Scientific Programming for Heterogeneous Networks

Alexey Lastovetsky
Dept. of Computer Science, University College Dublin
Alexey.Lastovetsky@ucd.ie

Alexey Kalinov
Inst. for System Programming, Russian Academy Of Sciences
ka@ispras.ru

Tutorial Subject

- Heterogeneous networks of computers
  - Becoming a popular platform for high performance scientific computing
  - New approaches are needed for efficient solution of scientific problems on this platform
- Our focus is on
  - Algorithms, models and programming tools for high performance computing in heterogeneous environments

Tutorial Outline

- Outline of hardware platforms
  - Classification
  - Typical uses of heterogeneous networks
- Programming issues: basic solutions
  - Heterogeneity of processors
    - » Design of heterogeneous parallel and distributed algorithms
    - » Analysis of heterogeneous algorithms
    - » Adaptation of homogeneous algorithms to heterogeneous platforms

Tutorial Outline

- Programming issues: basic solutions (ctd)
  - Implementation of heterogeneous and homogeneous algorithms on heterogeneous platforms
    - » Accuracy of the performance model of heterogeneous processors
    - » Programming systems for high performance heterogeneous computing
      - Parallel programming systems (mpC, HeteroMPI)
      - Grid programming systems (GridSolve)
      - Programming systems not suitable for HPHC

Tutorial Outline

- Programming issues: advanced topics
  - Communication layer
    - » Design of heterogeneous algorithms
    - » Implementation of heterogeneous and homogeneous algorithms on heterogeneous platforms
  - Memory heterogeneity and memory constraints
    - » Design of heterogeneous algorithms

Hardware platforms

Hardware platforms

- Heterogeneous hardware for parallel and distributed computing
  - Multiple processors
  - Communication network interconnecting the processors
  - Can be heterogeneous in many ways
    - » Only one way for a distributed memory multiprocessor system to be homogeneous

Homogeneous system

- Homogeneous system
  - Identical processors
  - Homogeneous communication network
    - » Links of the same latency and bandwidth between any pair of processors
  - Dedicated system
    - » One application at a time
  - Designed for high performance parallel computing
    - » Used to run a small number of similar applications

Homogeneity vs heterogeneity

- Homogeneity
  - Easy to break
    - » Multiprogramming makes the system heterogeneous
  - May be expansive to keep
    - » Cluster of commodity processors
      - Cheap alternative to vendor systems
      - Should be a dedicated system to stay homogeneous
      - Cannot be upgraded in part
- Heterogeneity is natural

Taxonomy of heterogeneous systems

- Heterogeneous systems (in the increasing order of heterogeneity and complexity)
  - Heterogeneous clusters
  - Local networks of computers
  - Organizational networks of computers
  - Global general purpose networks of computers

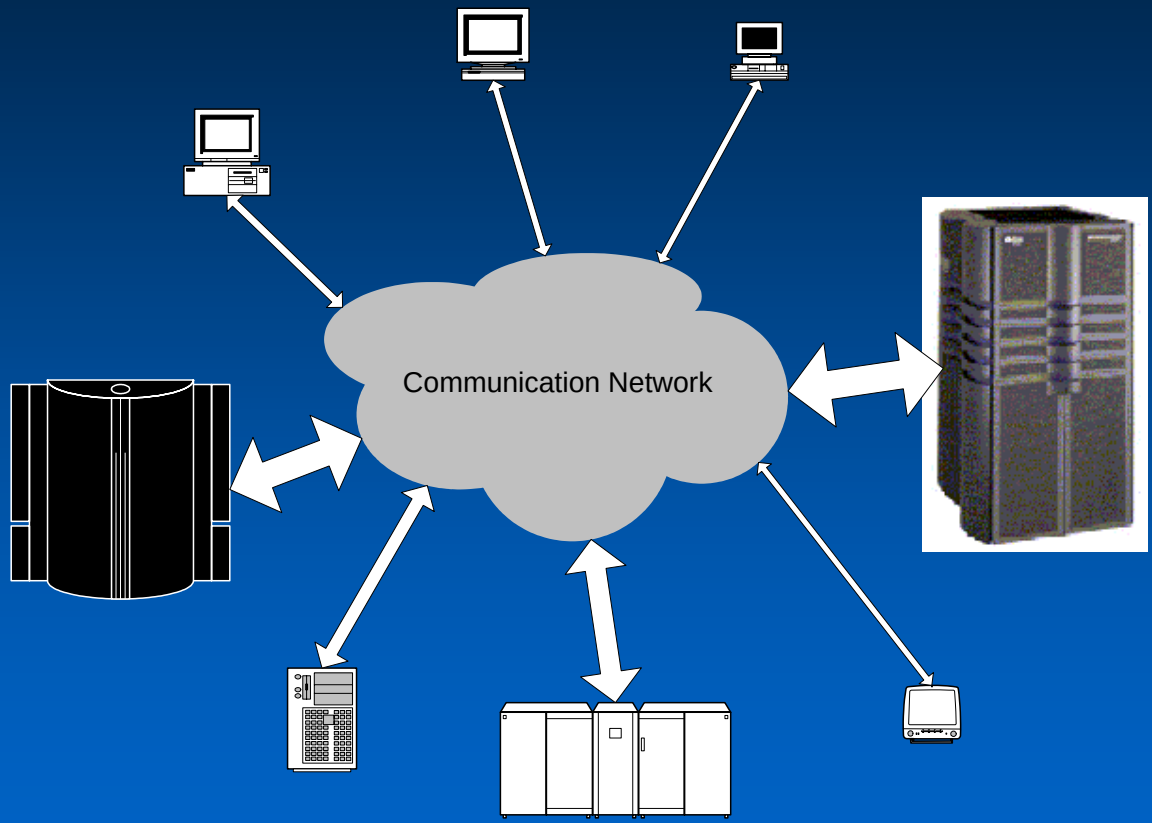
Heterogeneous cluster

- Heterogeneous cluster
  - Still a dedicated system designed for parallel computing, but
    - » Processors may not be identical
    - » Communication network may have a regular but heterogeneous structure
      - Faster segments interconnected by slower links
    - » May be a multi-user system
      - Still dedicated to high performance computing
      - Dynamic performance characteristics of processors

Heterogeneous cluster (ctd)

- Heterogeneity of processors
  - Different architectures
  - Different models of the same architecture
  - The same architecture and model but running different operating systems
  - The same architecture, model and OS but different configurations or basic software (compilers, run-time libraries, etc.)

Local network of computers (LNC)



Local network of computers (LNC)

LNC (ctd)

- LNC is a multi-user systems by nature
- Like highly heterogeneous clusters, LNC
  - Consists of processors of different architectures
    - » Dynamically changing their performance characteristics
  - Interconnected via heterogeneous communication network
- Unlike HCs, LNC
  - General-purpose computer system
    - » Typically associated with an individual organization

LNC: communication network

- Communication network of a typical LNC
  - Primary factors determining the structure
    - » structure of the organisation, tasks that are solved, security requirements, construction restrictions, budget limitations, qualification of technical personnel, etc.
  - High performance computing
    - » secondary factor if considered at all
  - Constantly developing
    - » occasionally and incrementally
    - » the communication network reflects the evolution of the organization rather than its current snapshot

LNC: communication network (ctd)

- As a result, the communication network
  - May be extremely heterogeneous
  - May have links of low speed and/or narrow bandwidth

LNC: functional heterogeneity

- Different computers may have different functions
  - Relatively isolated
  - Providing services to other computers in the LNC
  - Providing services to local and external computers
- Different functions => different levels of integration into the network
  - The heavier the integration, the less predictable the performance characteristics

LNC: functional heterogeneity (ctd)

- A heavy server may be configured differently
  - Often configured to avoid paging
    - » Leads to abnormal termination of some applications
      - If they do not fit into the main memory
    - » The loss of continuity of performance characteristics

LNC: decentralized system

- Components of a LNC are not as strongly integrated and controlled as in HCs
  - Relatively autonomous computers
    - » Used and administered independently
  - Configuration of LNC is dynamic
    - » Computer can come and go
      - Users switch them on and off
      - Users may re-boot computers

Global network of computers

- Global network of computers (GNC)
  - Includes geographically distributed computers
- Three main types of GNCs
  - Dedicated system for HPC
    - » Consists of several interconnected homogeneous systems and/or HCs
      - Similar to HCs
  - Organizational network
  - General-purpose global network

GNC (ctd)

- Organizational GNC
  - Comprises computer resources of some individual organization
  - Can be seen as geographically extended LNC
  - Typically very well managed
  - Typically of higher level of centralization, integration and uniformity than LNCs
- Organizational GNCs are similar to LNCs

GNC (ctd)

- General-purpose GNC
  - Consists of individual computers interconnected via Internet
    - » Each computer is managed independently
    - » The most heterogeneous, irregular, loosely integrated and dynamic type of heterogeneous network
- Some other heterogeneous sets of interconnected processing devices (not for scientific computing)
  - Mobile telecommunication system
  - Embedded control multiprocessor systems

Typical Uses of Heterogeneous Networks

Uses of heterogeneous networks

- Heterogeneous networks are used
  - Traditionally
  - For parallel computing
  - For distributed computing

Uses of heterogeneous networks (ctd)

- Traditional use
  - The network is used as an extension of the user's computer
    - » The computer may be serial or parallel
    - » Applications are traditional
      - May be run on the user's computer
      - Data and code are provided by the user
    - » Applications may be run on any relevant computer of the network
      - Scheduled by an operating environment
      - Aimed at better utilization of the computing resources (rather that at higher performance)

Uses of heterogeneous networks (ctd)

- Parallel computing
  - The network is used as a parallel computing system
    - » The user provides a dedicated parallel application
      - The (source) code and data are provided by the user
      - The source code is sent to the computers where it is locally compiled
      - All computers are supposed to have all libraries necessary to produce local executables
  - High performance is the main goal of the use

Uses of heterogeneous networks (ctd)

- Distributed computing
  - Some components of the application are not available on the user's computer
- Some code may be only available on remote computers
  - Lack of resources to execute the code
  - Too much efforts and/or resources needed compared to the frequency of its execution
  - Not available for installation
  - Meaningful only if executed remotely (ATM)

Uses of heterogeneous networks (ctd)

- Some components of input data cannot be provided by the user and reside on remote storage devices
  - The size of data is too big for the disk storage of the user's computer
  - The input data are provided by a third party
    - » Remote scientific device, remote data base, remote application
- Both some code and data are not available

Programming issues

- Programming issues
  - Performance
    - » Primary for parallel programming
    - » High priority for distributed programming
      - Primary for high performance DP
  - Fault tolerance
    - » Primary for distributed computing
    - » Never used to be primary for parallel computing
      - Getting primary for large scale parallel computing and heterogeneous parallel computing

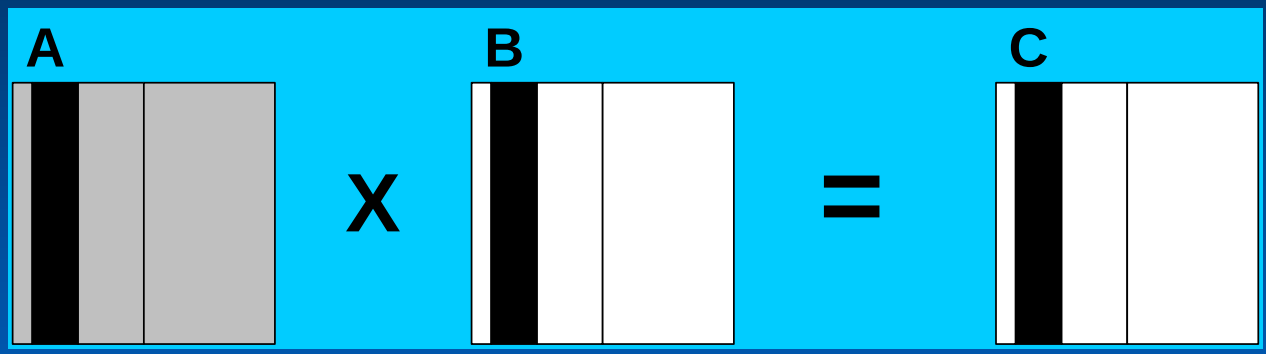
Performance issues

Processors Heterogeneity

- Heterogeneity of processors
  - The processors run at different speeds
    - » A good parallel program for MPPs evenly distributes workload
    - » Ported to *heterogeneous cluster*, the program will align the performance with the slowest processor
    - » A good parallel application for a NoC must distribute computations unevenly
      - taking into account the difference in processor speed
      - Ideally, the volume of computation performed by a processor should be proportional to its speed

Processors Heterogeneity (ctd)

- Example. Multiplication of two dense $n \times n$ matrices A, B on a p-processor heterogeneous cluster.



- ✓ Matrices A, B, and C unevenly partitioned in one dimension
- ✓ The area of the slice mapped to each processor is proportional to its speed
- ✓ The slices mapped onto a single processor are shaded black
- ✓ During execution, this processor requires all of matrix A (shaded gray)

Data partitioning over heterogeneous processors

- The problem of distribution of computations in proportion to the speed of processors
  - Reduced to partitioning of mathematical objects
    - » sets, matrices, graphs, etc.
  - Many interesting problems have been formulated and investigated over last 10 years
  - Only first steps have been taken

Data partitioning over heterogeneous processors (ctd)

- Typical partitioning problem in a generic form
  - Given a set of processors, the speed of each of which is characterized by a positive constant
  - Partition a mathematical object into sub-objects
    - » 1-to-1 mapping between partitions and processors
    - » The size of each partition ~ the speed of the processor
      - Number of elements in the sub-set or sub-matrix, the number of nodes in the sub-graph
      - Implicit assumption: the volume of computation ~ the size of the processed mathematical object

Data partitioning over heterogeneous processors (ctd)

- Typical partitioning problem (ctd)
  - Some additional restrictions on the partitions to be satisfied
    - » Sub-matrices to form 2-D $p \times q$ arrangement (p and q may be given constants or parameters of the problem, the optimal value of which should be also found)
  - The partitioning minimizes some functional used to measure each partitioning
    - » Minimize the sum of perimeters of the rectangles representing sub-matrices (characterizes the volume of communications for some algorithms)

Partitioning matrices

- Matrices
  - The investigated problems mainly deal with matrices
  - The most widely used in scientific computing
- Matrix partitioning problems investigated
  - General
    - » Partitioning a square into rectangles
      - With no restrictions on partitions
      - Minimizing the sum of perimeters
  - Column-based
    - » General + vertical rectangles of the same width
  - Column-based with a given 2D arrangement of rectangles

Partitioning matrices (ctd)

- General matrix partitioning problem
  - NP-complete
- Column-based
  - Polynomial complexity (no worse than $O(p^3)$)
- Column-based with a given arrangement of rectangles
  - Given arrangement of the processors
    - » Linear complexity
  - Optimal mapping of processors should be found
    - » NP-complete
    - » Optimal solution will arrange the processors in the decreasing order of the speed along each dimension

Partitioning matrices (ctd)

- Cartesian with a given arrangement of rectangles
  - Given arrangement of the processors
    - » NP-complete
  - Optimal mapping of processors should be found
    - » NP-complete
  - Heuristic algorithms
    - » Natural decomposition
      - Arrange the processors in the decreasing order of the speed along each dimension
      - Partition independently for each dimension in proportion with the sum of speeds along each row and column of the grid of the processors

Partitioning matrices (ctd)

- What is the optimal shape of arrangement of the processors?
  - Open problem, no theoretical analysis yet
  - Experimental results
    - » The shape does impact the efficiency if the rest parameters of the algorithm are not optimal
    - » No impact observed in the case of optimal parameters

- More questions than answers. More problems than solutions. Much more research needed

Heterogeneous algorithms: Case study

Heterogeneous computation distribution

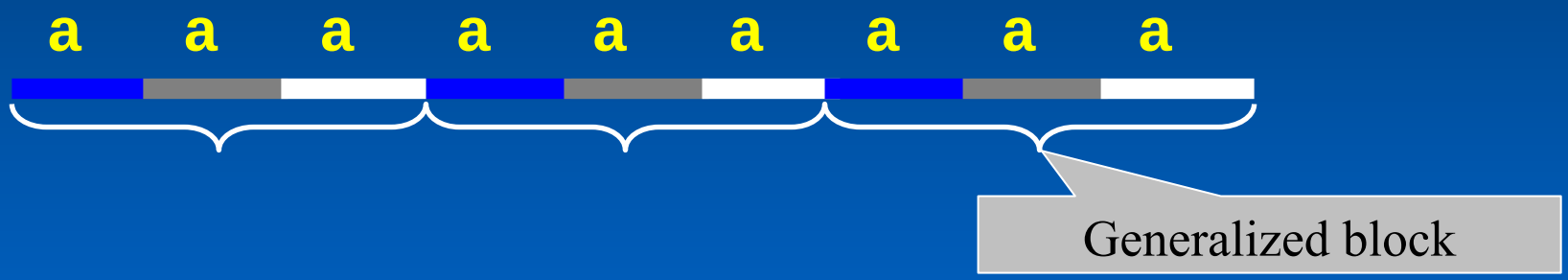
- Simplest model of performance – a constant representing relative performance
- Static distributions
- Possible effect of heterogeneous distribution

$$speedup_{ideal}^{het} = \frac{t_{ho}}{t_{he}} = \frac{r_{aver}}{r_{\min}}.$$

r_{aver} - average, $r_{\min}$ - minimal performance

1D1D block-cyclic distribution

block size **a**
number of processes **e**
data space **N**
element **k** is stored in **(k-1)/a mod e** process



Homogeneous 1D block-cyclic distribution of 1D data space (**N=9·a**) with block size **a** over 3 processes

1D1D heterogeneous distribution

average block size **a**
number of processes **e**
generalized block size **s=a·e**
set of process performance **r={r_i}**

$$n_i = \frac{r_i}{\sum_{i=0}^{e-1} r_i} \cdot (ne)$$

Natural 1D distribution of 1D generalized block over 3 processes with **r={1,2,3}**.

block-cyclic 2D2D distribution

	0	1	2	3	4	5
0	0,0	0,0	0,1	0,1	0,2	0,2
1	0,0	0,0	0,1	0,1	0,2	0,2
2	1,0	1,0	1,1	1,1	1,2	1,2
3	1,0	1,0	1,1	1,1	1,2	1,2
4	0,0	0,0	0,1	0,1	0,2	0,2
5	0,0	0,0	0,1	0,1	0,2	0,2
6	1,0	1,0	1,1	1,1	1,2	1,2
7	1,0	1,0	1,1	1,1	1,2	1,2

(a)

Generalized block

	0	1	2	3	4	5
0	0,0	0,0	0,1	0,1	0,2	0,2
1	0,0	0,0	0,1	0,1	0,2	0,2
4	0,0	0,0	0,1	0,1	0,2	0,2
5	0,0	0,0	0,1	0,1	0,2	0,2
2	1,0	1,0	1,1	1,1	1,2	1,2
3	1,0	1,0	1,1	1,1	1,2	1,2
6	1,0	1,0	1,1	1,1	1,2	1,2
7	1,0	1,0	1,1	1,1	1,2	1,2

(b)

Homogeneous block-cyclic distribution of matrix 8x6 over process grid 2x3 with block 2x2

Optimal 2D2D heterogeneous distribution

average block size **m×n**
process grid **P×Q**
generalized block size **S=mP×nQ**
matrix of processes performance **R={r_{ij}}**

$$m_{ij} \cdot n_{ij} = \frac{r_{ij}}{\sum_{i=0}^{P-1} \sum_{j=0}^{Q-1} r_{ij}} \cdot mP \cdot nQ$$

$$\sum_i m_{ij} = mP,$$
$$\sum_j n_{ij} = nQ$$

Column based optimal distribution

$$m_{ij} = \frac{r_{ij}}{\sum_{i=0}^{P-1} r_{ij}} \cdot mP;$$

$$n_{ij} = n_j = \frac{\sum_{i=0}^{P-1} r_{ij}}{\sum_{i=0}^{P-1} \sum_{j=0}^{Q-1} r_{ij}} \cdot nQ.$$

	0	1	2	3	4	5
0	0,0	0,0	0,0	0,1	0,1	0,2
1	0,0	0,0	0,0	0,1	0,1	0,2
2	1,0	1,0	1,0	1,1	1,1	0,2
3	1,0	1,0	1,0	1,1	1,1	1,2

Column based distribution of generalized block over process grid 2x3 with average block 2x2 and matrix of process performances

$$R = \begin{Bmatrix} 6 & 4 & 2 \\ 5 & 3 & 1 \end{Bmatrix}.$$

Column based optimal distribution (ctd.)

- Advantages
  - Optimal distribution of computations
- Disadvantages
  - The communication pattern is worse than in homogeneous case
  - Requires a lot of efforts for rewriting of applications with homogeneous distribution

Cartesian nDmD distribution

- Mapping set of processes into **m**-dimensional process grid
- Distributing **n**-dimensional data over the process grid

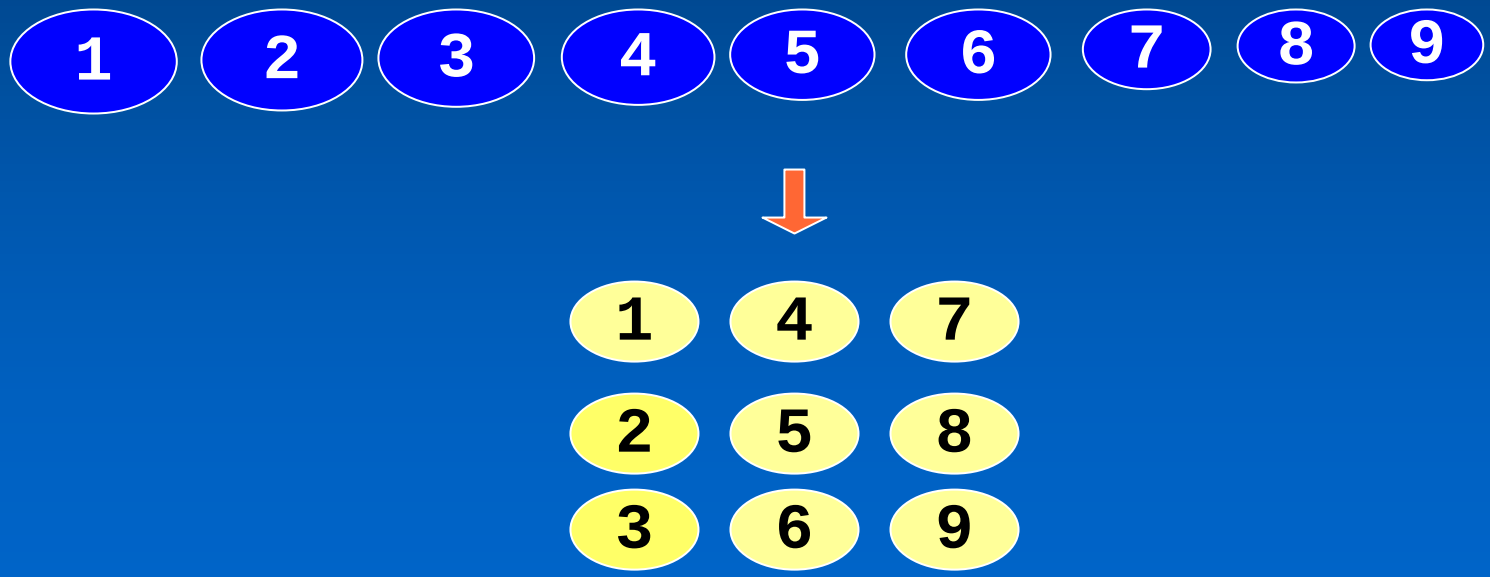
Starting with 2D process grid heterogeneous decomposition is NP-hard

The heuristics approaches are required

Process mapping

- The simplest heuristics

  - arranging the set of processes in ascending order according to process performances



Natural 2D2D data distribution

$$m_i = \frac{\sum_{j=0}^{Q-1} r_{ij}}{\sum_{i=0}^{P-1}\sum_{j=0}^{Q-1} r_{ij}} \cdot mP,$$

$$n_j = \frac{\sum_{i=0}^{P-1} r_{ij}}{\sum_{i=0}^{P-1}\sum_{j=0}^{Q-1} r_{ij}} \cdot nQ.$$

	0	1	2	3	4	5
0	0,0	0,0	0,0	0,1	0,1	0,2
1	0,0	0,0	0,0	0,1	0,1	0,2
2	1,0	1,0	1,0	1,1	1,1	1,2
3	1,0	1,0	1,0	1,1	1,1	1,2

Natural distribution of generalized block over process grid 2x3 with average block 2x2 and matrix of process performances

$$R = \begin{Bmatrix} 6 & 4 & 2 \\ 5 & 3 & 1 \end{Bmatrix}.$$

Distributions comparison

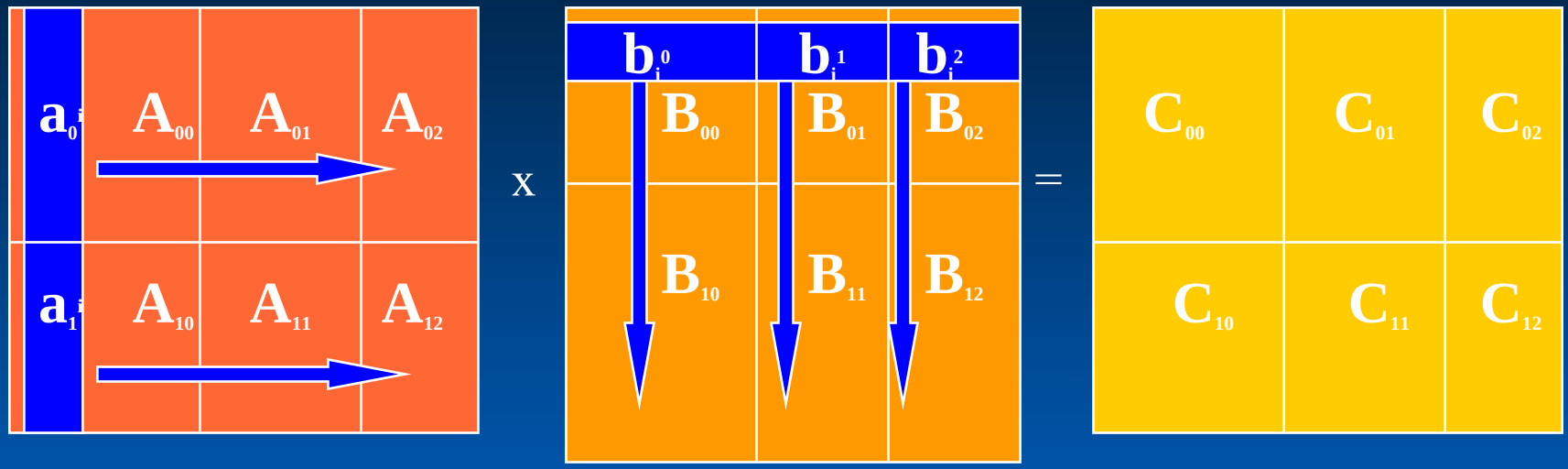
- Criteria of comparison

$$\max_{(i,j)}\left(\frac{m_{ij}n_{ij}}{mPnQ}-\frac{r_{ij}}{\sum_i\sum_j r_{ij}}\right)$$

Column based (round-off errors) – 0,03

Natural (method + round-off errors) – 0,036

Example: Matrix multiplication




```
for (I=0; I<k; k++) {
    selecting processes row and column owning a^i and b_i
    broadcast a^i
    broadcast b_i
    C=C+a^i x b_i^t
}
```

SUMMA

Scalable Universal Matrix Multiplication Algorithm – broadcast as shift over ring: pipelining of computations and communications.

Model of communications: t=a+bL, a – latency, b – bandwidth.

Assumptions for performance analysis: squire matrices, squire process grid; matrices dimension **N** >> number of processes **p**; non-blocking algorithm.

Time complexity (homogeneous system)

Time complexity for homogeneous distribution with parallel point-to-point communications

$$T_p(N,p) = N\left(\frac{N^2}{pr} + 2\left(a + \frac{N}{\sqrt{p}}b\right)\right) = \frac{N^3}{pr} + 2N\left(a + \frac{N}{\sqrt{p}}b\right)$$

Time complexity for homogeneous distribution with sequential point-to-point communications

$$T_p(N,p) = N\left(\frac{N^2}{pr} + 2\sqrt{p}\,(a + Nb)\right) = \frac{N^3}{pr} + 2N\sqrt{p}\,(a + Nb)$$

Time complexity (heterogeneous system)

Time complexity for homogeneous distribution with parallel point-to-point communications

$$T^{ho}(N,p) = \frac{N^3}{pr_{\min}} + 2N\left(a + \frac{N}{\sqrt{p}}b\right)$$

Time complexity for homogeneous distribution with sequential point-to-point communications

$$T^{ho}(N,p) = \frac{N^3}{pr_{\min}} + 2N\sqrt{p}(a + Nb)$$

Why heterogeneous distribution of data

- Volume of computations ~ N^3
- Ideal: volume of computation is proportional to performance
- N steps of algorithm
- от each step volume of computation ~ volume of data
- general approach
- true only for simplest model of processor performance

Heterogeneous distribution and SUMMA

We can think that heterogeneous distribution provide ideal distribution of computations

Time of local computations

$$T_{comp}(N,p) = \frac{N^3}{pr_{aver}}$$

Time complexity for heterogeneous distribution with sequential point-to-point communications

$$T^{he}(N,p) = \frac{N^3}{pr_{aver}} + 2N\sqrt{p}(a+Nb)$$

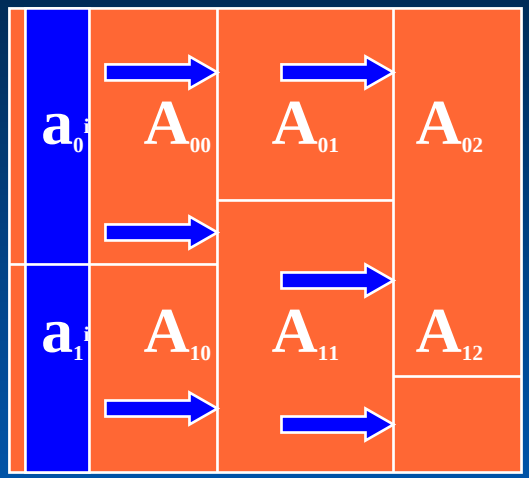
Heterogeneous distribution (ctd)

Sequential point-to-point communications

$$T^{he}(N,p) = \frac{N^3}{pr_{aver}} + 2N\sqrt{p}(a + Nb)$$

$$T^{ho}(N,p) = \frac{N^3}{pr_{\min}} + 2N\sqrt{p}(a + Nb)$$

Column based distribution and SUMMA



Column broadcast is not independent for process grid rows even for communication platform with parallel point-to-point communications

$$T_{column\_bcast}(N, p) = N\sqrt{p}(a + Nb)$$

Column based distribution (ctd)

$$n_j = \frac{\sum_i r_{ij}}{\sum_i \sum_j r_{ij}} N \le \frac{\sqrt{p} r_{\max}}{p r_{aver}} N \le \frac{r_{aver}}{r_{\min}} \frac{\sqrt{p} r_{\max}}{p r_{aver}} N = \frac{r_{\max}}{r_{\min}} \frac{N}{\sqrt{p}}.$$

Time complexity for heterogeneous distribution with parallel point-to-point communications

$$T^{col}(N,p) == \frac{N^3}{p r_{aver}} + 2N\sqrt{p}a + N^2\left(1 + \frac{r_{\max}}{r_{\min}\sqrt{p}}\right)b$$

Column based distribution (ctd)

$$T^{col}(N,p) == \frac{N^3}{pr_{aver}} + 2N\sqrt{p}a + N^2\left(1 + \frac{r_{\max}}{r_{\min}\sqrt{p}}\right)b$$

$$T^{ho}(N,p) = \frac{N^3}{pr_{\min}} + 2Na + \frac{2N^2}{\sqrt{p}}b$$

Natural decomposition

Parallel point-to-point communications

$$T^{nat}(N,p) == \frac{N^3}{pr_{aver}} + 2N\left(\alpha + \frac{r_{\max}}{r_{\min}}\frac{N}{\sqrt{p}}b\right)$$

Natural decomposition (ctd)

$$T^{nat}(N,p) == \frac{N^3}{pr_{aver}} + 2N\alpha + \frac{r_{\max}}{r_{\min}}\frac{2N^2}{\sqrt{p}}b$$

$$T^{ho}(N,p) = \frac{N^3}{pr_{\min}} + 2Na + \frac{2N^2}{\sqrt{p}}b$$

SUMMA summary

Parallel point-to-point communications

Homogeneous

$$T^{ho}(N,p) = \frac{N^3}{pr_{\min}} + 2N\left(a + \frac{N}{\sqrt{p}}b\right)$$

Column-based

$$T^{col}(N,p) = \frac{N^3}{pr_{aver}} + 2N\sqrt{p}a + N^2\left(1 + \frac{r_{\max}}{r_{\min}\sqrt{p}}\right)b$$

Natural

$$T^{nat}(N,p) == \frac{N^3}{pr_{aver}} + 2N\left(\alpha + \frac{r_{\max}}{r_{\min}}\frac{N}{\sqrt{p}}b\right)$$

Communication Network

- Communication network interconnecting heterogeneous processors
  - Has a significant impact on optimal distribution of computations over the processors
  - Can only be ignored if the contribution of communication operations into the total execution time << that of computations

Communication Network (ctd)

- Optimal distribution of computation taking account of communication links
  - Up to p^2 communication links in addition to p processors
    - » Increases the complexity of the problem even if the optimal distribution involves all available processors
  - If some links are of low latency/bandwidth
    - » May be profitable to involve not all available processors
    - » The problem becomes extremely complex
      - All subsets of the set of processors should be tested

Communication Network (ctd)

- Taking account of communication operations
  - The relative speed of processors makes no sense
    - » Cannot help to estimate the cost of computations
  - The processor speed has to be absolute
    - » Volume of computations per time unit
      - Given a volume of computations, the execution time of the computations can be calculated
  - Similarly, communication link have to be characterized by absolute units of performance

Communication Network (ctd)

- Communication links
  - No problem with absolute units of performance
    - » Byte/sec
- Processors
  - How to measure the absolute speed of a processor?
    - » The same units as in complexity analysis, arithmetical operations per sec (Mflop/sec)?
      - An average characteristic (not accurate), processors with the same Mflops/sec may execute the same application in significantly different times

Communication Network (ctd)

- A solution
  - ↑ granularity of computation units + make them sensitive to computations of the application
    - » Measure the processor speed in the context of the application
  - Proposed in mpC
    - » computation unit is a piece of code provided by the programmer
      - The code is supposed to be representative for the application → given the volume of computation in the units, the execution time can be calculated

Performance Analysis

- Performance analysis of homogeneous parallel algorithms
  - Based on some model of parallel computers (LogP,…)
    - » All assume a parallel computer to be a homogeneous multiprocessor
  - Includes 2 steps
    - » Theoretical analysis of the algorithm
    - » A relatively small number of experiments on a homogeneous parallel computer system to demonstrate that
      - The analysis is correct
      - The algorithm is faster than its counterparts

Performance Analysis (ctd)

- Performance analysis of heterogeneous parallel algorithms is a much more difficult task
  - No adequate and practical model of HNOCs able to predict the execution time of heterogeneous parallel algorithms with satisfactory accuracy

Performance Analysis (ctd)

- One approach
  - Analyse the derived data partitioning problem instead of the original one
    - » Typically, NP-complete
    - » Sub-optimal solutions are proposed and analyzed
      - The analysis is typically statistical: the sub-optimal solutions for a big number of generated inputs are compared to one other and the optimal one
  - Experiments are still needed

Performance Analysis (ctd)

- Another approach to assessment of a heterogeneous parallel algorithm
  - Experimental comparison with some homogeneous counterpart on one or several heterogeneous platforms
  - Different heterogeneous algorithms are also compared mostly experimentally
  - Not as convincing as for homogeneous ones
    - » If algorithm A is better than algorithm B on some HNOCs, it does not prove that A will be better on other HNOCs with essentially different performance characteristics

Performance Analysis (ctd)

- The third approach
  - A small number of carefully designed experiments
    - » The efficiency of the heterogeneous parallel algorithm is compared with some experimentally obtained ideal efficiency
      - The efficiency of its homogeneous prototype in an equally powerful homogeneous environment
  - Directly estimation of the efficiency
  - Relatively high confidence in the results

Performance analysis (ctd)

- A typical heterogeneous parallel algorithm
  - A modification of some homogeneous one
  - The approach assess the heterogeneous modification rather than the algorithm as an isolated entity
- Basic postulate
  - **The heterogeneous algorithm cannot be more efficient than its homogeneous prototype**
- See more on this approach in Lastovetsky&Reddy, Parallel Computing 30(11), 2004.

Processors Heterogeneity (ctd)

- Alternative approach to parallel programming for heterogeneous networks
  - Even distribution of computation across processes with uneven distribution over the processors
    - » Partition the whole computation into a large number of equal chunks
    - » Each chunk is performed by a separate process
    - » The number of processes run by each processor proportional to the relative speed of the processor

Implementation issues

Accuracy of Performance Model

- **The efficiency of any application unevenly distributing computations over heterogeneous processors strongly depends on the accuracy of estimation of the relative speed of the processors**
  - If this estimation is not accurate, the load of processors will be unbalanced
- Traditional approach
  - Run some test code once
  - Use the estimation when distributing computations

Accuracy of Performance Model (ctd)

- Two processors of the same architecture only differing in the clock rate
  - No problem to accurately estimate their relative speed
    - » The relative speed will be the same for any application
- Processors of different architectures
  - Differ in everything
    - » set of instructions, number of IEUs, number of registers, memory hierarchy, size of each memory level, and so on
  - May demonstrate different relative speeds for different applications

Accuracy of Performance Model (ctd)

- Example. Consider two implementations of a 500x500 matrix Cholesky factorisation:
  - The code

```
for(k=0; k<500; k++) {
  for(i=k, lkk=sqrt(a[k][k]); i<500; i++)
    a[i][k] /= lkk;
  for(j=k+1; j<500; j++)
    for(i=j; i<500; i++)
      a[i][j] -= a[i][k]*a[j][k];
}
```

  estimated the relative speed of SPARCstation-5 and SPARCstation-20 as 10:9

Accuracy of Performance Model (ctd)

- Example. (ctd)
  - The code

```
for(k=0; k<500; k++) {
  for(i=k, lkk=sqrt(a[k][k]); i<500; i++)
    a[i][k] /= lkk;
  for(i=k+1; i<500; i++)
    for(j=i; j<500; j++)
      a[i][j] -= a[k][j]*a[k][i];
}
```

    estimated their relative speed as 10:14
  - Routine `dptof2` from LAPACK, solving the same problem, estimated their relative speed as 10:10

Accuracy of Performance Model (ctd)

- Multiprocessing is another complication
  - The real performance can dynamically change depending on external computations and communications
  - One approach to the problem
    - » Measure the level of the current workload of each processor, $u \in [0,1]$
    - » Correct the relative speed by multiplying by the coefficient

Accuracy of Performance Model (ctd)

- A combined approach to the 2 problems (mpC)
  - Test code is provided by the application programmer
    - » Should be representative for the application and relatively small
    - » The recon statement is used to specify the code and where and when it should be run
    - » Execution of each recon statement
      - All processors in parallel execute the test code
      - The time elapsed is used to calculate their relative speeds
  - The accuracy of the estimation of the relative speed of the processors is controlled by the application programmer

Programming Systems for HNOCs

- Programming systems
  - For parallel computing
    - » Traditional systems (MPI, HPF) do not address the challenges of heterogeneous parallel computing
    - » mpC, HeteroMPI
  - For high performance distributed computing
    - » NetSolve/GridSolve

mpC

- mpC
  - A programming language for parallel computing on heterogeneous networks
    - » High level language abstractions to avoid going into details of message passing programming model of the MPI level
    - » Takes care of the optimal mapping of the algorithm to the computers of the network

mpC (ctd)

- The mapping
  - Performed at runtime
  - Based on two performance models
    - » The PM of the implemented algorithm
      - Provided by the application programmer
        - a part of the mpC program
    - » The PM of the network of computers

mpC (ctd)

- The PM of the network of computers
  - Each processor is characterized by the execution time of the same serial code
    - » The code is provided by the application programmer
    - » Representative for computations of the application
    - » Performed at runtime at the points specified by the application programmer
    - » *The PM of the processors provides current estimation of their speed demonstrated on the code representative for the particular application*

mpC (ctd)

- The PM of the network of computers (ctd)
  - The communication model
    - » A hierarchy of homogeneous communication layers
      - Each characterized by the latency and bandwidth
      - Static (obtained once and forever)

mpC (ctd)

- The PM of the algorithm
  - The number of processes executing the algorithm (typically, a parameter of the PM)
  - The total volume of computation to be performed by each process
    - » A formula including the parameters of the PM
    - » The volume is measured in computation units provided by the application programmer
      - The very code that has been used to measure the performance of the processors

mpC (ctd)

- The PM of the algorithm (ctd)
  - The total volume of data transferred between each pair of the processes
  - How the processes perform the computations and communications and interact
    - » In terms of traditional algorithmic patterns (for, while, parallel for, etc)
    - » Expressions in the statements specify not the computations and communications themselves but rather their amount
      - Parameters of the algorithm and locally declared variables can be used

mpC (ctd)

- The mpC compiler
  - Translates the description of the PM of the algorithm into the code which
    - » Calculates the execution time of the algorithm for each mapping of the processes to the computers of the network
    - » Input arguments of the code are the parameters of the PMs of both the algorithm and the network of computers

mpC (ctd)

- The mpC runtime system
  - Uses the generated code to
    - » Find the best mapping
      - If all parameters of the algorithm are specified by the programmer (the one that minimizes the execution time)
    - » Find the optimal number of processes and the best mapping of the fully specified algorithm
      - If the number of processes is not specified

HeteroMPI

- HeteroMPI
  - An extension of MPI
  - Programmer can describe the performance model of the implemented algorithm
    - » In a small model definition language shared with mpC
  - Given this description
    - » HeteroMPI tries to create a group of processes executing the algorithm faster than any other group

HeteroMPI (ctd)

- Standard MPI approach to group creation
  - Acceptable in homogeneous environments
    - » If there is one process per processor
    - » Any group will execute the algorithm with the same speed
  - Not acceptable
    - » In heterogeneous environments
    - » If there are more that one process per processor
- In HeteroMPI
  - The programmer can describe the algorithm
  - The description is translated into a set of functions
    - » Making up an algorithm-specific part of HMPI run-time system

HeteroMPI (ctd)

- A new operation to create a group of processes:

```
HMPI_Group_create(
  HMPI_Group* gid,
  const HMPI_Model* perf_model,
  const void* model_parameters)
```

- Collective operation
  - In the simplest case, called by all processes **HMPI_COMM_WORLD**

HeteroMPI (ctd)

- Dynamic update of the estimation of the processors speed can be performed by

```
HMPI_Recon(
    HMPI_Benchmark_function func,
    const void* input_p,
    int num_of_parameters,
    const void* output_p)
```

- Collective operation
  - Called by all processes of **HMPI_COMM_WORLD**

HeteroMPI (ctd)

- Prediction of the execution time of the algorithm

```
HMPI_Timeof(
   HMPI_Model *perf_model,
   const void* model_parameters)
```

- Local operation
  - Can be called by any processes

HeteroMPI (ctd)

- Another collective operation to create a group of processes:

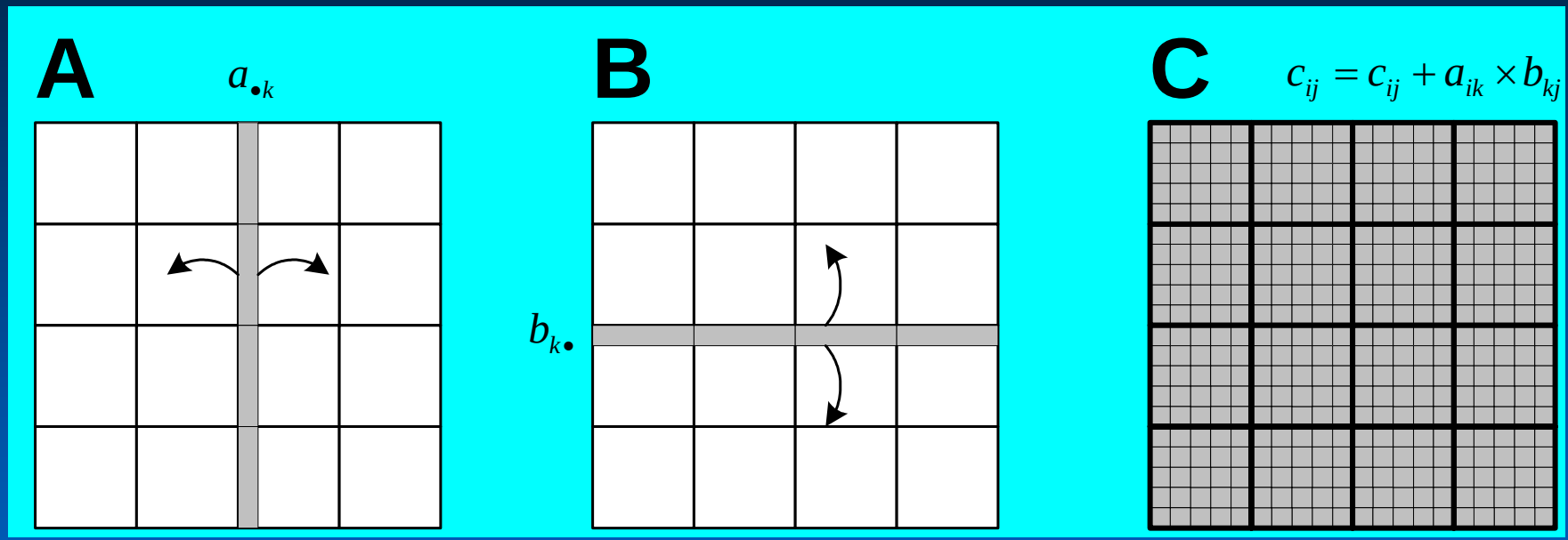
```
HMPI_Group_auto_create(
  HMPI_Group* gid,
  const HMPI_Model* perf_model,
  const void* model_parameters)
```

- Used if the programmer wants HeteroMPI to find the optimal number and arrangement of processes

Example: Matrix Multiplication on Heterogeneous Platforms

- We outline of a homogeneous parallel MM algorithm
- We describe its implementation in HeteroMPI for heterogeneous platforms

Parallel Matrix Multiplication



One step of the block algorithm of parallel multiplication of two dense *n*x*n* matrices on a *p*-processor MPP. Each element in *A*, *B*, and *C* is a square *rxr* block.

Parallel Matrix Multiplication (ctd)

Block cyclic algorithm: The blocks are scattered in a cyclic fashion along both dimensions of the 2D processor grid.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	P_{11}	P_{12}	P_{13}	P_{11}	P_{12}	P_{13}	P_{11}	P_{12}	P_{13}	P_{11}	P_{12}	P_{13}	P_{11}	P_{12}	P_{13}	P_{11}	P_{12}	P_{13}
2	P_{21}	P_{22}	P_{23}	P_{21}	P_{22}	P_{23}	P_{21}	P_{11}	P_{11}	P_{21}	P_{22}	P_{23}	P_{21}	P_{22}	P_{23}	P_{21}	P_{22}	P_{23}
3	P_{31}	P_{32}	P_{33}	P_{31}	P_{32}	P_{33}	P_{31}	P_{11}	P_{11}	P_{31}	P_{32}	P_{33}	P_{31}	P_{32}	P_{33}	P_{31}	P_{32}	P_{33}
4	P_{11}	P_{12}	P_{13}	P_{11}	P_{12}	P_{13}	P_{11}	P_{12}	P_{13}	P_{11}	P_{12}	P_{13}	P_{11}	P_{12}	P_{13}	P_{11}	P_{12}	P_{13}
5	P_{21}	P_{22}	P_{23}	P_{21}	P_{22}	P_{23}	P_{21}	P_{22}	P_{23}	P_{21}	P_{22}	P_{23}	P_{21}	P_{22}	P_{23}	P_{21}	P_{22}	P_{23}
6	P_{31}	P_{32}	P_{33}	P_{31}	P_{32}	P_{33}	P_{31}	P_{32}	P_{33}	P_{31}	P_{32}	P_{33}	P_{31}	P_{32}	P_{33}	P_{31}	P_{32}	P_{33}
7	P_{11}	P_{12}	P_{13}	P_{11}	P_{12}	P_{13}	P_{11}	P_{12}	P_{13}	P_{11}	P_{12}	P_{13}	P_{11}	P_{12}	P_{13}	P_{11}	P_{12}	P_{13}
8	P_{21}	P_{22}	P_{23}	P_{21}	P_{22}	P_{23}	P_{21}	P_{22}	P_{23}	P_{21}	P_{22}	P_{23}	P_{21}	P_{22}	P_{23}	P_{21}	P_{22}	P_{23}
9	P_{31}	P_{32}	P_{33}	P_{31}	P_{32}	P_{33}	P_{31}	P_{32}	P_{33}	P_{31}	P_{32}	P_{33}	P_{31}	P_{32}	P_{33}	P_{31}	P_{32}	P_{33}
10	P_{11}	P_{12}	P_{13}	P_{11}	P_{12}	P_{13}	P_{11}	P_{12}	P_{13}	P_{11}	P_{12}	P_{13}	P_{11}	P_{12}	P_{13}	P_{11}	P_{12}	P_{13}
11	P_{21}	P_{22}	P_{23}	P_{21}	P_{22}	P_{23}	P_{21}	P_{22}	P_{23}	P_{21}	P_{22}	P_{23}	P_{21}	P_{22}	P_{23}	P_{21}	P_{22}	P_{23}
12	P_{31}	P_{32}	P_{33}	P_{31}	P_{32}	P_{33}	P_{31}	P_{32}	P_{33}	P_{31}	P_{32}	P_{33}	P_{31}	P_{32}	P_{33}	P_{31}	P_{32}	P_{33}
13	P_{11}	P_{12}	P_{13}	P_{11}	P_{12}	P_{13}	P_{11}	P_{12}	P_{13}	P_{11}	P_{12}	P_{13}	P_{11}	P_{12}	P_{13}	P_{11}	P_{12}	P_{13}
14	P_{21}	P_{22}	P_{23}	P_{21}	P_{22}	P_{23}	P_{21}	P_{22}	P_{23}	P_{21}	P_{22}	P_{23}	P_{21}	P_{22}	P_{23}	P_{21}	P_{22}	P_{23}
15	P_{31}	P_{32}	P_{33}	P_{31}	P_{32}	P_{33}	P_{31}	P_{32}	P_{33}	P_{31}	P_{32}	P_{33}	P_{31}	P_{32}	P_{33}	P_{31}	P_{32}	P_{33}
16	P_{11}	P_{12}	P_{13}	P_{11}	P_{12}	P_{13}	P_{11}	P_{12}	P_{13}	P_{11}	P_{12}	P_{13}	P_{11}	P_{12}	P_{13}	P_{11}	P_{12}	P_{13}
17	P_{21}	P_{22}	P_{23}	P_{21}	P_{22}	P_{23}	P_{21}	P_{22}	P_{23}	P_{21}	P_{22}	P_{23}	P_{21}	P_{22}	P_{23}	P_{21}	P_{22}	P_{23}
18	P_{31}	P_{32}	P_{33}	P_{31}	P_{32}	P_{33}	P_{31}	P_{32}	P_{33}	P_{31}	P_{32}	P_{33}	P_{31}	P_{32}	P_{33}	P_{31}	P_{32}	P_{33}

Parallel Matrix Multiplication (ctd)

```
algorithm mxm(int n, int b, int p, int q)
{
  coord I=p, J=q;
  node
  {
    I>=0 && J>=0: bench*((n/(b*p))*(n/(b*q))*(n/b));
  };
  link (K=p, L=q)
  {
    (I!=K && J==L) || (I==K && J!=L) :
      length*((n/p)*(n/q)*sizeof(double)) [I, J]->[K, L];
  };
```

Parallel Matrix Multiplication (ctd)

```
scheme
{
  int i, j, k;
  for(k = 0; k < n; k+=b) {
    par(i = 0; i < p; i++)
      par(j = 0; j < q; j++)
        if (j != ((k/b)%q))
          (100.0/(n/(b*q))) %% [i,((k/b)%q)]->[i,j];
    par(i = 0; i < p; i++)
      par(j = 0; j < q; j++)
        if (i != ((k/b)%p))
          (100.0/(n/(b*p))) %% [((k/b)%p),j]->[i,j];
    par(i = 0; i < p; i++)
      par(j = 0; j < q; i++)
        ((100.0×b)/n) %% [i,j];
  }
};
};
```

Parallel Matrix Multiplication (ctd)

```
int main(int argc, char **argv) {
   static int p, q, n, b;
   void *model_params;
   HMPI_Group gid;
   HMPI_Init(argc, argv);
   // Estimation of speeds of the processors
   if(HMPI_Is_member(HMPI_COMM_WORLD_GROUP)
     HMPI_Recon(…);
   // Model parameter initialization
   if(HMPI_Is_host()) {
     model_params[0] = n;
     model_params[1] = b;
   }
   // HMPI Group creation
   if(HMPI_Is_host())
     HMPI_Group_auto_create(&gid, &HMPI_Model_mxm, model_params);
   if (HMPI_Is_free())
          HMPI_Group_auto_create(&gid, &HMPI_Model_mxm, NULL);
```

Parallel Matrix Multiplication (ctd)

```
    // Execution of the algorithm
    if(HMPI_Is_member(&gid)) {
      MPI_Comm algocomm = *(MPI_Comm*)HMPI_Get_comm(&gid);
      HMPI_Group_topology(&gid, &nd, dp);
      p = (*dp)[0];
      q = (*dp)[1];
      mxm(algocomm,…);
    }
    // HMPI Group Destruction
    if(HMPI_Is_member(&gid))
    HMPI_Group_free(&gid);
    HMPI_Finalize(0);
}
```

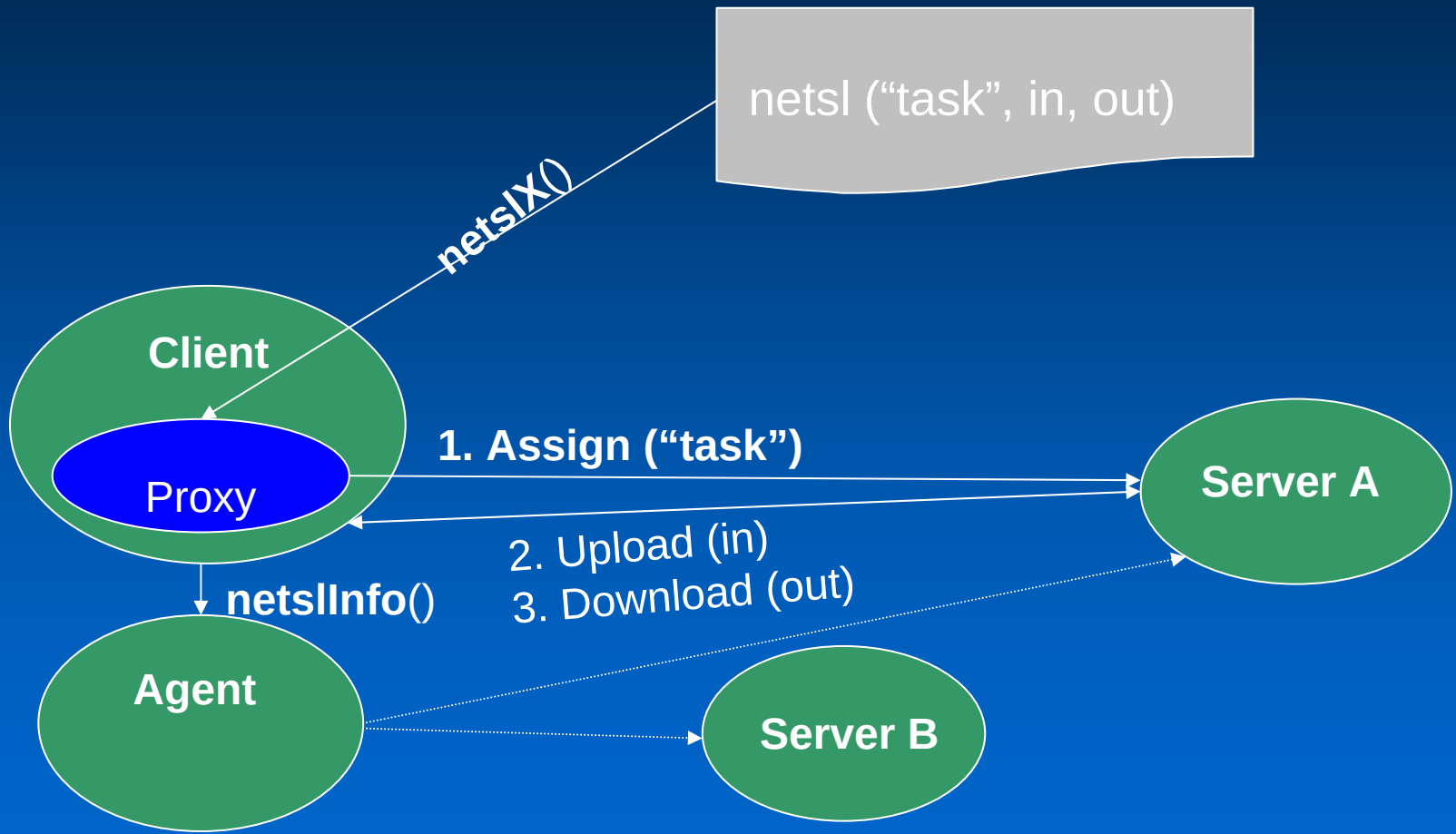
NetSolve

- NetSolve
  - Programming system for HPDC on global networks
    - » Based on the RPC mechanism
  - Some components of the application are only available on remote computers
- NetSolve application
  - The user writes a client program
    - » Any program (in C, Fortran, etc) with calls the NetSolve client interface
    - » Each call specifies
      - Remote task
      - Location of the input data on the user's computer
      - Location of the output data (on the user's computer)

NetSolve (ctd)

- Execution of the NetSolve application
  - A NetSolve call results in
    - » A task to be executed on a remote computer
    - » The NetSolve programming system
      - Selects the remote computer
      - Transfers input data to the remote computer
      - Delivers output data to the user's computer
  - The mapping of the remote tasks to computers
    - » The core operation having an impact on the performance of the application

NetSolve (ctd)



NetSolve (ctd)

- Mapping algorithm
  - Each task is scheduled separately and independently on other tasks
    - » A NetSolve application is seen as a sequence of independent tasks
  - Based on two performance models (PMs)
    - » The PM of heterogeneous network of computers
    - » The PM of a task

NetSolve (ctd)

- Network of computers
  - A set of interconnected heterogeneous processors
    - » Each processor is characterized by the execution time of the same serial code
      - Matrix multiplication of two 200×200 matrices
      - Obtained once on the installation of NetSolve and does not change
    - » Communication links
      - The same way as in NWS (latency + bandwidth)
      - Dynamic (periodically updated)

NetSolve (ctd)

- The performance model of a task
  - Provided by the person installing the task on a remote computer
  - A formula to calculate the execution time of the task by the solver
    - » Uses parameters of the task and the execution time of the standard computation unit (matrix multiplication)
  - The size of input and output data
  - The PM = a distributed set of performance models

NetSolve (ctd)

- The mapping algorithm
  - Performed by the agent
  - Minimizes the total execution time, T_{total}
    - » $T_{total} = T_{computation} + T_{communication}$
    - » $T_{computation}$
      - Uses the formulas of the PM of the task
    - » $T_{communication} = T_{input\ delivery} + T_{output\ receive}$
      - Uses characteristics of the communication link and the size of input and output data

Performance issues: advanced topics

Processors Heterogeneity (ctd)

- A factor preventing from accurate estimation of the relative speed
  - Different levels of integration into the network
    - » The heavier the integration, the less predictable the performance characteristics
      - Constant unpredictable fluctuations in the workload
    - » A problem for general-purpose local and global networks
      - Dedicated clusters are much more predictable

Modelling processor speed

- Traditional model
  - The relative speed is represented by a positive constant
  - Important: application specific
    - » Different relative speeds for different applications
  - Building and maintaining the model
    - » Run a representative piece of code at run time (mpC approach)
      - Efficient and effective
      - The responsibility of the application programmer
- Heterogeneous parallel algorithms are mainly based on the traditional model

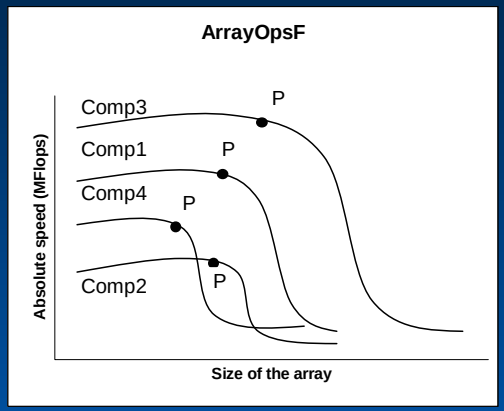
Modelling processor speed (ctd)

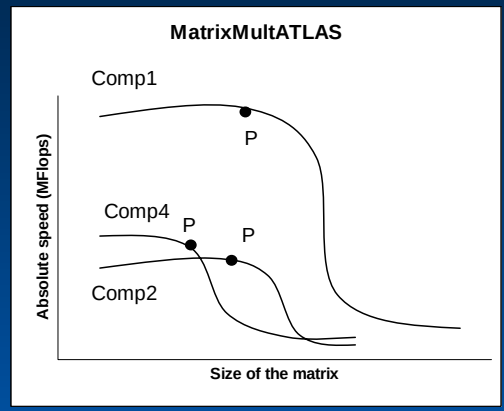
- Limitations of the traditional model
  - Do not take account of memory heterogeneity
    - » Different sizes of the main memory, different paging algorithms
      - The relative speed may not be constant independent on the problem size
    - » Possible approach: avoid paging
      - May not make sense or be impossible
  - Suitable mainly for
    - » Carefully designed application efficiently using memory hierarchy and solving problem of relatively small size

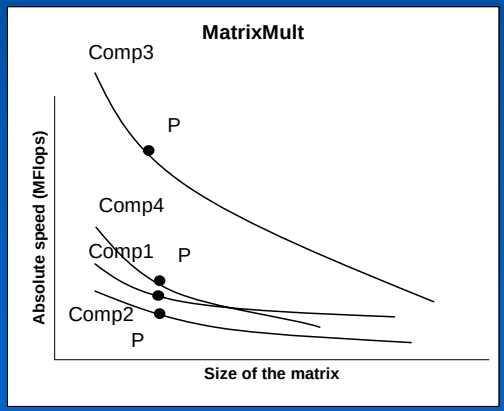
Modelling processor speed (ctd)

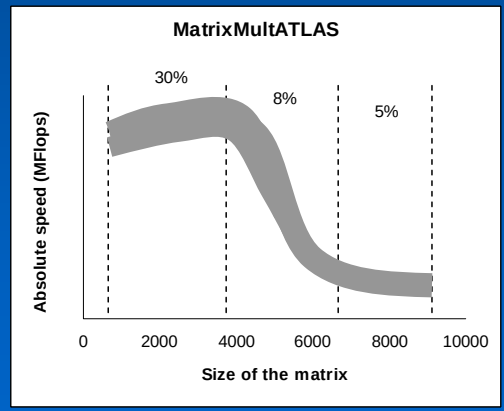
- More realistic approach
  - Represent the processor speed by a function of problem size
  - Question 1:
    - » What are the shape and properties of the speed function?

Some typical shapes of the speed function









Functional performance model

- The speed function
  - Continuous function (not step-wise)
    - » Many ordinary applications cannot be accurately approximated by a step-wise function
    - » Integration into the network
      - Constant unpredictable fluctuations in the workload
      - Speed bands rather than speed curves
      - => Smooth rather than sharp speed functions
  - Continuous function with a limit on the problem size (to model differently configured computers)

Functional performance model (ctd)

- Problems
  - Building and maintaining the model
  - Partitioning and scheduling with the model
- Building the functional model
  - Open problem
  - Some promising preliminary results
    - » Application specific
    - » Piece-wise linear approximation
      - Optimization of the number of points
        - Based on the width of the fluctuation band

Functional performance model (ctd)

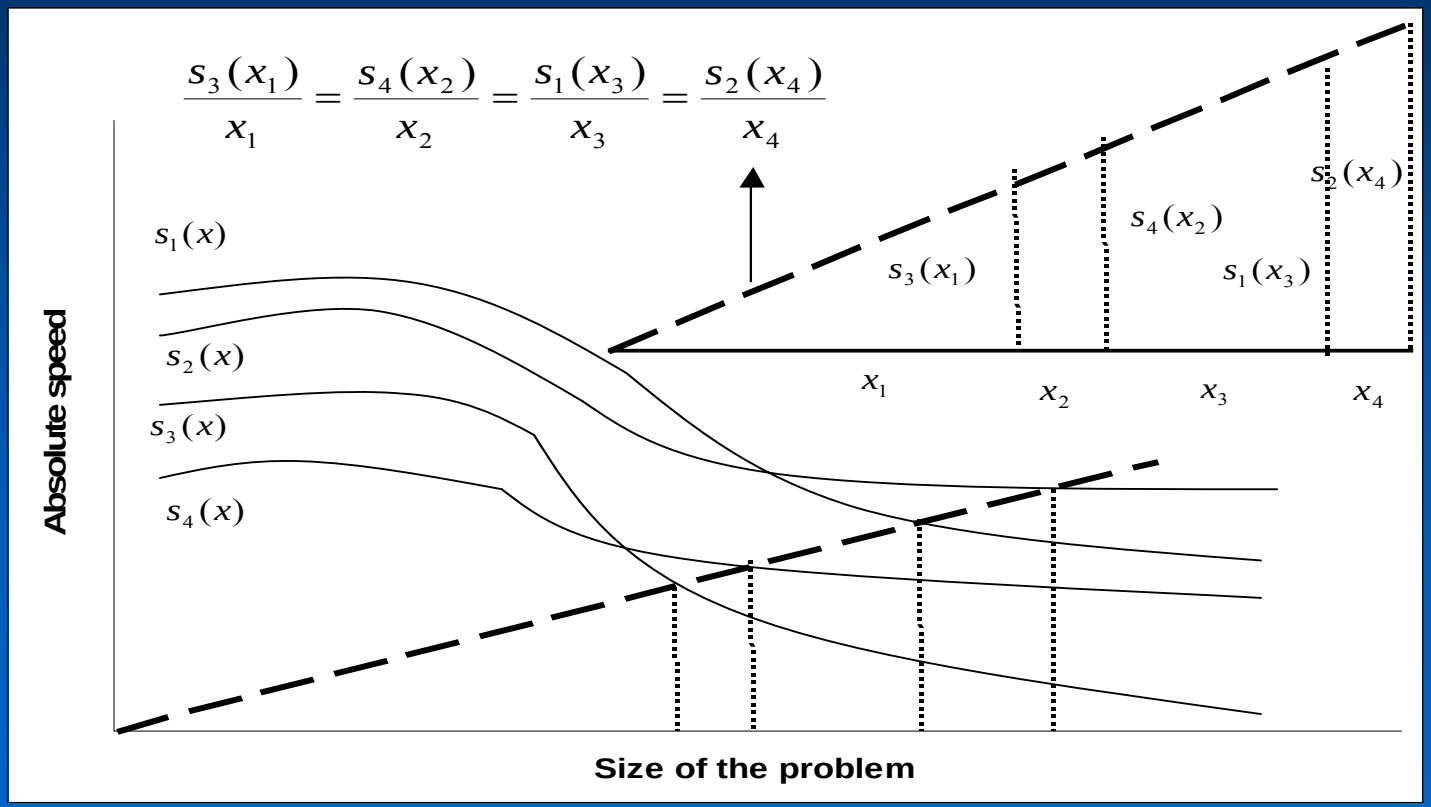
- Maintaining the functional model
  - Preliminary but promising results
    - » Correction based on the current workload
- Partitioning and scheduling with the model
  - Functional model in the form of curve
    - » Some efficient algorithms have been designed
  - Functional model in the form of band
    - » Ongoing research

Partitioning Sets

- We consider the following problem
  - Given
    - » A set of n elements
    - » A well-ordered set of p processors whose speeds are functions of the size of the problem, $s_i=f_i(x)$, such that
      - Each function is continuous
      - There is only one intersection point of the graph of the function with any straight line passing through the origin.
  - Partition the set into p disjoint partitions such that
    - » The number of elements in each partition is proportional to the speed of the processor owning that partition

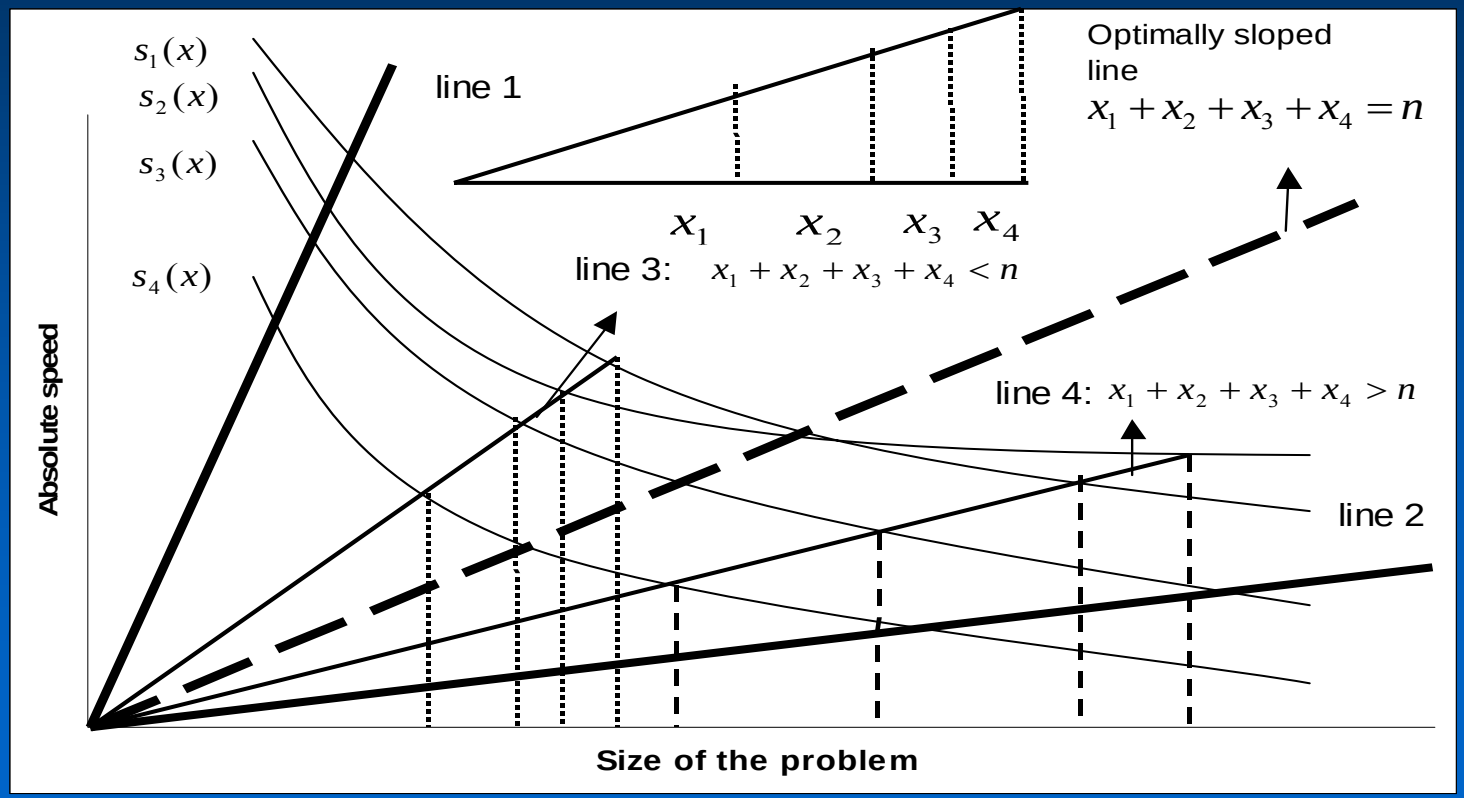
Partitioning Sets (ctd)

- Our algorithms are based on the observation:



Partitioning Sets (ctd)

- A basic algorithm

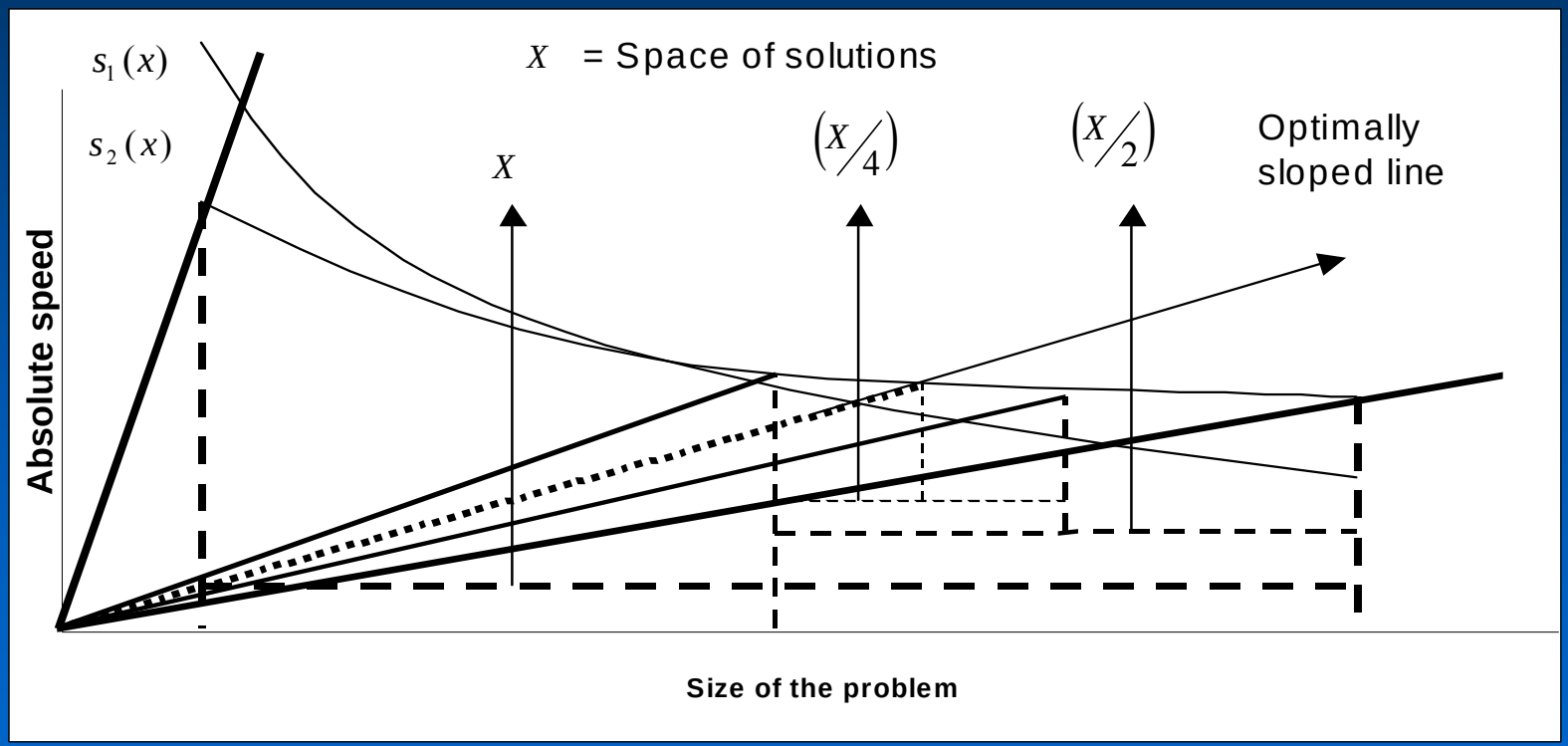


Partitioning Sets (ctd)

- Complexity of the basic algorithm
  - Each step is of complexity $O(p)$
  - If $\theta_{opt}(n) \sim n^{-k}$, where $k=const$
    - » The maximal number of steps to arrive at the optimal line will be $\sim k \times \log_2 n$ => the complexity will be $O(p \times \log_2 n)$
  - If $\theta_{opt}(n) \sim e^{-n}$
    - » The number of steps will be $\sim n$ => the complexity will be $O(p \times n)$
- The efficiency of this algorithm depends on the shape of the graphs

Partitioning Sets (ctd)

- A modified algorithm

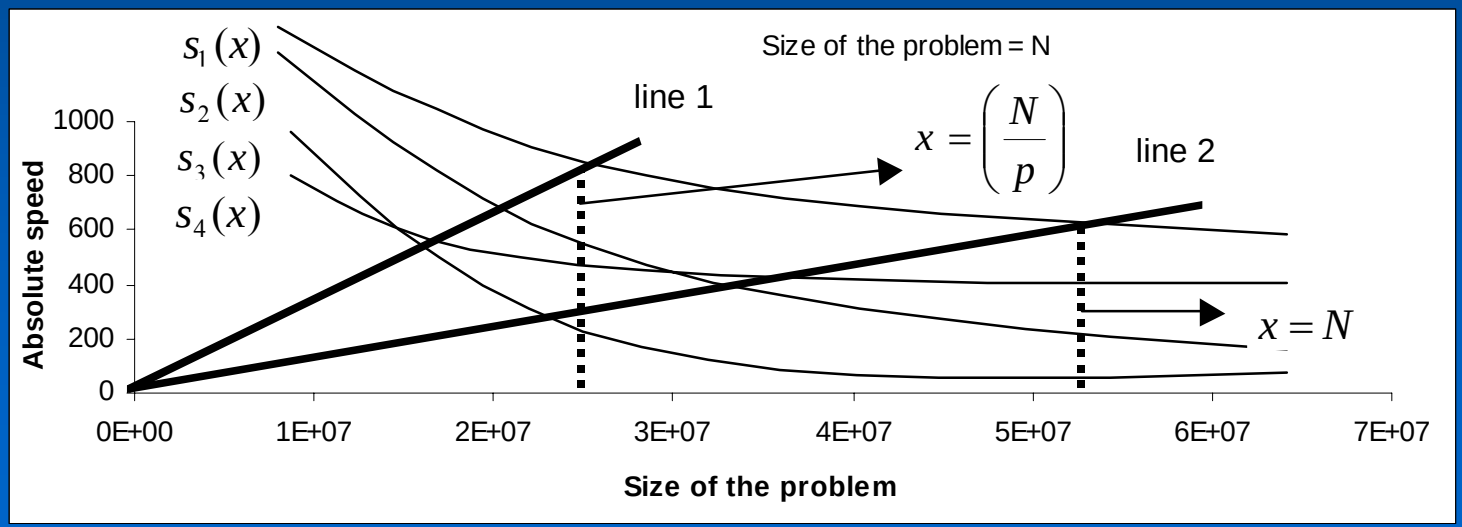


Partitioning Sets (ctd)

- At each step of this algorithm
  - Detect the graph with maximal number of solutions
  - Bisect this region of solutions
- Complexity of the modified algorithm
  - Does not depend on the shape of the graphs
  - After p such bisections the total number of solutions in the region is reduced at least by 50%
    - » This means we need no more than $p \times \log_2 n$ steps to arrive at the sought line => the complexity will be $O(p^2 \times \log_2 n)$

Partitioning Sets (ctd)

- Synthetic algorithms are possible
  - Whose worst-case complexity is $O(p^2 \times \log_2 n)$ but the best-case complexity is $O(p \times \log_2 n)$
- Detection of the initial two lines:



References

- Heterogeneous Computing Workshop (HCW)
- Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks (HeteroPar)
- A.Lastovetsky. *Parallel Computing on Heterogeneous Networks*. Wiley, June 2003.
- J.Dongarra and A.Lastovetsky. A Survey of Heterogeneous High Performance and Grid Computing. In *Engineering the Grid: Status and Perspective*. Eds. B.DiMartino, J.Dongarra et al. American Scientific Publishers, January 2006 (also at www.netlib.org/people/JackDongarra/PAPERS/hetero-dist-survey-2004.pdf).