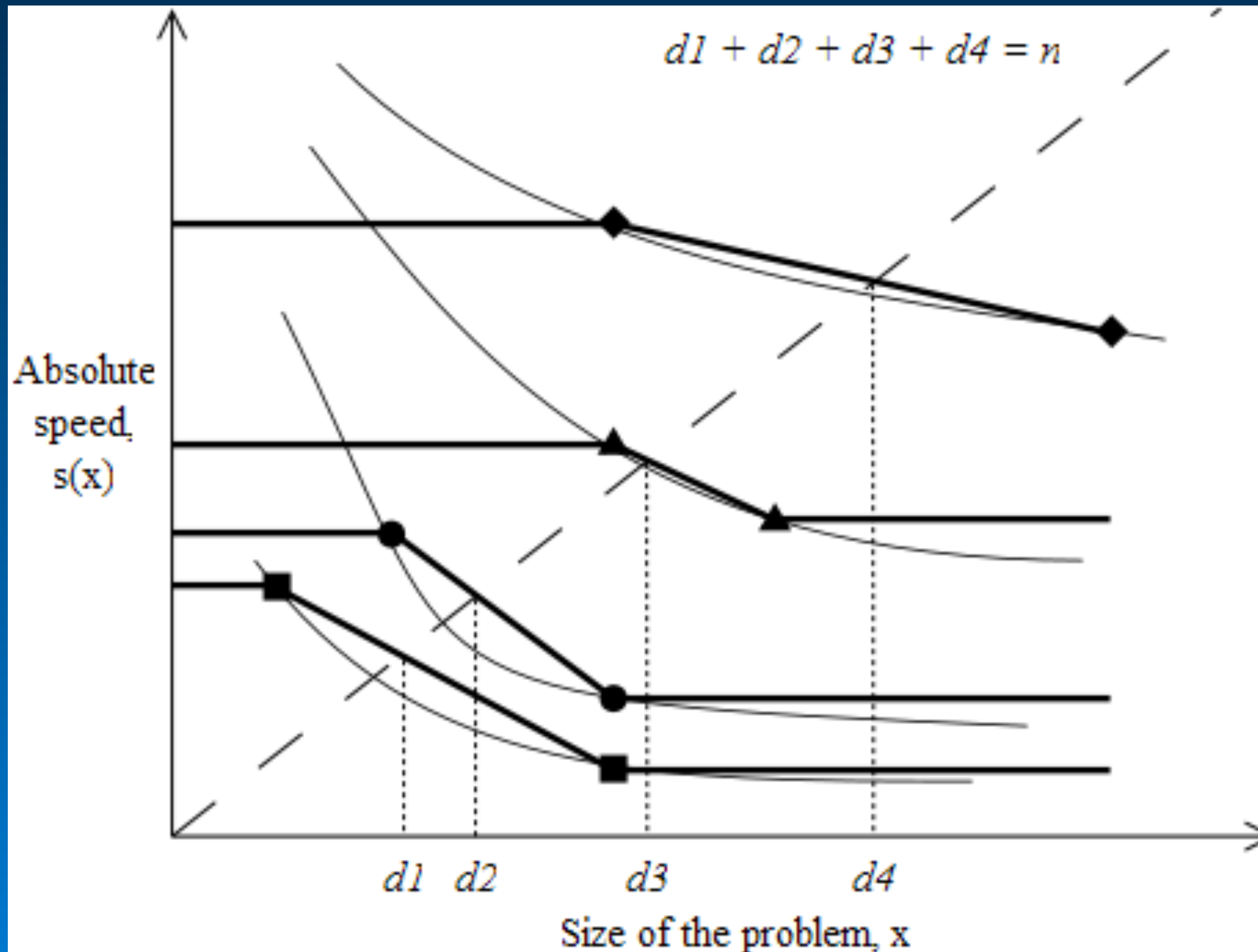
Adaptive partitioning algorithm (0)



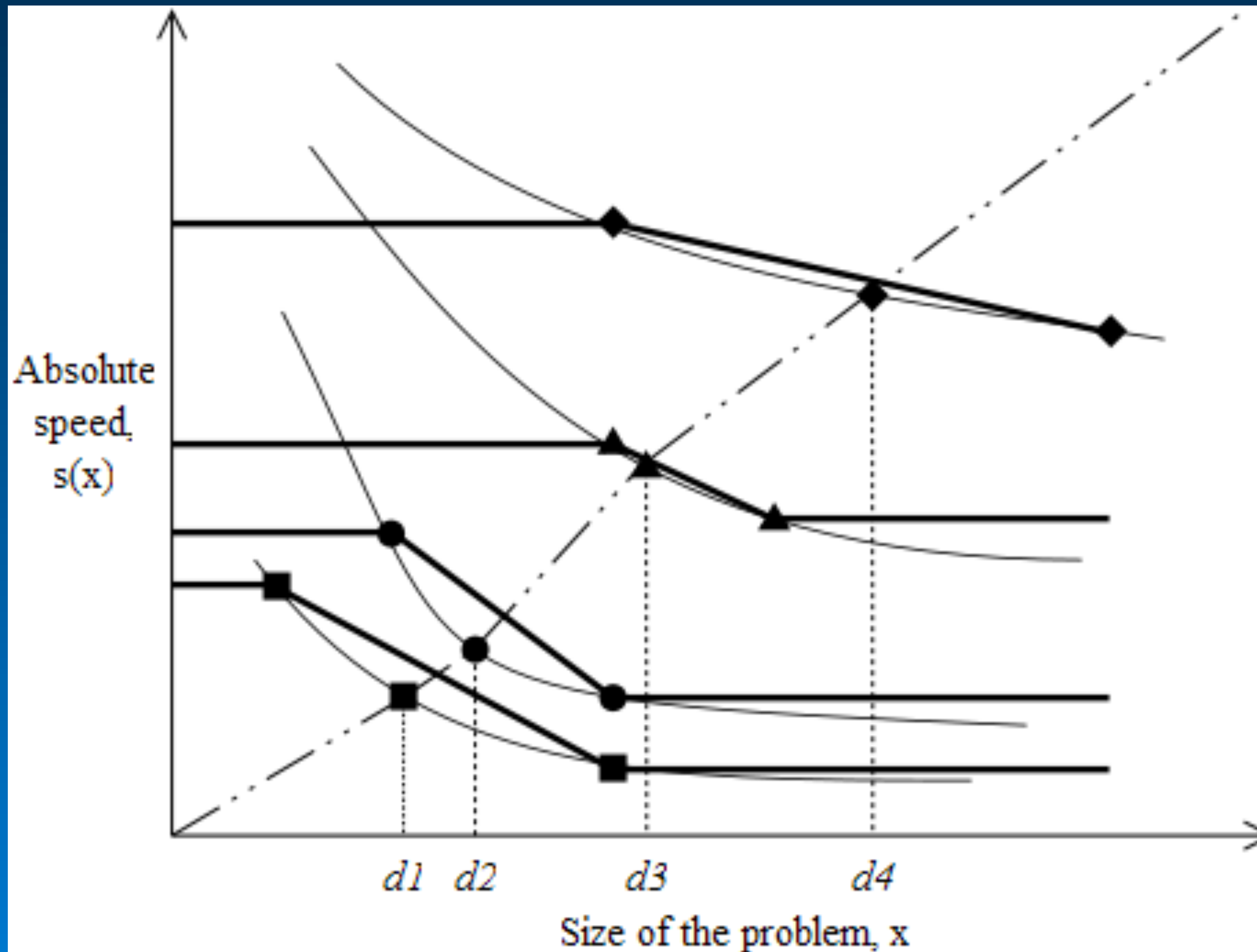
Adaptive partitioning algorithm (1)



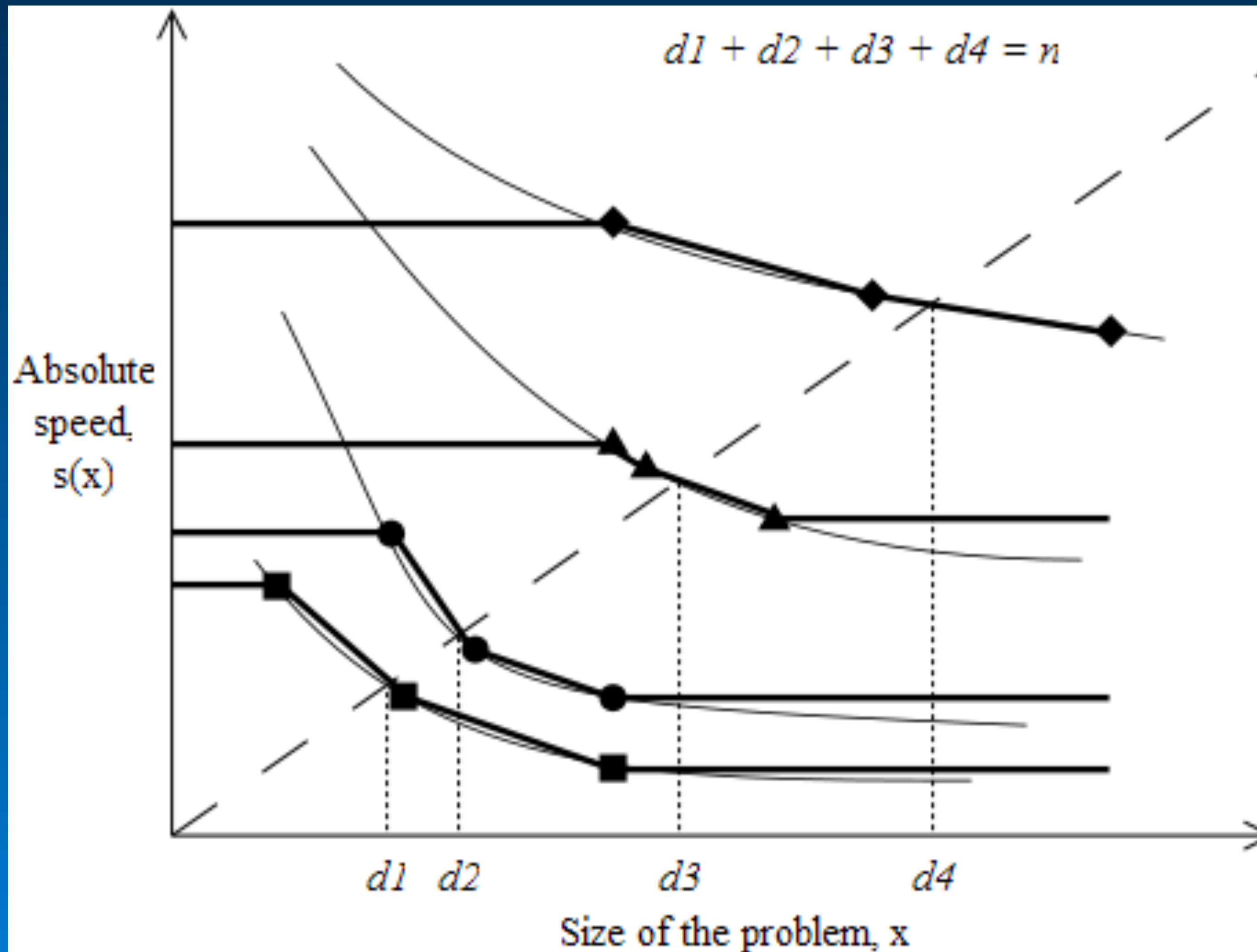
Adaptive partitioning algorithm (2)



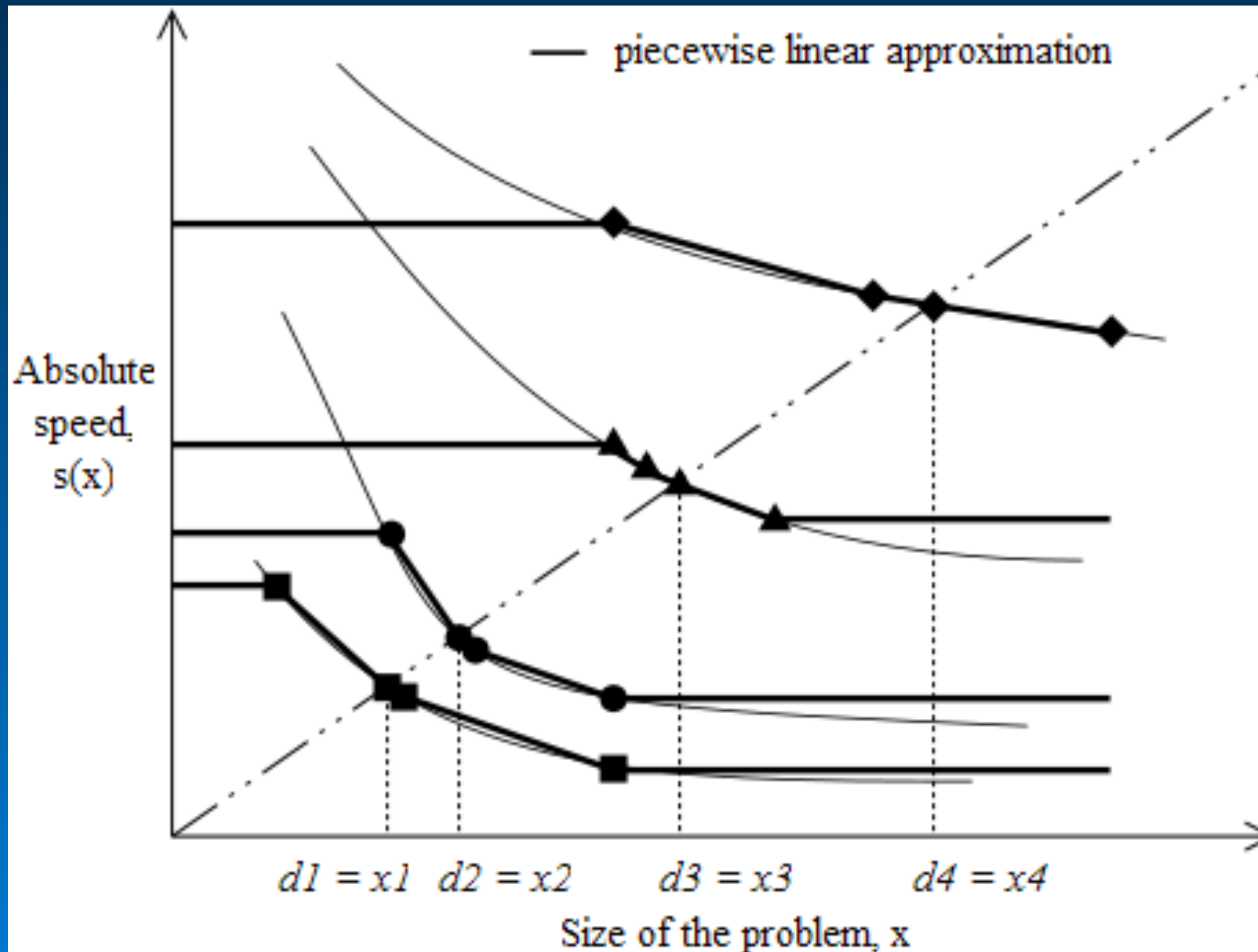
Adaptive partitioning algorithm (3)



Adaptive partitioning algorithm (4)



Adaptive partitioning algorithm (5)

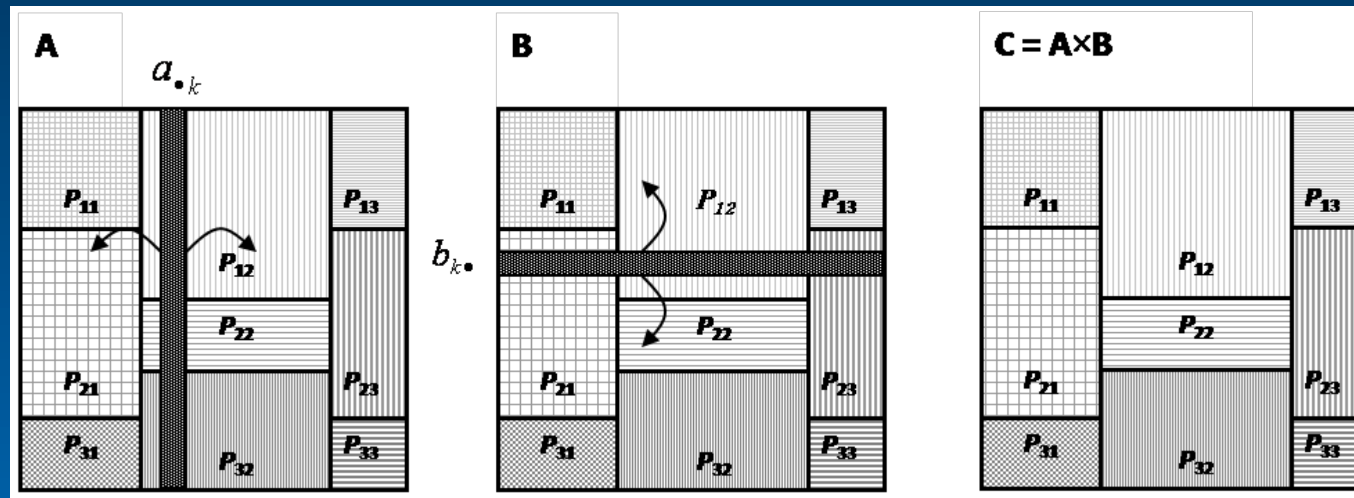


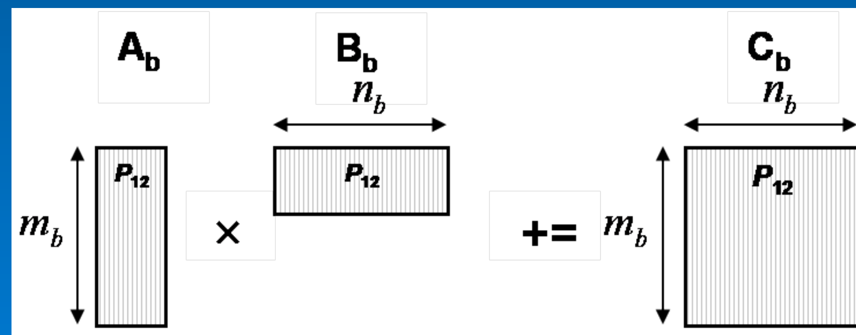
Adaptive FPM-based partitioning algorithm

- The adaptive algorithm
  - Distributed
    - » Involves all participating processors
- Implementation issues
  - Mainly, FPM related
  - Accuracy
    - » Higher accuracy of FPM ➔ more accurate partitioning
  - Quality
    - » Smoother approximations ➔ faster convergence
  - Efficiency
    - » Minimization of estimation cost
    - » Minimization of the overall execution time

Experiments: matrix multiplication

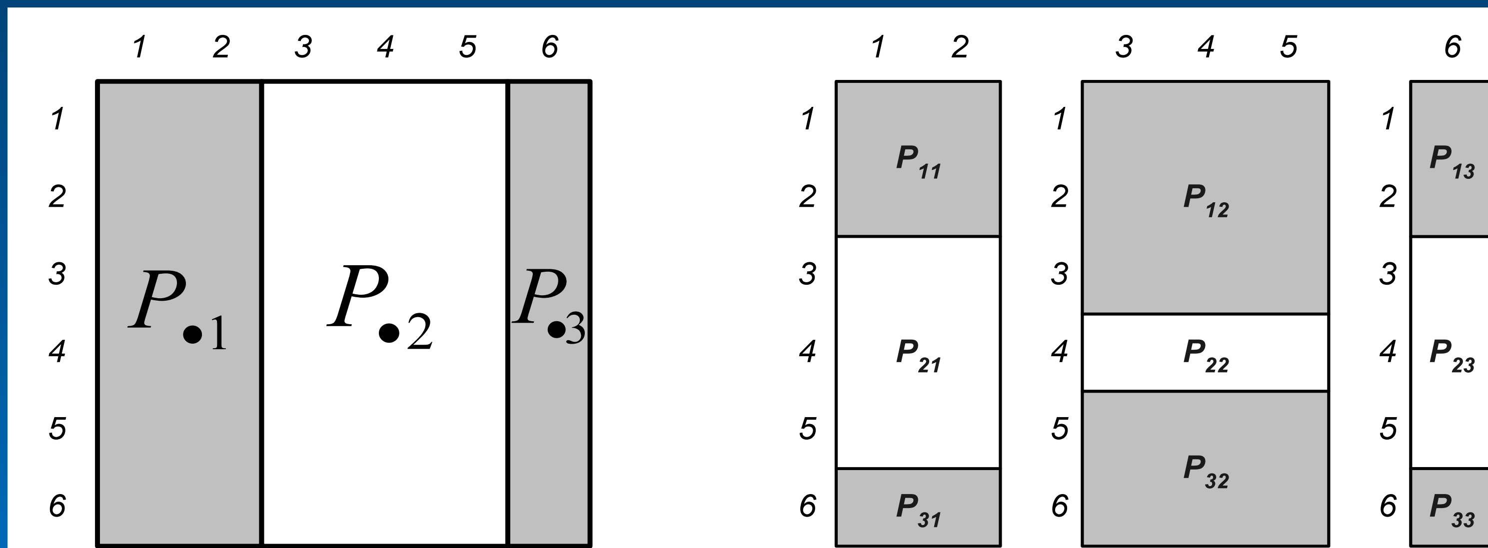
- Parallel matrix multiplication on a heterogeneous cluster



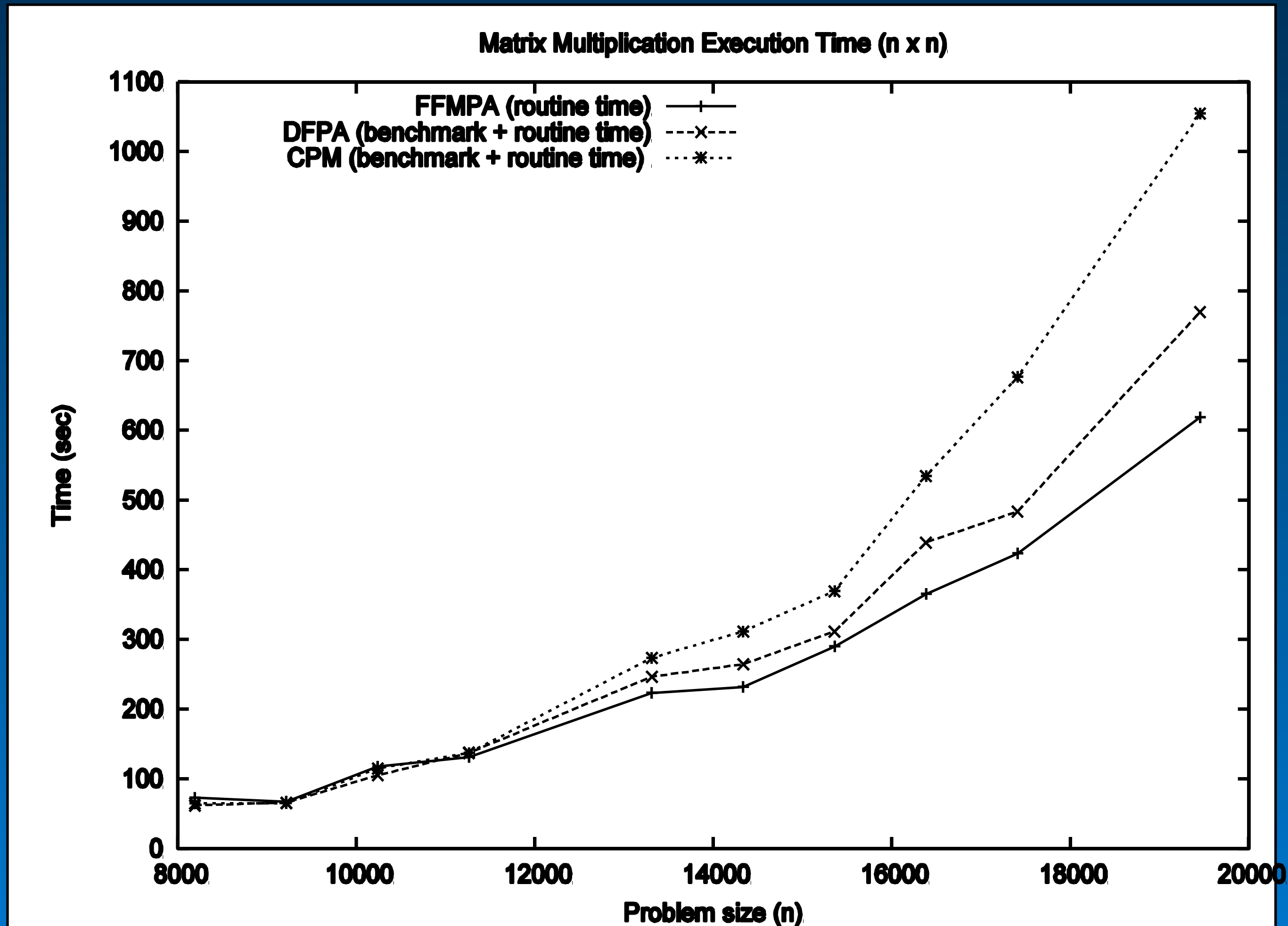


Experiments: matrix multiplication (ctd)

- Partitioning matrices



Experiments: matrix multiplication (ctd)



Experiments: matrix multiplication (ctd)

Matrix size ($n \times n$)	Total execution time (sec)	DFPA time (sec)	DFPA iterations	Matrix multiplication (sec)	DFPA cost (%)
8192	**61.91**	**0.17**	**16**	**61.74**	**0.28**
9216	**65.91**	**0.14**	**11**	**65.76**	**0.21**
10240	**105.22**	**0.19**	**13**	**105.02**	**0.18**
11264	**137.34**	**0.22**	**15**	**137.11**	**0.16**
13312	**246.49**	**5.84**	**44**	**240.65**	**2.36**
14336	**264.45**	**16.25**	**62**	**248.20**	**6.14**
15360	**311.28**	**24.06**	**69**	**287.22**	**7.73**
16384	**448.27**	**28.44**	**71**	**419.83**	**6.34**
17408	**483.23**	**52.51**	**69**	**430.71**	**10.86**

Experiments: Load balancing of iterative routines

- n computational units distributed across p processors.
- Processor P_i has d_i units such that $n = \sum_{i=1}^{p} d_i$
- **Initially** $d_i^0 = n / p$
- **At each iteration**
  - Execution times measured and gathered to root
  - **if** relative difference between times $\leq \varepsilon$
    then no balancing needed
    else new distribution is calculated as:

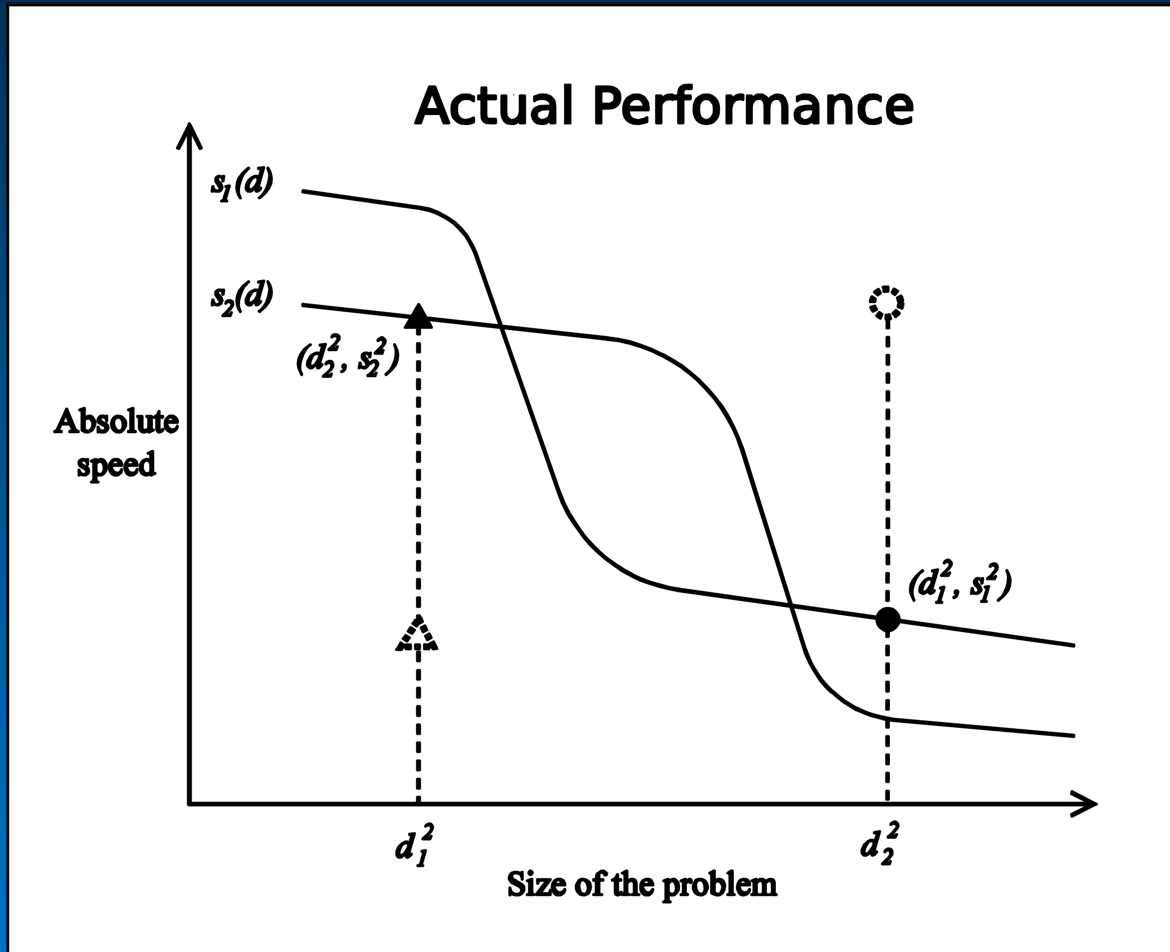
$$d_i^{k+1} = n \times \frac{s_i^k}{\sum_{j=1}^{p} s_j^k}$$ where speed $$s_i^k = \frac{d_i^k}{t_i(d_i^k)}$$

  - New distributions d_i^{k+1} broadcast to all processors and where necessary data is redistributed accordingly

Experiments: Load balancing of iterative routines (ctd)

- Speed of each processor is considered as a constant positive number at each iteration.
- Within the range of problem sizes for which this is true, traditional algorithms can successfully load balance.
- Can fail for problem sizes for which the speed is not constant.

Experiments: Load balancing of iterative routines (ctd)

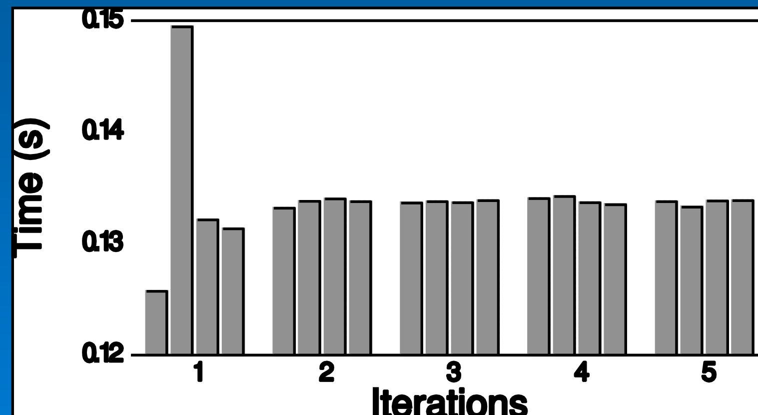


Experiments: Load balancing of iterative routines (ctd)

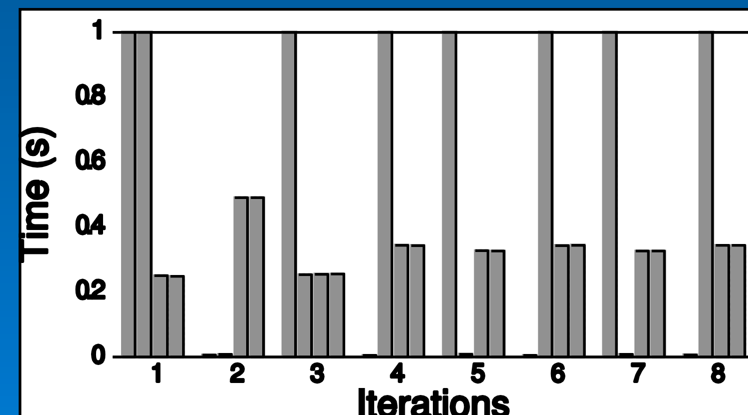
- **Iterative Routine**
  Jacobi method for solving a system of linear equations
- **Experimental Setup**

	P_1	P_2	P_3	P_4
Processor	3.6 Xeon	3.0 Xeon	3.4 Xeon	3.4 Xeon
Ram (MB)	256	256	512	1024

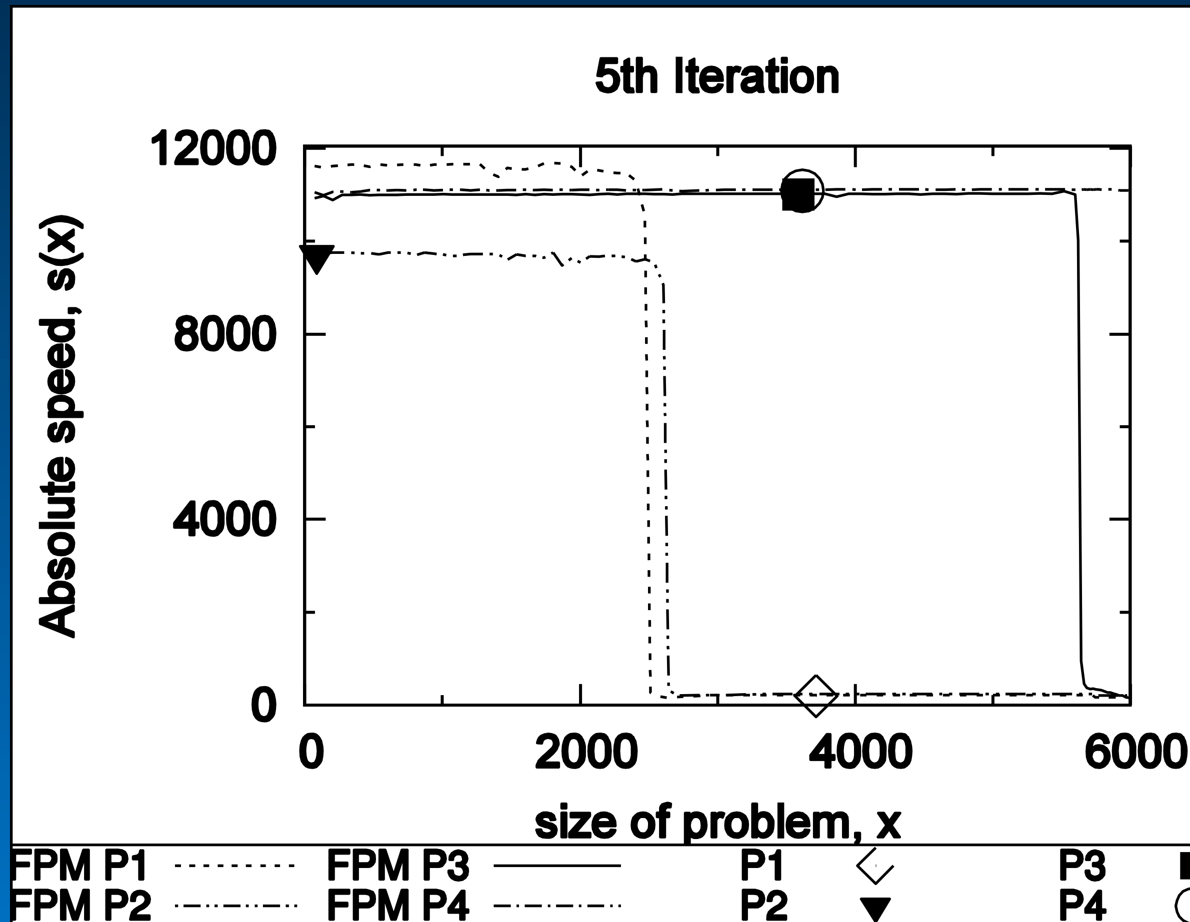
***n* = 8000**



***n* = 11000**



Experiments: Load balancing of iterative routines (ctd)



Experiments: Load balancing of iterative routines (ctd)

- Our algorithm is based on models for which speed is a function of problem size.
- Load balancing achieved when:

$$t_1 \approx t_2 \approx \ldots \approx t_p$$

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

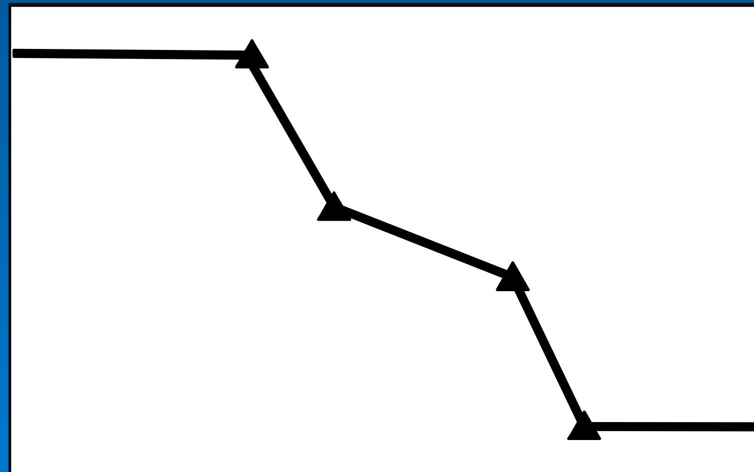
$$d_1 + d_2 + \ldots + d_p = n$$

Experiments: Load balancing of iterative routines (ctd)

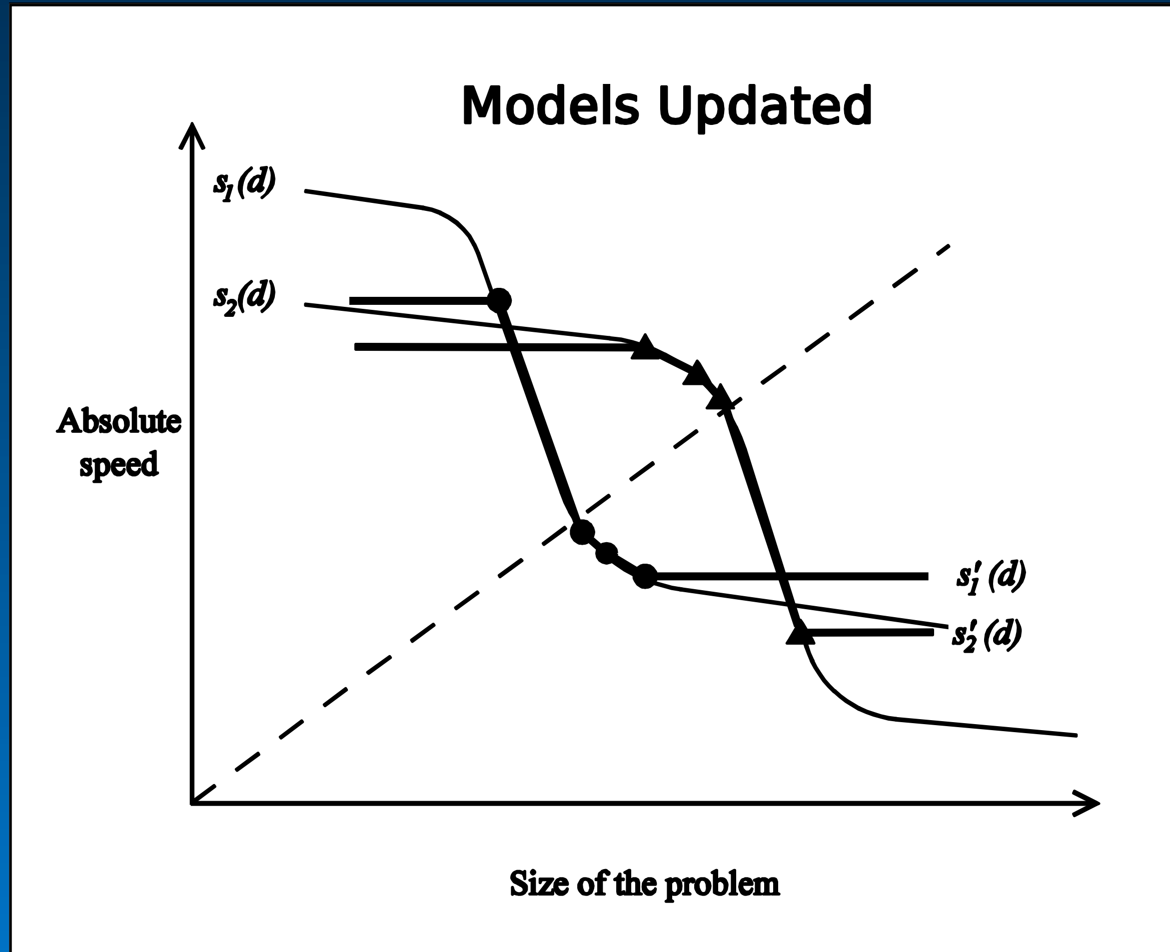
- **First iteration**
  Point (n / p, s_i^0) with speed $s_i^0 = \dfrac{n/p}{t_i(n/p)}$

  First function approximation s_i'(d) = s_i^0
- **Subsequent iterations**
  Point (d_i^k, s_i^k) with speed $s_i^k = \dfrac{d_i^k}{t_i(d_i^k)}$

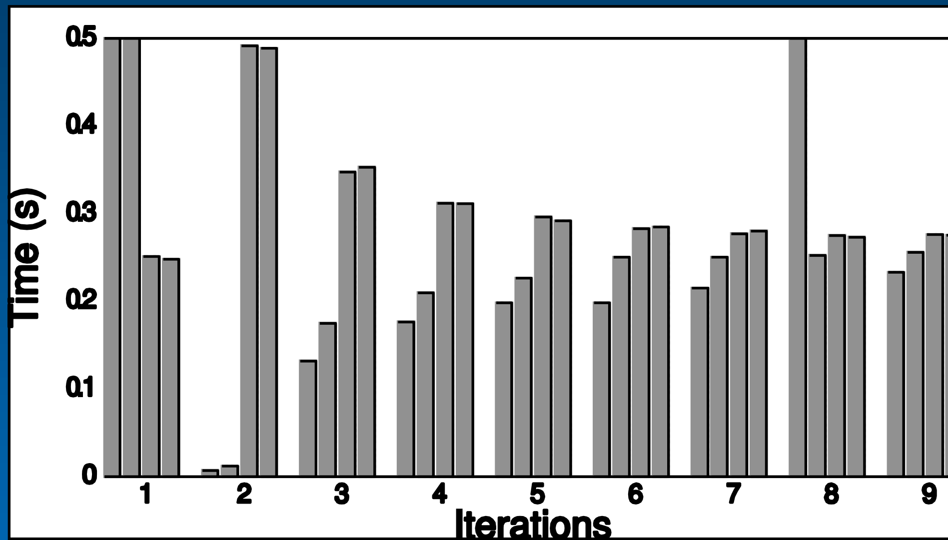
  Function approximation updated by adding the point

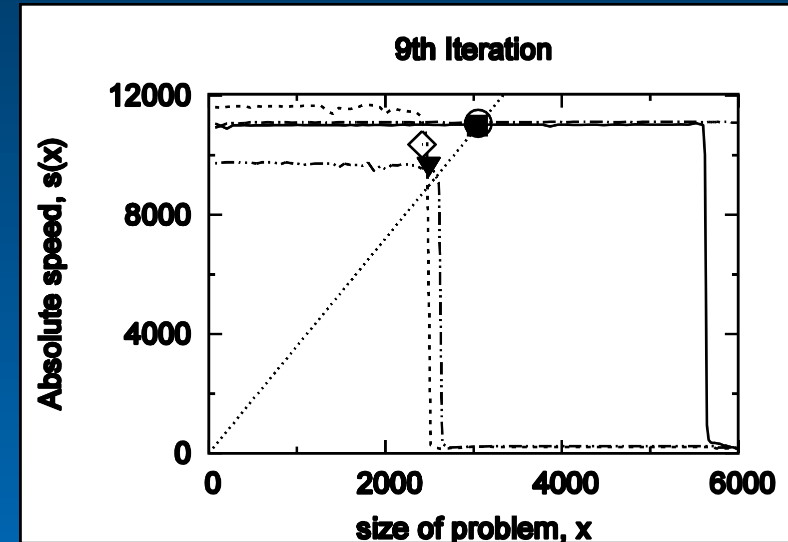


Experiments: Load balancing of iterative routines (ctd)



Experiments: Load balancing of iterative routines (ctd)





Experimental setup

- Heterogeneous cluster
  - 16 P4/Xeon/AMD/Celeron processors with Linux
    - » See http://hcl.ucd.ie/Hardware/Cluster+Specifications for detailed specs
  - 2 Gigabit interconnect
  - Software
    - » MPICH-1.2.5
    - » ATLAS
  - Processor speeds in million flop/s (C+=A×B, A=2560×16, B=16×2560)
    - » {7696, 5196, 7852, 14418, 8000, 8173, 7288, 7396, 9037, 8987, 13661, 14194, 11182, 14410, 12008, 15257}
    - » Indicative heterogeneity of the cluster ≈ 3

Conclusions

- New parallel computing platforms are built from increasingly heterogeneous processing devices
- Traditional heterogeneous parallel algorithms become less and less applicable
- Our solution: algorithms based on the functional performance models