DYNAMIC LOAD BALANCING IN HETEROGENEOUS DISTRIBUTED ENVIRONMENTS

By

Tiberiu Rotaru

DISSERTATION

Submitted to the Faculty of Sciences in accordance
with the requirements for the degree of
"Docteur ès Sciences"

UNIVERSITY OF NEUCHÂTEL
INSTITUTE OF COMPUTER SCIENCE
RUE EMILE-ARGAND 11
CH-2007 NEUCHÂTEL
SWITZERLAND

Supervisor:

Prof. Hans-Heinrich Nägeli, Université de Neuchâtel

Examiners:

Prof. Denis Trystram, Institut National Polytechnique de Grenoble

Prof. André Schiper, Ecole Polytechnique Fédérale de Lausanne

Prof. Kilian Stoffel, Université de Neuchâtel

IMPRIMATUR POUR LA THESE

Dynamic Load Balancing in Heterogeneous Distributed Environments

de M. Tiberiu ROTARU

UNIVERSITE DE NEUCHATEL

FACULTE DES SCIENCES

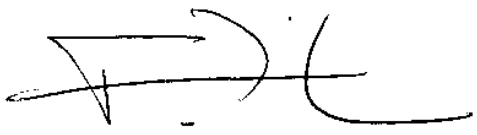
La Faculté des sciences de l'Université de Neuchâtel, sur le rapport des membres du jury

MM. H.-H. Nägeli (directeur de thèse),
K. Stoffel, D. Trystram (Grenoble)
et A. Schiper (Lausanne)

autorise l'impression de la présente thèse.

Neuchâtel, le 5 juin 2003

Le doyen:



François Zwahlen

To my children Andrei and Mara.

Table of Contents

List of Tables

List of Figures

Abstract

Parallel and distributed architectures are emerging as commonplace platforms for high performance computing. The existence of standard parallel programming libraries facilitates the use of such platforms for running challenging applications in science or engineering. However, as compared to the homogeneous case, a number of additional factors should be taken into account in order to ensure an efficient execution on heterogeneous computing environments. In such systems one must consider not only the application's dynamic behavior but also the environment's dynamics. We dealt with the dynamic load balancing for parallel adaptive simulations in heterogeneous computing environments. Such applications are usually characterized by the fact that the workloads assigned to processors may change significantly and unpredictably.

Our work followed two main directions. In a first step we developed a theoretical heterogeneous computational model and we extended and analyzed the well-known diffusion algorithms with respect to such a model. Afterwards, our variant of generalized diffusion was further compared with other approaches and was shown to be faster than another popular technique, the hydrodynamic algorithm. The possibility of using polynomial acceleration schemes in a dynamic context was further investigated. Other faster dynamic load balancing schemes were proposed and investigated. An incremental algorithm that is capable of taking advantage of an already computed fair distribution for computing a new one was described. In a second step, we designed and implemented a testbed for adaptive numerical simulations in heterogeneous computing environments. A tool, called HeRMeS, which allows to perform such simulations and that incorporates the methods developed at the first step, was designed and implemented. Experiments with the discussed methods and with HeRMeS were performed.

Acknowledgements

I thank to Prof. Hans-Heinrich Nägeli for his guidance and support. The numerous discussions that we had during regular meetings helped me to crystallize my ideas and to improve my work.

I thank to the members of the jury Prof. Denis Trystram, Prof. André Schiper and Prof. Kilian Stoffel for their valuable comments and remarks.

I thank to my colleague Etienne Richard for the help with the installation and the utilization of the NWS tool.

I thank to my children Andrei and Mara and to my wife Andrada for their unconditional support, love and understanding.

Chapter 1

Problem overview

Numerous applications in science or engineering, in domains like fluid dynamics, global climate modeling, formation of the universe, aerodynamics, ocean circulation are computationally intensive, requiring huge memory and processor resources. The distributed network computing environments can offer sufficient resources on which to execute such applications, as in many companies or institutions they are unused most of the time. Such systems represent an attractive alternative to the multi-processor computers because of their characteristics: better performance to price ratio, extensibility, accessibility.

However, programming in distributed environments constitutes a challenging task and poses a number of additional problems. Heterogeneous computing systems present a new challenge to dynamic load-balancing software [3]. Multiple users usually share these platforms and the performance of the applications may be affected by a number of factors that are subject to dynamic and unpredictable variations. One major question is how to efficiently exploit these systems. An important task of a developer in the context of heterogeneous environments is to ensure a fair use of all the available resources. Within such contexts, the application developers must account for widely varying processor powers, memory capacities, and network connections. Besides the dynamic variation of the workloads assigned to processors, generated by the application, one should also consider the possible variation of the physical parameters of the host environment.

The problem of dynamic load balancing has been extensively studied in the last decade mainly

in homogeneous systems [13, 6, 7, 43, 16, 33, 57, 13, 93, 34, 91]. Significant progress has been achieved in the context of numerical applications based on adaptive unstructured meshes; complex tools incorporating a number of mature techniques are now available [36, 35, 39, 42, 37].

Moving parallel applications to dynamic and heterogeneous computing environments introduces a new level of complexity. These environments require the selection and the configuration of program parameters at runtime, depending upon the state and the availability of resources. However, the selection of a best mix between system resources, mappings, load distributions and communication mechanisms is non-trivial in a heterogeneous environment. The system's dynamics and the application's adaptivity make the dynamic load balancing problem to be a significant challenge. The straightforward adaptation of the techniques employed in the homogeneous case for the use in the heterogeneous case is often not an easy task. In distributed environments, in addition to the parameters that are dependent on the application, the processor parameters as well as the network parameters must be taken into account in order to determine an optimal load distribution that leads to an improved execution time of the entire application. Specifically, physical parameters of both the interconnection network and the processors, such as the latency, bandwidth, processor capacity, etc..., which are often ignored in the homogeneous systems where the assumption of uniformity is sound, must be taken into account. As an example, the tests performed by Ripeanu et al. [65] with the Cactus platform led the authors to the conclusion that *"one should definitely consider network performance issues when taking load-balancing decisions"*.

In this thesis we developed or extended dynamic load balancing techniques that are applicable in heterogeneous systems [66, 67, 69, 68]. Our work has been carried out with respect to a theoretical heterogeneous model that assumes that the processors may have different capacities and that the communication may have different costs. The techniques we sketch are sufficiently general to be applied for a broad class of parallel and distributed applications. In the second part, the simulation of an application that performs adaptive numerical computations, relying on unstructured meshes, is described. We designed HeRMeS, a tool that can serve primarily as a testbed for our dynamic data redistribution methods in heterogeneous environments and secondly, as an extensible platform for performing numerical adaptive simulations on a heterogeneous collection of distributed memory machines. The design philosophy and implementation of such a testbed is described. Results of

experimental tests performed on a computational grid that we built using commodity hardware available in our institute are reported.

Chapter 2

Introduction

A main concern when running large applications on distributed environments is the efficient use of the available computational resources. The execution time of the application is influenced by the speed of the slowest processor, therefore the workloads should be fairly distributed over processors. From time to time, it may be necessary to proceed to a redistribution, as at runtime the workloads assigned to processors as well as the available capacities of the processors and the network performance may vary, often in an unpredictable manner.

The networks of workstations constitute cost-effective platforms for parallel computing. The progresses made in the direction of improving the speed of computers and of interconnection networks make such environments suitable for parallel and distributed computing. Many academic institutions and companies dispose now of a great unexploited potential in terms of processing power that could be made effective by putting together the available computing resources. Furthermore, the existence of standard or widely used programming libraries, such as MPI or PVM, allows for programming such hybrid systems independently of their hardware characteristics. However, although heterogeneous environments offer numerous advantages, it is difficult to obtain the maximum benefit of their theoretical computing power.

A parallel application usually consists in a set of communicating components that have to be allocated onto the physical resources of a target architecture [23]. The allocation can be static or dynamic. Static allocation occurs before execution and can take into account both the execution and

the communication needs. This is feasible when the computation time of the components assigned to processors can be estimated a priori. However, for an important class of applications the workloads assigned to processors may change during the computations and cannot be estimated in advance. These applications require dynamic allocation. The application's specific dynamic needs must be considered at runtime, on the basis of the system's current state. One should take into account the availability of the system's physical resources when trying to assure a fair workload distribution in the system.

One of the most important capabilities of a dynamic load balancing algorithm should be the ability to act incrementally, i.e. the ability to take advantage of a previously computed fair data distribution when computing a new one, in order to minimize the overhead. An important restriction imposed by some applications is that of preserving the data locality. A popular class of methods that meet these requirements to a large extent is represented by the nearest-neighbor algorithms. They lead an unbalanced system to a global equilibrium state by allowing information exchanges only between processors that are neighbors in the communication graph induced by the application. The most popular variants are the diffusion [13, 6, 43, 16, 33], the dimension exchange [57, 13, 93], the multi-level diffusion [34], and the gradient model [91]. Such techniques are applicable in a larger context of parallel processing that includes the adaptive numerical mesh-based simulations.

2.1 Load balancing for parallel adaptive computations

Parallel adaptive finite element applications arising in domains like Computational Fluid Dynamics or Computational Mechanics have been considered relevant for illustrating the complexity of the problem of dynamic load balancing [30, 44], primarily in homogeneous systems. Such applications often rely on adaptive unstructured meshes. In practice, these meshes may contain a large number of finite elements and due to memory limitations and/or computational requirements they must be fairly distributed over processors, in a way that preserves the data locality. These meshes may undergo in the course of a computation significant structural changes that cause the modification of the workloads assigned to processors (refinement/de-refinement operations). A fast method is often necessary to apply in order to restore the load-balance. Because the inter-processor communication is generally costly, a fair workload redistribution algorithm must minimize the cost of data migration

and the number of dependencies between vertices residing on different processors. A centralized approach possibly leads to heavy communication and generally is not scalable.

2.2 Static load balancing strategies

2.2.1 The case of homogeneous systems

Various heuristics for static load balancing have been proposed. Farhat proposed an automatic mesh decomposer based on the Greedy Algorithm [20]. A widely used technique was the Recursive Coordination Bisection, which was originally described by Berger and Bokhari [4]. This algorithm recursively bisects the mesh at each step by dividing a subdomain in two equal parts along the coordinate axis of the longest extent. A similar method, Recursive Inertial Bisection, was proposed by Nour-Omid et al. [63]. This technique differs from the above one by that the vertices are partitioned into subsets of equal size by a plane that is perpendicular to the unique non-degenerate eigenvalue of the Inertial Matrix. A technique applicable when the mesh coordinates are not available is Recursive Graph Bisection. This was first described by George et al. [25] and, as opposed to the above methods, it uses the graph distance rather than the Euclidian distance. A better technique, regarding the quality of the generated partitions, the Recursive Spectral Bisection, was described by Simon [75]. The basic idea used with this technique was to sort at each step the vertices using as keys the values of their associated components in the eigenvector that corresponds to the second smallest eigenvalue of the Laplacian matrix of the graph. Trying to improve the performance of the partitioning techniques enumerated above, Barnard et al. [1] and Karypis et al. [53] proposed the use of multilevel schemes.

For improving the quality of the partitions, Kernighan and Lin [55] proposed an algorithm for local optimization that attempts to swap two equal-sized subsets from the initial partition with the objective to reduce the number of connecting edges at each iteration.

In the case of the unstructured meshes, the problem of static partitioning in homogeneous systems, despite all the imperfections of the underlying model addressed by Hendrickson et al. [31], cannot be further considered a challenge. Fast and high quality parallel mesh partitioning codes implementing various well studied techniques have been made available [44].

2.2.2 The case of heterogeneous systems

Less work has been done with mesh partitioning in heterogeneous systems compared to homogeneous systems. Chen and Taylor considered a hierarchical heterogeneous model and conducted experiments in distributed systems with mesh partitioning [9, 8]. They used the term of *group* to designate a collection of processors that have comparable performance and that are connected via a local network. They proposed an algorithm that acts in three steps:

- A coarse partitioning into subdomains is generated and to each group is assigned a subdomain whose size is proportional to the number of constituent processors. The performance of the processors and the computational complexity of the application are taken into account.
- The subdomain that is assigned to a group is partitioned among its constituent processors. A parallel simulated annealing technique is used to balance the execution time by taking into account the variance of network performance. The processors responsible for inter-group communication are given less computational load to compensate the cost of remote communication.
- A global optimization step is performed taking into consideration the performance of the local interconnection network of each group. Elements on the boundaries between group partitions are moved according to the execution time variance between neighboring groups.

The same authors implemented a tool for static partitioning in heterogeneous systems called PART (ParaPART, its parallel version) that was used for conducting experiments in distributed systems.

The problem of mesh partitioning in heterogeneous environments has been also tackled by Walshaw and Cross [84]. The authors extended their work related to mesh partitioning, carried out in homogeneous systems and adapted the multilevel schemes for the heterogeneous case.

2.3 Dynamic load balancing strategies

2.3.1 Dynamic load balancing in homogeneous systems

Dynamic repartitioning raises more complex problems in practice. The most important is that these methods must be performed online, in parallel. Secondly, they must be fast and their result at least compensate the cost of their application. Thirdly, care must be taken in designing adequate data structures, as well as manipulation/update primitives, because, otherwise, the cost of their bookkeeping could be high. In order to meet these requirements, the cost of data migration must be kept at a minimum. The volume of data migration is usually considered to be a rough measure of the redistribution time (whereas the edge-weight is a rough measure of the communication time of the application itself).

The general problem of dynamic load balancing for data parallel applications has been an extensively studied research topic in the last decade [93]. In the particular context of adaptive unstructured meshes, two types of methods proved to be efficient [44].

1. **Dynamic load balancing by repartitioning**

   Any static strategy for graph partitioning falls in this category. However, such methods are not generally able to explicitly minimize the data movement resulting from the repartitioning, but only the the number of edges that have the extremities in different subdomains (edge-cut). Regarding the modalities of how to take into account also the vertex migration, several ways were suggested. A key idea was to associate a virtual vertex with each subdomain and to connect it with the other vertices in the subdomain by virtual edges [81, 32]. Thus, by partitioning the modified graph, both of the objectives of minimizing the edge-cut and the data movement were considered at the same time. Schloegel et al. proposed a strategy that constructs coarser graphs by merging pairs of neighboring vertices into a single vertex, in each subgraph assigned to a processor [73]. This strategy guarantees that in the uncoarsening phase, the vertices that participated to form a vertex in a coarser graph in the old partitioning are located in the same subdomain in the new partitioning. Another suggested approach was to use after repartitioning a re-mapping algorithm to allocate a new subdomain to a processor on which the most of its vertices reside, in order to reduce the data movement [64, 74].

2. **Dynamic load balancing through node migration**

   These methods generate a new partition by migrating vertices between adjacent subdomains. Usually, they act in two phases [44]:

   - Flow calculation: each processor computes the number of vertices to be sent/received to/from neighboring subdomains residing on different processors. All these quantities define the components of a *balancing flow* on the edges of the subdomain graph.
   - Vertex selection: each subdomain/processor determines which vertices should be migrated to adjacent subdomaims/processors so as to satisfy the balancing flow.

For the flow calculation one of the following methods may be used:

1. **Diffusion**

   This technique consists in simulating a heat diffusion process between the subdomains: their *heat*, representing their weight (i.e. the sum of the weights of the constituent elements), flows from *warmer* to *colder* subdomains. It balances load distributions by working only on local states and by locally moving elementary items in the direction suggested by an energy-minimization goal. Usually, every subdomain exchanges in each iteration step a fraction of the difference between its weight and the weights of the adjacent subdomains; these steps are iterated until a global equilibrium is reached. At each step, the exchange is done with all neighbors at a time. The original diffusion, as it was described by Boillat [6] and Cybenko [13], converges slowly on subdomain graphs with small connectivity. Efforts to improve it were done by different authors [62, 43, 16]. Other variants of this method were proposed Heirich et al. [33] and Watts et. al. [89].

2. **Dimension exchange**

   The underlying idea is similar, i.e. exchanging a fraction of the difference between a subdomain's weight and the weight of its neighbors, but, contrary to diffusion, the exchange is done with only one neighbor at a time: only distinct pairs of neighbors are allowed to exchange information in one step. A way that was suggested to handle this was to color the edges of the subdomain graph and, at each step, to allow exchanges only between subdomains that

are connected by edges of the same color [57]. Originally, the exchange fraction was set to 1/2 [13]. Because the convergence on subdomain graphs with small connectivity is low, too, other values were suggested for the exchange parameter in order to accelerate the convergence. Xu and Lau [93] proposed a variant of dimension exchange that consists in exchanging, each time, a fraction different from 1/2 between distinct pairs. They called this method Generalized Dimension Exchange (GDE) and they gave optimal values of the exchange parameter for different types of graphs. Unfortunately the authors ignored the quality of the generated balancing flow.

3. **The method of potentials**

   Hu et al. proposed [45] a direct method that finds the minimum balancing flow w.r.t. a homogeneous computing model. This method consists in finding the solution of a system, using a conjugate gradient method. This system involves the Laplacian matrix of the subdomain graph and the imbalance vector (whose components are the differences between the weight of a subdomain and the average weight). This method uses a more global view but is faster than the diffusion algorithms. The authors referred it to as *the method of potentials* [44].

   As Walshaw noticed [85], it can be viewed as a diffusion whose parameters are determined iteratively at each step, as opposed to the original diffusion that uses fixed parameters. Schloegel and Karypis [71] used the term *directed diffusion* when referring to this method and *undirected diffusion* for the original diffusion described by Cybenko [13].

4. **Multilevel diffusion method**

   The method consists in recursively bisecting the subdomain graph. At each step the algorithm keeps track of the number of elements that must be migrated from one subdomain to another so that they become balanced. The number of bisection steps is logarithmic in the number of subdomains, but each step is complex and requires much interprocessor communication. This method was described by Horton [34]. Unfortunately, as in the case of the dimension exchange, the quality of the generated balancing flow was ignored.

2.3.2 Techniques for vertex selection for migration

After finding out *how many* vertices a processor has to exchange with its neighbors to restore the balance, one must decide *which* vertices should be migrated so as to satisfy the migration flow with maximum profit. Clearly, the border vertices are the first candidates to be examined. Usually the selection process is guided by using a gain function. Several ideas emerged:

- A natural idea used by Vidwans et al. [83] was to transfer successive layers of boundary vertices between sender and receiver processors.
- Walshaw et al. [86] used heuristics for vertex selection based on the idea of gain. The gain is generally related to a cost function that express the quality of a partition, like for example the sum of the weights of the edges with the extremities residing on different subdomains. The vertices on the border of a host subdomain having neighbors on a specified target subdomain were sorted using a criteria based on the idea of relative gain and then migrated in a decreasing order. The number of vertices to be transferred was calculated taking into account the balancing flow and the total weight of vertices with positive gain on the border that *preferred* the target subdomain.
- Schloegel et al. [72] used multilevel schemes. The boundary vertices in the coarsest graph were visited in a random order and a quantity of vertices were migrated so as to satisfy the computed balancing flow. After the load was balanced, multilevel refinement started and the graph was refined again. Boundary vertices were visited randomly and migrated according to a certain vertex migration criterion.
- In the case of the graphs that are highly imbalanced in a certain area, an algorithm called *wavefront diffusion* was proposed [73]. The migration was started in a wavefront form from the subdomains that were the most overloaded.

2.3.3 Heterogeneous dynamic load balancing

The use of heterogeneous distributed systems for high performance computing calls for adequate efficient dynamic load balancing strategies. Unfortunately, although dynamic load balancing has

been intensively studied, most of the proposed schemes are inadequate for distributed environments. Some of them assume that all the processors have similar capabilities and that the interconnection network is dedicated and has uniform performance. Some efforts to adapt the already existing schemes for the use in heterogeneous environments have been done [18]. Unfortunately, most of these assume a fixed underlying communication topology. Watts et al. [89] and Hui et al. [46] proposed algorithms for heterogeneous environments but they ignored the network performance. Experiments with dynamic repartitioning in heterogeneous systems for the case of adaptive structured meshes were reported by Sinha et al. [78]. Multilevel algorithms have been successfully used for solving the graph partitioning problem in homogeneous environments, being incorporated in most of the partitioning tools (METIS, Jostle, Chaco). However, they fail to address the limitations imposed by heterogeneous computational models and supplementary work has to be done to adapt them for the use in such environments.

2.3.4 Packages/Tools for dynamic load balancing

The research efforts pursued in the last years in the context of dynamic load balancing for adaptive unstructured meshes were fruitful and concretized into several packages for dynamic repartitioning. The most popular were ParMETIS [36], PJostle [39], Party [42]. Meanwhile, as it was remarked by Hendrickson and al. [30], the relatively large number of methods proposed for dynamic repartitioning did not necessarily result in widely used software. The idea of a general-purpose dynamic load balancing library comprising the most efficient dynamic repartitioning techniques emerged. Among the reasons for implementing such software one can enumerate:

- Contrary to the case of static partitioning, the adaptive case requires the use of *online* techniques; the procedures used for adaptive repartitioning must therefore be used at runtime.
- As different adaptive applications have different characteristics and as no method is adequate in all cases, the programmer of a parallel numerical solver (PNS) needs a palette of different algorithms that allow for a trade-off between the execution time of the repartitioning procedure and the communication time of the PNS.
- Many industrial applications need complex solvers and need to be run on parallel systems

in order to make them efficient; thus, adaptive repartitioning libraries constitute a stringent necessity. A general-purpose library implementing several dynamic load balancing algorithms would permit the choice of the most appropriate method for a specific kind of application.

- For the researchers, the simulations and the tests with real applications would permit to form a more realistic and critical point of view regarding the imperfections of the model inherited from graph partitioning. This would allow to improve the existing techniques or to propose new methods for dynamic repartitioning.

So far, two such general libraries have been implemented and made publicly available: Zoltan [38], developed at Sandia National Laboratories, and DRAMA [37], developed at Leuven University in collaboration with an international consortium. Zoltan provides an object-oriented interface and includes a series of load-balancing algorithms. Interfaces to ParMETIS and Jostle packages are provided as well as data migration tools for data movement during re-balancing. DRAMA offers a mesh based interface and the possibility of using the methods implemented by ParMETIS and PJostle.

Unfortunately, the existing software provides only limited support for the use on heterogeneous environments; generally, they are not able to take into account the variation of the processor or the network performance. No widely used tool for dynamic repartitioning in heterogeneous systems exists. Nevertheless, as it is widely accepted that the future of high performance computing is the distributed computing, efforts in this direction exists [38].

2.3.5 Load balancing on Grid

The Grid environments are more and more paid attention by the researchers from all areas of computer science. The Grid designate an infrastructure that allows an integrated and collaborative access to resources owned and managed by multiple organizations. These environments are characterized by a high degree of heterogeneity and no assumptions can be done regarding the resources that are involved.

Running applications on Grid requires the existence of easy-to-use tools for managing it in such a way that it may be used in a reliable and efficient way by a wide variety of users. Globus [24] and Legion [27] are among the best known tools of this kind. Legion is tool that sits on top of the

user's operating system and allows to see other resources in the Grid. It applies its own security and scheduling policies and it supports interoperability between objects written in multiple languages. The Globus project is a research and development project focused on enabling the application of Grid concepts to scientific and engineering computing. It deals with resource management, data management and access, application development environments, information services, and security.

Regarding the Grid, some work has been recently done relative to load balancing for mesh-based applications. Kumar et al. [56] report experiments with graph partitioning on this type of environments. The authors considered a heterogeneous model in which is assumed that the processors have varying processing power and that the communication is non-uniform in the underlying network. They argued that with the emergence of technologies such as the Grid, it is imperative to study the partitioning problem taking into consideration the differing capabilities of such distributed heterogeneous systems.

Chapter 3

Contributions

Our work w.r.t. dynamic load balancing in heterogeneous environments followed two main directions:

- In a first step, we investigated a number of dynamic load balancing methods w.r.t. the assumed theoretical heterogeneous computing model;
- In a second step, we designed and implemented a testbed for adaptive numerical simulations in heterogeneous computing environments, with incorporated dynamic load balancing support, including the methods developed in the first step.

A basic theoretical heterogeneous computational model was proposed. It was assumed that the processors may have different and varying capacities and that the network parameters are non-uniform and may vary at runtime. Several methods for fair dynamic load redistribution were described. They were first tested with synthetic topologies, loads, processing capacities and network parameters. Afterwards, these methods were incorporated in a software module and tested with a simulated adaptive numerical application on a heterogeneous environment that we built using commodity hardware available in our institute. The theoretical model that we assume here relies on a weighted graph in which the links correspond to logical communication patterns (induced by the application), the vertex weights correspond to processor speeds and the edge weights correspond to communication costs. The main results obtained in the area of dynamic load balancing w.r.t. the assumed model can be summarized as follows:

1. In a first step, we tried to adapt the diffusion algorithm, in the form it was proposed by Boillat [6], for the fair redistribution of workloads in a heterogeneous environment (proportionally to the available capacities of the processors) [66, 67]. Other approaches proposed in the literature generalize the variant of diffusion described by Cybenko [13] [19]. We investigated the convergence of this type of algorithms (generalized diffusion) and we put in evidence some of the properties that recommend them as potential candidates to be used in distributed environments: the load imbalance measured in a weighted norm like $\|\cdot\|_{2,D^{-1/2}}$, $\|\cdot\|_{\infty,D^{-1}}$ or $\|\cdot\|_1$ (D being the diagonal matrix of capacities) decreases or remains the same after each iteration step; globally the algorithm results in a geometrical reduction of the total load imbalance. Furthermore, it requires only local communication and needs synchronization between neighbors only.

2. It was put in evidence that as in the homogeneous case, the convergence rate of the generalized diffusion is tightly connected to the second largest eigenvalue of the corresponding diffusion matrix [66, 67]. This is based on the generalized *Laplacian matrix* of the communication graph induced by the application and defines a discrete reversible Markov chain. As for the mixing time of Markov chains, bounds can be formulated for the convergence factor of generalized diffusion algorithms, using Cheeger type inequalities. These inequalities are related to the second largest eigenvalue of the underlying stochastic matrix and to the notion of conductance [76, 77]. In the case of a generalized diffusion algorithm the conductance coincides with the isoperimetric constant and the Cheeger constant relative to the generalized Laplacian [11, 66, 67]. Although these constants generally give tight bounds for the second largest eigenvalue of the generalized diffusion matrix, it is NP-hard to compute them. We tried to found alternative bounds for the convergence factor of a generalized diffusion. We proved the following results:

   - Adapting a result given by Mohar [61], we improved the lower bound in the Cheeger inequality related to the second largest eigenvalue of the generalized Laplacian as follows [66]:

     $$\mu_2(\mathcal{L}) \geq 1 - \sqrt{1 - i_M^2(G)},$$

     where the $i_M(G)$ is the isoperimetric constant and $\mu_2(\mathcal{L})$ is the second largest eigenvalue

of the generalized Laplacian matrix.

- Following a different way, we gave a tight upper bound for the convergence factor of the generalized diffusion by combining results from the graph theory, linear algebra and the theory of nonnegative matrices. We extended a result of Fiedler [21] relative to doubly stochastic matrices. We proved that in a heterogeneous environment (G, l, c, w) the second smallest eigenvalue of a generalized diffusion matrix M, $\lambda_2(M)$, satisfies the following inequality:

$$\lambda_2(M) \leq 1 - \frac{4}{c_{\max}} \left(\min_{\emptyset \neq S \subset V} \sum_{i \in S, j \notin S} m_{ij} c_j \right) \sin^2 \left(\frac{\pi}{2p} \right), \tag{3.0.1}$$

  where p is the number of processors, c_i denotes the capacity of the processor i, $c_{\max}$ the maximal capacity of a processor and m_{ij}'s are diffusion parameters.

- Another bound for the second largest eigenvalue was found:

$$\lambda_2(M) \leq 1 - \frac{\theta(M)}{\operatorname{diam}(G)}, \tag{3.0.2}$$

  where $\theta(M) = \min_{ij \in E(G)} m_{ij} c_j$ and $\operatorname{diam}(G)$ is the diameter of the communication graph G induced by the application.

3. We suggested a better choice for the generalized diffusion matrix than that commonly used in the homogeneous case. We considered a more general case, when $M = I - ASWA^T D^{-1}$, where s is a vector of scalars of size $|E(G)|$ and $S = \operatorname{diag}(s)$ is a diagonal matrix, w is a vector of edge-weights and $W = \operatorname{diag}(w)$, c is the vector of capacities of processors and $D = \operatorname{diag}(c)$ and A is the vertex-edge incidence matrix of the communication graph. The variant proposed by Elsässer et al. [18] corresponds to the particular case when $S = \alpha I$, with $\alpha \in \mathbb{R}$. We investigated the case when, for an arbitrary $\epsilon \geq 0$ and for any edge e_k, the parameters s_k are defined as follows:

$$s_k(\epsilon) = \min_{e_k = \{i,j\}} \left\{ \frac{c_i}{\delta_i^w + \epsilon} \frac{c_j}{\delta_j^w + \epsilon} \right\}. \tag{3.0.3}$$

   By considering $M(\epsilon) = I - AS(\epsilon)WA^T D^{-1}$ and

$$\epsilon_0 = 2e(G) w_{\min} \frac{c_{\min}}{c_{\max}} \sin^2 \left(\frac{\pi}{2p} \right),$$

we showed that $M(\epsilon_0)$ improves the upper bound fixed by the inequality (3.0.1) for all $\epsilon \geq \epsilon_0$. In the homogeneous case, the diffusion scheme proposed by Boillat [6] corresponds to that based on $M(1)$. Experiments showed that a diffusion based on $M(\epsilon_0)$ is generally faster than one that uses $M(1)$. For common topologies, ϵ_0 is close to the optimal diffusion parameters given by Xu et al. [93].

4. Using the inequality (3.0.1), we could establish that the maximal number of steps required by the above generalized diffusion algorithm is in

$$O\left(\frac{c_{\max}\,\Delta^w\,p^2}{c_{\min}\,w_{\min}\,e(G)}\right),$$

where $e(G)$ is the edge connectivity of the graph G, $c_{\min}$ and $c_{\max}$ are the minimal and the maximal capacity of a processor, $w_{\min}$ is the minimal communication cost and Δ_w is the maximum weighted degree. This result is consistent with what happens in the homogeneous case [6] and is tight for a communication topology of type ring or path, as the maximal number of steps is in this case in the Θ class of the above function.

5. Using the inequality (3.0.2), we proved [66] that there exist generalized diffusion algorithms that theoretically perform faster than the hydro-dynamic algorithm given by Hui et al. [46].

6. The flow generated by various neighbor schemes is usually characterized and analyzed indirectly through a number of properties rather than using a quantitative characterization. It is approximated in a number of iterations by accumulating at each step fractions of load differences. We gave a direct explicit formula for the balancing flow generated by a generalized diffusion algorithm, as a function of the generalized Laplacian matrix and of the parameters defining the assumed theoretical model [69, 68]. The same flow is identical to that generated by the polynomial schemes.

7. Implicit schemes were used both in a homogeneous context and in heterogeneous context by Watts et al. [89, 88]. However, the proposed schemes generally ignore the communication costs as well as the quality of the generated balancing flow. We proposed an implicit heterogeneous diffusion scheme and we showed that it generates the same balancing flow as a generalized diffusion algorithm [69].

8. The quality of the balancing flow generated by the heterogeneous diffusion-like schemes was investigated. This flow has a particular property: it is the scaled projection of any balancing flow in a given heterogeneous computational model. As a natural consequence, it is minimal w.r.t. the weighted 2-norm $\|\cdot\|_{2,W^{-1/2}}$ in any heterogeneous model (G, l, c, w), where l denotes the vector of workloads, c denotes the vector of capacities and w the array of the edge-weights and $W = \mathrm{diag}(w)$.

9. In the homogeneous case, an efficient algorithm based on the conjugate gradient method was given by Hu et al. [45]. We extended this idea and we described and implemented an algorithm applicable in a heterogeneous environment [70].

10. We gave an alternative characterization for the minimum balancing flow as a function of a proper divisor of the characteristic polynomial of the generalized Laplacian of the communication graph. This allowed us to describe a nearest neighbor algorithm that requires a number of communication steps equal to the number of processors only [68]. The algorithm LPS requires knowledge at runtime of the coefficients of this polynomial but efficient ways to compute them in parallel were indicated [70]. It is more appropriate in a static context, when only the workloads change at runtime, but neither the communication topology nor the processing capacities change.

11. The above algorithm was further improved in order to be used in a dynamic context. The result was a fast algorithm, called LPA [70]. It takes into consideration all the parameters of the assumed heterogeneous model.

12. The key idea in developing faster algorithms for dynamic load balancing in distributed environments is to reduce the number of communication steps and to avoid global communication. A natural idea for diminishing the complexity of the above algorithm was to use the minimum generating polynomial of the generalized Laplacian matrix rather than the characteristic polynomial. We described a parallel implementation of this algorithm, called MPA.

13. An incremental algorithm, capable of taking into account a previously computed fair distribution when computing a new one, was designed and implemented. The key idea resides in the explicit expression of the minimal balancing flow [68]. The minimal balancing flow can be

viewed as the result of the application of a *balancing operator* to the current workload vector. This balancing operator can be obtained from a combination of the Laplacian of the communication graph, the vector of computational capacities and the communication weights. The key idea is that in some cases, after the modification of a parameter, this balancing operator can be computed faster if one uses its previous expression.

14. Experiments were conducted in a heterogeneous networked computing environment in order to test the our methods. A dynamic repartitioning scheme capable of using any of the methods described above was derived from ParMETIS [36]. A tool called HeRMeS, that simulates the behavior of an adaptive numerical application, was designed and implemented. Tests were performed on a computational grid that we built using commodity hardware available in our institute.

Chapter 4

Methodology

Many problems in science and engineering are modeled using partial or ordinary differential equations. Often, the solution of complex problems cannot be determined analytically, therefore they need to be computationally solved through a discrete time simulation. This involves the use of a grid whose node values are updated iteratively for a fixed number of steps or until a convergence condition is satisfied. At each time step, the value of a grid node is computed using the values of its neighboring nodes. Parallel computing strategies are necessary to apply in the case of large problems. Such problems can be parallelized explicitly by performing a data or a functional decomposition. Many algorithms that solve partial differential equation follow the steps indicated below [10]:

1. discretize the domain using a mesh M;
2. decompose the mesh M into N submeshes M_i such that such that $M = \bigcup_{i=1}^{N} M_i$;
3. generate in parallel a distributed linear system of equations;
4. solve the distributed linear system in parallel using an iterative solver.

The parallel design methodology usually consists of four distinct stages: partitioning, communication, agglomeration and mapping, as it is illustrated in the figure 4.1 [23]. Typically, a region corresponding to a submesh is associated to a processor, according to a mapping that tries to match the computation grain size with the communication time between regions. The computations are carried out in phases, each phase consisting of computations on the local linear subsystem, followed

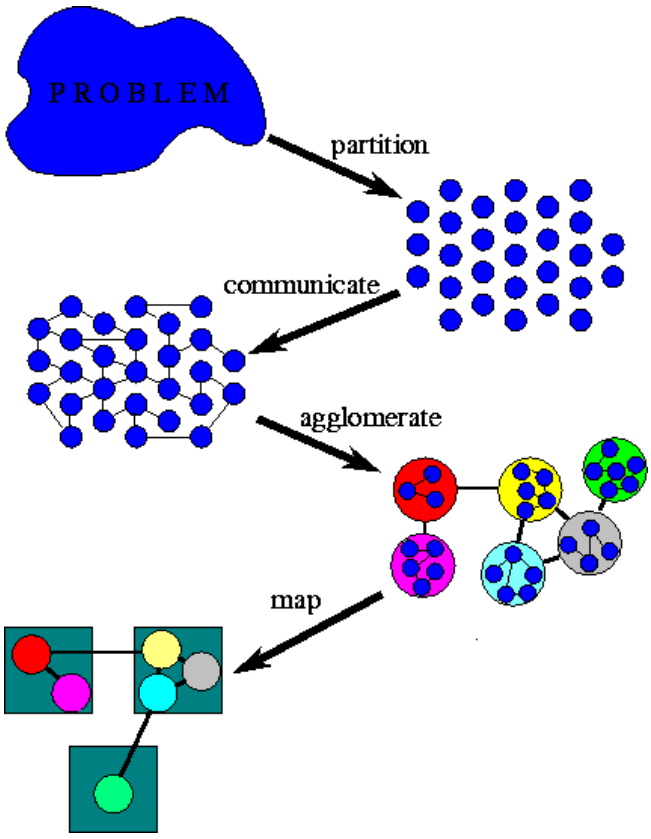


Figure 4.1: PCAM

by inter-processor communication for non-local data. Each local computation is performed in parallel for a time-step and each iteration uses the values computed in the previous step. It is possible that different conditions change:

- the lengths of the time steps may differ form one time step to the other or from one region to the other;
- the convergence rate may differ from one time step to the other or from one region to the other;
- the size or the shape of the regions my vary with the time.

In these cases it is often necessary to perform dynamic load balancing.

The general scheme of an application performing parallel adaptive computations is illustrated in Algorithm 1 ([51]). A simplified scheme is the given in Algorithm 2

4.1 Dynamic load balancing policy stages

The networks of computers are usually distributed in their nature. Their use as platforms for developing parallel applications forces the software designers to use decentralized control load-balancing policies in order to avoid bottlenecks and to achieve scalability. Local coordination policies are usually used in order to achieve this goal.

The fundamental questions that a dynamic load balancing policy should answer are: *when* should one carry out load redistribution, *how* to carry out this redistribution, *where* to transfer the load in excess. The main phases of such a policy are:

- **Load evaluation:** In order to be able to determine whether there is a load imbalance, appropriate estimations for the processors' load should be provided.
- **Profitability determination:** After having computed the load assigned to the processors (w.r.t. the given application), the load imbalance is estimated. If the cost of the imbalance exceeds the cost of application of a load balancing method, then load balancing should be carried out.

Algorithm 1 Generic algorithm

Input: M_0 - a mesh corresponding to the input geometry

if not already partitioned **then**

Partition M_0 into submeshes corresponding to the p processors

end if

$i = 0$

repeat

Assemble a sparse matrix A_i from M_i

Solve the linear system $A_i x_i = b_i$

Estimate the error on M_i

if maximum error estimate on M_i is too large **then**

Based on the error estimates, refine M_i to get M_{i+1}

if partitioning is not satisfactory for M_{i+1} **then**

Repartition M_{i+1} into submeshes corresponding to the p processors

end if

end if

$i = i + 1$

until maximum error estimate on M_i is satisfactory

Algorithm 2 Simplified scheme

repeat
  $nIter = 0$
  repeat
    $nIter = nIter + 1$
    Local computation
    Local communication
    Global communication
  until $(nIter > N)$
  Mesh adaption
  Mesh redistribution
until (global convergence criteria is satisfied)

- **Balancing flow calculation:** Based on the measurements made in the first phase, the workload transfers needed to balance the computation are calculated.
- **Selection phase:** Data items that best fulfill the flows provided by the previous step are selected for transfer or exchange. The selection process is usually subject to some restrictions; it can be expressed as an optimization problem.
- **Migration phase:** Once selected, data items are transferred between processors.

4.2 The execution time of a parallel application

The implementation of the aforementioned phases of a dynamic load balancing policy asks for some accurate estimation of the application's execution time. In a distributed environment the communication is strongly influenced by two network parameters: the latency and the bandwidth. Generally, it is assumed that the communication time between two processors takes place accordingly to the following law, which is graphically represented in the figure 4.2:

$$T_{ij}^{\text{comm}} = \alpha_{ij} + \beta_{ij} L_{ij},$$

where

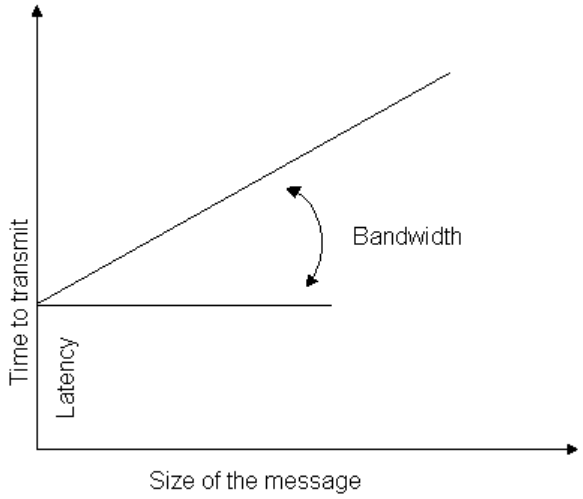


Figure 4.2: The α-β model.

- T_{ij}^{comm} denotes the communication time for transferring L_{ij} bytes between the processors i and j;
- α_{ij} is the latency of the communication between i and j;
- $\beta_{ij} = 1/b_{ij}$, where b_{ij} is the communication bandwidth between i and j;
- L_{ij} is the size of the transferred data (for instance expressed in bytes).

Usually, it is considered that the latency is the time it takes to send a 0-bytes message from one processor to another. The bandwidth is the amount of data that can be sent from one processor to another in a given time period. The latency measures the delay caused by communication between processors and memory modules over the network in a parallel system. This is often called the "$\alpha - \beta$" model ([41]) and is written

$$T = \alpha + \beta\, n.$$

Usually, $\alpha >> \beta >>$ time per floating point operation. This shows that one long message is cheaper than many short ones.

For a given application, we assume that $\mathcal{D}$ is a collection of elementary data and $P = \{1, \ldots, p\}$, the set of processors. An allocation is an application $A : \mathcal{D} \longrightarrow P$. Then, the execution time of the

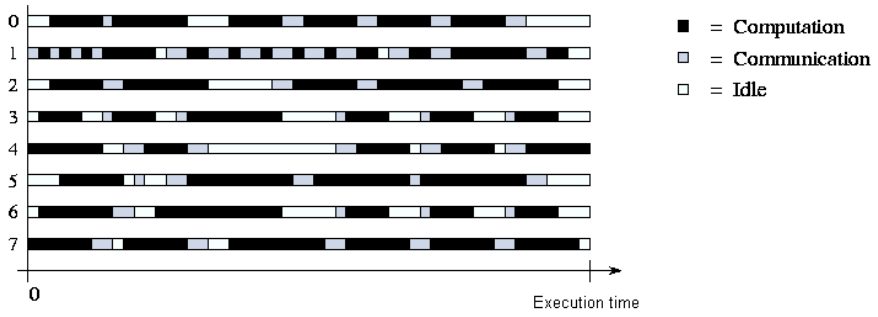


Figure 4.3: Possible execution schedule of a parallel program on eight processors.

application for an allocation A on the set of processors P is

$$T(A) = \max_{i \in P} T_i(A),$$

where $T_i(A)$ is the execution time of processor i. This can be expressed as

$$T_i(A) = T_i^{\text{comp}}(A) + T_i^{\text{comm}}(A) + T_i^{\text{sync}}(A),$$

where,

- $T_i^{\text{comp}}(A)$ denotes the total time spent by the processor i for performing computations,
- $T_i^{\text{comm}}(A)$ denotes the total time spent by the processor i for communicating with other processors,
- $T_i^{\text{sync}}(A)$ is the total inactivity time due to synchronization.

Considering now the case of an application performing adaptive computations, which matches the scheme indicated in algorithm 2, the above terms can be expressed as follows:

- The elapsed computation time is:

  $$T_i^{\text{comp}} = \alpha_i \, f(N_i)$$

  where N_i denotes the number of elements on processor i, α_i is the time it takes to the processor i to process an element and f is a function.

- The elapsed local communication time spent by each processor i can be estimated as

$$T_i^{comm} = T_i^{comm,1} + T_i^{comm,2},$$

with $T_i^{\text{comm},1}$ and $T_i^{\text{comm},2}$ defined as shown below. $T_i^{\text{comm},1}$ is the time for performing point to point communications and can be estimated as

$$T_i^{\text{comm},1} = \sum_{j=1}^{|\mathcal{N}(p)|} \left(N_{ij}\, l_{ij} + \frac{I_{ij}}{b_{ij}} \right)$$

where

  - b_{ij} is bandwidth of the network interconnection between processor i and j;
  - l_{ij} is the latency of the network interconnection between the processors i and j ;
  - I_{ij} is the number of bytes exchanged between the processors i and j ; it can be taken proportionally to the edge-weight between i and j.
  - N_{ij} is the number of messages exchanged between the processors i and j .

- $T_i^{\text{comm},2}$ represents the time for performing global communication. Many global operations can be performed with $O(\log(p))$ parallel point-to-point communication phases (usually using a tree or hypercube communication model).

Therefore, the execution time of the algorithm 2 on processor i can be expressed as

$$\max_i (T_i^{\text{comp}} + T_i^{\text{comm}} + T_i^{\text{sync}}) + T^{\text{redist}},$$

where T^{redist} is the redistribution time.

One is interested to minimize the execution time of a parallel application. Unfortunately, the maximum norm is not easy to handle. However, it is common practice to use instead the 1-norm i.e.to minimize an objective function of type

$$f(A) = \frac{1}{p} \sum_{i=1}^{p} (T_i^{\text{comp}}(A) + T_i^{\text{comm}}(A) + T_i^{\text{sync}}(A)) + T^{\text{redist}}.$$

This corresponds in fact to minimizing the *average execution time*. The problem of finding an allocation of data to processors that minimizes the above function is NP-hard.

However, it is possible that a dynamic allocation result in a degradation of the performance of the application. Then, a dynamic load balancing algorithm is considered to be efficient if

$$T_{\text{app\_with\_DLB}} + T_{\text{DLB}} < T_{\text{app\_without\_DLB}}.$$

4.3 Validation methodology

4.3.1 Motivation for building a testbed for adaptive computations

The number of scientific and industrial parallel applications has significantly grown in the last decade. However, the existing software is still characterized by a lack of generalization and standardization, and, maybe most importantly, by a lack of adequate documentation. This reduces the possibility of reusing already available codes, many researchers/programmers preferring to write their own programs from scratch. On the other hand, developing parallel software for scientific and computational engineering problems requires significant expertise and effort, as the applications tend to involve more and more advanced interdisciplinary knowledge.

4.3.2 Benefits

A key problem that any parallel application must cope with is that of ensuring a fair utilization of the available system resources at runtime. Developing specialized software modules, particularly a dynamic load balancing module, requires validation for a broad set of test cases. In many cases the validation is done with respect to some specific applications and/or hardware architectures. The absence of clearly defined reference testbeds for testing various dynamic load balancing methods constitutes a limiting factor for the developers. On the other hand, developing a reusable simulation framework may contribute to a substantial reduction of the development costs and permits to satisfy the requirements of large, complex simulations. Thus, the development time of applications attempting to solve large problems in science and engineering and involving extensive software and hardware resources can be significantly reduced.

4.3.3 Building a testbed

Our goal was to provide a general framework incorporating dynamic load balancing support for a large class of applications in the area of scientific computing, respecting their general characteristics, and at the same time enabling the user the possibility to program his own application with reasonable effort, without having to cope with the dynamic load balancing aspects.

In heterogeneous environments, in addition to parameters that are dependent on the application, the processor related parameters as well as the network related parameters must be taken into account for determining an optimal load distribution, for improving the execution time of the entire application. Specifically, physical parameters of both the interconnection network and the processors, such as the latency, bandwidth, processor capacity, memory, which can be ignored in homogeneous systems, must be considered. In order to be able to test the effectiveness of our dynamic load balancing techniques, a testbed simulating the behaviour of a generic parallel application that performs adaptive computations was proposed. We designed HeRMeS, a tool that allows to perform such simulations and that includes several general methods for dynamic load balancing and uses the system capabilities to compute and perform a new load redistribution.

The goals of the simulation are:

- to test the effectiveness of the assumed theoretical model;
- to compare the proposed dynamic load redistribution techniques;
- to test the proposed algorithms in real situations.

The testbed consists in the following software modules: the *Application simulation module*, the *Data management module*, the *Dynamic load balancing module*, the *Resource monitoring module*. Additionally, an independent module simulating the activity of external users/processes, the *Extra-load module*, and a *Visualization module* were further implemented. The description and the functionality of these modules is detailed in chapter 10.

However, before starting the development of such testbed, several problems should be addressed, such as how to measure the absolute speed of a machine or how to to measure the network and processor parameters? We get basic system information by using home-made benchmarks executed before starting the actual simulation. Online values for the system parameters, such as bandwidths, latencies, CPU capacities, memory sizes, are provided by the Network Weather Service [92] tool (NWS), which periodically monitors and dynamically forecasts the performance of the network and of the computational resources over a given period of time.

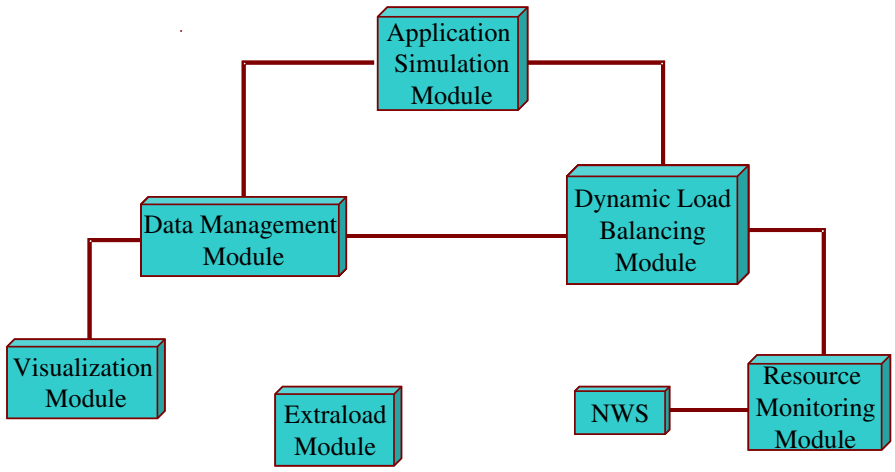


Figure 4.4: The general scheme of HeRMeS

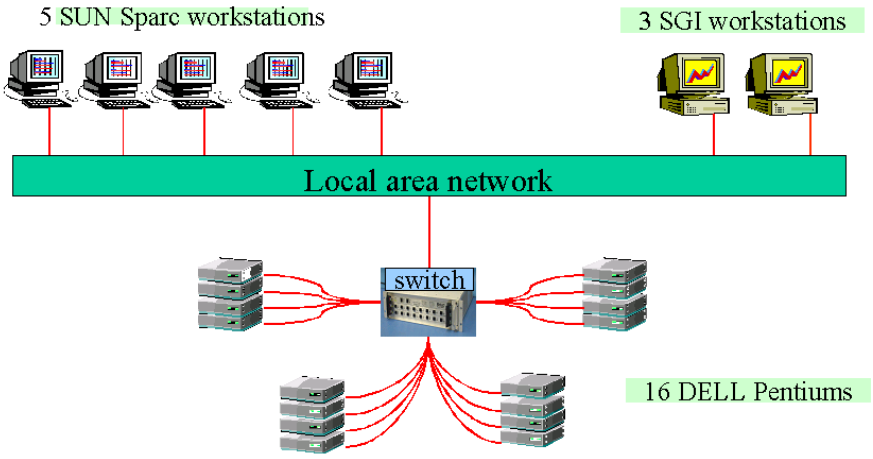
Table 4.1: Heterogeneous computational environment used for performing tests.

Machine name	OS	processor type	number	CPU	RAM
PC-151 .. PC-165	Solaris 2.7	i386	16	166 Mhz	32 M
peridot	Solaris 2.7	sparcv9	1	167 MHz	64 M
jargon	Solaris 2.7	sparc	1	110 MHz	64 M
outremer	Solaris 2.7	sparc	1	110 MHz	64 M
zircon	Solaris 2.8	sparc	1	70 MHz	32 M
tourmaline	Solaris 2.8	sparc	1	60 MHz	32 M
girasol	Solaris 2.8	sparc	1	75 MHz	32 M

4.4 Experimental setup

During the tests we used a computational environment built using commodity hardware available in our institute. It consists of a collection of heterogeneous machines with the characteristics given in table 4.1.

We installed LAM/MPI ([40]) and Network Weather Service 2.0.6 on these machines. In a first step, we focused mainly on testing the inter-operability between the dynamic load balancing module and the resource monitoring module. A test suite was generated using different load distribution and communication topologies, which are provided by the data management module. The application HeRMeS was executed on this environment. A number of experiments were conducted, by varying the work complexities, the network traffic and the processor's external loads in a controlled way, in



order to get a closer picture of how the system parameters influence the total execution time.

4.5 Heterogeneous Computing Model. Notations.

In the sequel, we assume we have to run a data-parallel application. In our model, we assume that the processors may have different relative speeds and we abstract the speed of a processor by a real positive number called *processing capacity*. A processor's workload is supposed to be infinitely divisible, so it can be represented by a positive real number. Different costs are assumed for the communication between the processors. It is also assumed that the communication topology may change between two load redistribution phases.

We consider a heterogeneous computing model (G, l, c, w) in which:

- $G = (V, E)$ is a connected graph whose vertices correspond to the processors and whose edges reflect computational dependencies with respect to a given application. Let $V = \{1, \ldots p\}$ and $E = \{e_1, e_2, \ldots e_q\}$.
- l is the vector of the processors' workloads;
- c is the vector of the processors' capacities; without loss of generality we shall consider that c is normalized relative to the 1-norm, i.e. $\sum_{1 \leq i \leq |V|} c_i = 1$.
- w_k, $1 \leq k \leq q$, are weights associated to the edges of G. W is the $q \times q$ diagonal matrix of

these weights.

The following notations are used:

- For a vector u, $\mathrm{diag}(u)$ denotes the diagonal matrix with the elements of u on the main diagonal.
- e is the vector of size p having all entries equal to 1;
- $c^{1/2}$ is the *vector of the square roots of the capacities*;
- $c_{\max}$ and $c_{\min}$ are the *maximum and the minimum capacity*;
- $D = \mathrm{diag}(c)$;
- $\overline{l}$ is the workload vector corresponding to the *fair distribution* i.e. one has, for all i and j, $\overline{l}_i/c_i = \overline{l}_j/c_j$;
- δ_i is the *degree of vertex* i, $\delta_i^w = \sum_{k:e_k=\{i,j\}} w_k$ is the *weighted degree* of the vertex i;
- $\Delta = \max_{i=1,p} \delta_i$ is *the maximum degree* of G, $\Delta^w = \max_{i=1,p} \delta_i^w$ is the *maximum weighted degree* of G;
- $e(G)$ is the *edge connectivity* of the graph G; it represents the minimum cardinal number of a set of edges whose removal disconnects the graph;
- $\mathcal{N}(i)$ is the *set of neighbors* of a node i;
- $\mathrm{diam}(G)$ is the diameter of the graph G:

$$\mathrm{diam}(G) = \max_{i,j \in V(G)} \{|P_{ij}| \ |P_{ij} \text{ is the shortest path between } i \text{ and } j\}.$$

- With respect to an arbitrary chose orientation of the edges of G, the extremities of an arc are called *head* or *tail*), depending on their position in the ordered pair that defines the arc. The *incidence matrix* A of G is a $p \times q$ matrix such that

$$a_{ij} = \begin{cases} 1, & \text{if } v_i \text{ is the head of } e_j, \\ -1, & \text{if } v_i \text{ is the tail of } e_j, \\ 0, & \text{otherwise.} \end{cases}$$

One should remark that $A^T e = 0$;

- A flow $f \in \mathbb{R}^q$ on the edges of G is called a *balancing flow* if $Af = l - \overline{l}$;
- $\|\cdot\|_q$ denotes the q-norm and $\|\cdot\|_{q,W}$ the weighted q-norm, that is $\|x\|_q = (\sum_{i=1}^{p} |x_i|^q)^{1/q}$ and $\|x\|_{q,W} = \|Wx\|_q$, for any vector $x \in \mathbb{R}^p$ and a diagonal matrix W with positive elements. Unless specified otherwise, $\|\cdot\|$ will denote the 2-norm (or the Euclidian norm);
- For any square matrix M, $\rho(M)$ denotes its *spectral radius* and λ_M *the second largest eigenvalue in absolute value of* M.

  Definition 4.5.1. A $q \times q$ real matrix P is an orthogonal projection matrix if it satisfies $P = P^T$ and $P^2 = P$.

- For any matrix $A_{n\times n}$ diagonalizable, with the spectrum $\{\lambda_1, \lambda_2, \ldots \lambda_k\}$, there exist a spectral decomposition

$$A = \lambda_1 G_1 + \lambda_2 G_2 + \ldots + \lambda_k G_k,$$

  where G_i is the projector onto $N(A - \lambda I)$ along $R(A - \lambda I)$, $G_i G_j = 0$, for all $i \neq j$ and $\sum_i G_i = I$, where $N(B)$ is the nullspace of B and $R(B)$ is the range of B [60].

- For a given model (G, l, c, w), the $p \times p$ matrix $L = AWA^T$, where A is the vertex-edge incidence matrix, is a *weighted Laplacian matrix* of G and $\mathcal{L} = D^{-1/2} L D^{-1/2}$ is a *generalized Laplacian matrix.*
- GDA is an abbreviation for *Generalized Diffusion Algorithm*
- GDM is an abbreviation for *Generalized Diffusion Matrix*
- SOS is an abbreviation for *Second Order Scheme*
- LPS is an abbreviation for *Laplacian Polynomial based Scheme*
- LPA is an abbreviation for *Laplacian Polynomial based Algorithm*
- MPA is an abbreviation for *Minimum Generating Polynomial based Algorithm*
- AdA is an abbreviation for *Adaptive Algorithm* or *Incremental Algorithm*

Chapter 5

Generalization to the heterogeneous case of the diffusion algorithm

A main concern when using distributed systems for running parallel applications is the efficient use of the available computational resources. As the execution time of the entire application is determined by the slowest processor, the total workload must be fairly distributed, i.e. proportionally to the available processing capacities. From time to time, it may be necessary to proceed to a redistribution, as during the execution both the workload assigned to a processor and its processing capacity may change, often in an unpredictable manner. Parallel adaptive finite element applications arising in Computational Fluid Dynamics or Computational Mechanics constitute typical examples illustrating the need for dynamic load balancing [30, 44]. Such applications rely on unstructured meshes, whose nodes represent finite elements/volumes or vertices of the finite elements/volumes and the links between them reflect neighborhood relations. In the course of a computation, the adaptive meshes may undergo significant changes due to refinement/de-refinement operations resulting in an uneven partitioning. It is well known that in distributed systems the inter processor communication is costly; consequently, a fair workload redistribution must be done with the least data migration

and so that the number of dependencies between nodes residing on different processors is minimized. A centralized approach leads to heavy communication and it is not scalable. Therefore, a dynamic load balancing algorithm should use local communication and should be able to rapidly compute a new fair data distribution. The nearest-neighbor algorithms meet these requirements to a large extent. For this reason they were intensively studied in the last years, mainly in the context of homogeneous systems. They lead an unbalanced system to a global "equilibrium" state by exchanging information/workload between neighboring processors only. The most popular variants are the diffusion [13, 6, 43, 16, 33], the dimension exchange [57, 13, 93], the multi-level difusion [34], and the gradient model [91]. Less attention has been paid to dynamic load balancing in heterogeneous systems [48, 47, 14]. An intuitive algorithm based on a hydrodynamic analogy was proposed by Hui et al. [48, 47]. Watts et al. [88] proposed an implicit diffusion scheme. In both cases it was assumed that the processors may have different performances but that communication is uniform. We treated the same case in a first step [66, 67], and we gave a generalization of the algorithm described by Boillat [6]. Elsässer et al. [19] generalized a variant of diffusion that was originally proposed by Cybenko [13].

The efficient execution of parallel applications on heterogeneous systems requires adequate dynamic load balancing techniques. Such methods should be fast and minimize the communication overhead. The problem to solve is to *determine the amount of workload to shift between adjacent vertices in the communication graph induced by the application (which is mapped onto a heterogeneous network of processors), so that each processor has the same proportion of workload to process relative to its capacity and the cost of data migration is minimized.* The minimum discussed here is with respect to the norm $\|\cdot\|_{2,W^{-1/2}}$.

The generalized diffusion algorithm (GDA for short) is a generalization of the classical diffusion algorithm and it reads as in the algorithm 3. The vector $l^{(0)}$ is the initial workload vector and $l^{(n)}$ is the workload vector after the n-th iteration. The algorithm can be expressed as an iterative process of the form $l^{(n+1)} = Ml^{(n)}$, with M a $p \times p$ nonnegative matrix such that

$$\begin{cases} m_{ij} > 0, \text{ if and only if } \{i,j\} \in E \text{ or } i = j; \\ \sum_i m_{ij} = 1, \forall\, j \in V; \\ m_{ij}c_j = m_{ji}c_i, \text{ for all } i,j. \end{cases} \tag{5.0.1}$$

A matrix satisfying the above properties is called *generalized diffusion matrix* (GDM for short).

Algorithm 3 Algorithm GDA

$k = 0$;

while (not converged) **do**

for all i **do**

send $l_i^{(k)}$ to neighbors

receive $l_j^{(k)}$ from all neighbors j

$\delta_{ij}^{(k)} = m_{ji}\, l_i^{(k)} - m_{ij}\, l_j^{(k)}$

$l_i^{(k+1)} = l_i^{(k)} - \sum_{(i,j)} \delta_{ij}^{(k)}$

$k = k + 1$

end for

end while

The first condition expresses that at each iteration step the communication is allowed only between neighbors. The second constraint guarantees the workload conservation in the absence of external disturbances. The equations in the third condition express the fact that no load should be migrated between processors in the fair state. These equations are referred to as *balance equations*, a term used in the theory of reversible Markov chains, as the above iterative process defines such a Markov chain. This condition ensures that M has real eigenvalues and that $\bar{l} = M\bar{l}$ as will be shown in the next section.

5.1 Convergence and metrics

Definition 5.1.1. A square $p \times p$ matrix M is *reducible* if, for some r with $0 < r < p$, there exists a $p \times p$ permutation matrix P such that

$$PMP^T = \begin{bmatrix} M_{11} & M_{12} \\ 0 & M_{22} \end{bmatrix},$$

where M_{11} is an $r \times r$ submatrix and M_{22} is an $(p-r) \times (p-r)$ submatrix. If no such permutation matrix exists, M is *irreducible*.

Definition 5.1.2. Let M be a nonnegative irreducible square matrix and $\rho(M)$ its spectral radius. The matrix M is *primitive* if it has exactly one eigenvalue whose absolute value is equal to $\rho(M)$.

Theorem 1 ([82]). *Let M be a nonnegative square matrix. M^n has only positive elements for some natural number n if and only if M is primitive.*

Theorem 2. *A generalized diffusion matrix is irreducible and primitive.*

Proof. It suffices to show that if M is a square nonnegative matrix satisfying the properties (5.0.1), then M^p has only positive elements. □

A consequence of satisfying the set of conditions 5.0.1 is that a generalized diffusion matrix have the spectral radius 1 and is primitive.

The *Perron vector* of a GDM is the eigenvector corresponding to the spectral radius and with the sum of elements equal to 1 [60].

Lemma 1. *The capacity vector of any GDM coincides with its corresponding Perron vector.*

Proof. Let M be a GDM. By properties (5.0.1), $\sum_j m_{ij}c_j = \sum_j (m_{ji}c_i/c_j)c_j = c_i(\sum_j m_{ji}) = c_i$. Hence, $Mc = c = \rho(M)c$. □

According to the Perron-Frobenius theorem [60, 82], c and its multiples are the only eigenvectors of M with all entries nonnegative. The workload vector corresponding to a fair distribution is the Perron vector multiplied by the sum of the workloads i.e. $\overline{l} = c(e^T l^{(0)})$.

Theorem 3. *A generalized diffusion matrix is diagonalizable and all its eigenvalues are real.*

Proof. A generalized diffusion matrix M is similar to $D^{-1/2}MD^{1/2}$, which is real symmetric. Therefore, the eigenvalues of M are real. □

In the sequel, we assume that the eigenvalues of a GDM are indexed so that

$$1 = \lambda_1 > \lambda_2 \geq \cdots \geq \lambda_p > -1.$$

For a generic GDM M, we denote by λ_M its *second largest eigenvalue* in absolute value, i.e. $\max\{|\lambda_2|, |\lambda_p|\}$.

Theorem 4. *The sequence $l^{(n)}$ tends to $\overline{l}$ when n tends to ∞.*

Proof. Being diagonalizable, there exists a spectral decomposition of M [60]. Therefore, $M = \lambda_1 G_1 + \ldots + \lambda_p G_p$, G_i's being its orthogonal spectral projectors and λ_i's its eigenvalues. We know that M has real eigenvalues and that $\rho(M) = 1$. Thus, we have

$$\left(\frac{M}{\lambda_1}\right)^n = G_1 + \left(\frac{\lambda_2}{\lambda_1}\right)^n G_2 + \ldots + \left(\frac{\lambda_p}{\lambda_1}\right)^n G_p.$$

Consequently, $M^n l^{(0)} \longrightarrow G_1 l^{(0)}$ when $n \longrightarrow \infty$. But, since $\lambda_1 = 1$ is a simple eigenvalue of M and c and e^T are its corresponding right and left eigenvectors, $G_1 = ce^T/(e^T c)$ [60]. As in our case $e^T c = 1$, we are left with $Ml^{(0)} \longrightarrow ce^T l^{(0)}$ and the convergence speed depends on $\lambda_M = \max\{|\lambda_2|, |\lambda_p|\}$. □

The above result indicates that the smaller λ_M, faster the generalized diffusion process converges. To be consistent with the homogeneous case [6], the term *convergence factor* will be used when referring to it.

The load imbalance existing in the system at a given time can be expressed using various metrics. We shall discuss some of them in the following subsections.

The convergence using the weighted 2-norm

We show that the distance induced by this norm, between the current workload vector and the asymptotic workload vector, improves after each iteration step.

Lemma 2. *For all nonnegative n, the sequence $\{l^{(n)}\}$ satisfies*

$$\|l^{(n+1)} - \overline{l}\|_{2,D^{-1/2}} \leq \lambda_M \, \|l^{(n)} - \overline{l}\|_{2,D^{-1/2}}.$$

Proof. The matrix $M' = D^{-1/2} M D^{1/2}$ is real symmetric and has the same eigenvalues as M. The vector $c^{1/2}$ is an eigenvector of M' corresponding to the greatest eigenvalue 1. The matrix M' being real symmetric, the Courant-Fischer principle can be applied: it gives

$$\lambda_M^2 = \max_{\substack{x \neq 0 \\ x^T c^{1/2} = 0}} \frac{x^T M'^2 x}{x^T x} = \max_{\substack{x \neq 0 \\ x^T c^{1/2} = 0}} \frac{\|M'x\|^2}{\|x\|^2}.$$

Let $x^{(n)}$ be $D^{-1/2}(l^{(n)} - \overline{l})$. Since the total workload in the system is conserved, it follows that, for all nonnegative n, $(x^{(n)})^T c^{1/2} = (l^{(n)} - \overline{l})^T D^{-1/2} c^{1/2} = (l^{(n)} - \overline{l})^T e = 0$. Hence we have, for all

nonnegative n,

$$\lambda_M^2 \geq \frac{\|M'x^{(n)}\|^2}{\|x^{(n)}\|^2} = \frac{\|D^{-1/2}M(l^{(n)} - \overline{l})\|^2}{\|D^{-1/2}(l^{(n)} - \overline{l})\|^2} = \frac{\|D^{-1/2}(l^{(n+1)} - \overline{l})\|^2}{\|D^{-1/2}(l^{(n)} - \overline{l})\|^2},$$

which is the inequality we had to prove. □

Theorem 5. *For all nonnegative n, the sequence $\left\{l^{(n)}\right\}$ satisfies*

$$\|l^{(n)} - \overline{l}\|_{2,D^{-1/2}} \leq \lambda_M^n \, \|l^{(0)} - \overline{l}\|_{2,D^{-1/2}}.$$

Proof. Using the above lemma, the proof is straightforward. □

The convergence using the 2-norm

We investigate here the load imbalance measured by the distance induced by the Euclidian norm.

Theorem 6. *For all nonnegative n, the sequence $\left\{l^{(n)}\right\}$ satisfies*

$$\|l^{(n)} - \overline{l}\| \leq \lambda_M^n \sqrt{\frac{c_{\max}}{c_{\min}}} \, \|l^{(0)} - \overline{l}\|.$$

Proof. The matrices M and $M' = D^{-1/2}MD^{1/2}$ are similar. The last matrix is real symmetric and it has a complete and orthonormal set of eigenvectors. Let λ_i, $1 \leq i \leq p$, denote its eigenvalues in the assumed ordering and v_i the corresponding orthonormal eigenvectors. Thus there are coefficients a_i such that

$$D^{-1/2}(l^{(0)} - \overline{l}) = \sum_i a_i v_i.$$

As $v_1 = c^{1/2}$ is the eigenvector corresponding to the unique eigenvalue of absolute value 1 and as its norm is 1, it follows that

$$\begin{aligned} a_1 &= (c^{1/2})^T \sum_i a_i v_i = (c^{1/2})^T D^{-1/2}(l^{(0)} - \overline{l}) \\ &= e^T(l^{(0)} - \overline{l}) = \sum_i l_i^{(0)} - \sum_i \overline{l}_i = 0. \end{aligned}$$

Therefore, we have

$$\begin{aligned} \|l^{(n)} - \overline{l}\| &= \|M^n l^{(0)} - \overline{l}\| = \|M^n l^{(0)} - M^n \overline{l}\| \\ &= \|M^n(l^{(0)} - \overline{l})\| = \|D^{1/2}(D^{-1/2}MD^{1/2})^n D^{-1/2}(l^{(0)} - \overline{l})\| \\ &= \|D^{1/2}(D^{-1/2}MD^{1/2})^n(\sum_i a_i v_i)\| = \|D^{1/2}(\sum_{i \neq 1} \lambda_i^n a_i v_i)\| \\ &\leq \lambda_M^n \|D^{1/2}\| \|D^{-1/2}(l^{(0)} - \overline{l})\| = \lambda_M^n \sqrt{\frac{c_{\max}}{c_{\min}}} \, \|l^{(0)} - \overline{l}\|. \end{aligned}$$

□

The convergence using the weighted ∞-norm

We define the *weighted ∞-norm* of a vector x as

$$\max_i \frac{|x_i|}{c_i}.$$

We show that the distance induced by this norm, between the current workload vector and the asymptotic workload vector decreases or remains the same after each iteration step. Moreover it reduces exponentially with the number of iterations.

Lemma 3. *For all nonnegative n, the sequence $\{l^{(n)}\}$ satisfies*

$$\max_i \frac{|l_i^{(n+1)} - \overline{l}_i|}{c_i} \leq \max_i \frac{|l_i^{(n)} - \overline{l}_i|}{c_i}.$$

Proof. We have

$$\frac{l_i^{(n+1)} - \overline{l}_i}{c_i} = \sum_j \frac{m_{ij}}{c_i}(l_j^{(n)} - \overline{l}_j) = \sum_j m_{ji} \frac{l_j^{(n)} - \overline{l}_j}{c_j}.$$

As $\sum_j m_{ji} = 1$, one has

$$\max_i \frac{|l_i^{(n+1)} - \overline{l}_i|}{c_i} \leq \max_j \frac{|l_j^{(n)} - \overline{l}_j|}{c_j} (\sum_j m_{ji}) = \max_j \frac{|l_j^{(n)} - \overline{l}_j|}{c_j}.$$

□

We denote the (i,j)-th element of the matrix M^n by $m_{ij}^{(n)}$. We further consider

$$\Delta(n) = \max_{i,j \in V} \frac{|m_{ij}^{(n)} - c_i|}{c_i}.$$

Lemma 4. *The following inequality holds for any nonnegative n:*

$$\Delta(n) \leq \frac{\lambda_M^n}{c_{\min}}.$$

Proof. We define g_i as the vector whose components are all 0, except the i-th that is 1. Furthermore, we define the coefficients $a_k^{(i)}$ by $D^{1/2} g_i = \sum_k a_k^{(i)} v_k$, where the $\{v_i\}$ is a set of orthonormal eigenvectors of $M' = D^{-1/2} M D^{1/2}$ as in the proof of theorem 6. It follows that $D^{-1/2} g_i = \sum_k a_k^{(i)} v_k / c_i$.

As we saw that $v_1 = c^{1/2}$, we have $a_1^{(i)} = (c^{1/2})^T D^{1/2} g_i = c_i$ and

$$\begin{aligned}
\Delta(n) &= \max_{i,j\in V} \frac{|g_i^T M^n g_j - c_i|}{c_i} \\
&= \max_{i,j\in V} \frac{|g_i^T D^{1/2}(D^{-1/2} M D^{1/2})^n D^{-1/2} g_j - c_i|}{c_i} \\
&= \max_{i,j\in V} \frac{|(c_i v_1^T + \sum_{k\neq 1} a_k^{(i)} v_k^T) M'^n (c_j v_1 + \sum_{l\neq 1} a_l^{(j)} v_l)/c_j - c_i|}{c_i} \\
&= \max_{i,j\in V} \frac{|(c_i v_1^T + \sum_{k\neq 1} a_k^{(i)} v_k^T)(c_j v_1 + \sum_{l\neq 1} a_l^{(j)} \lambda_l^n v_l)/c_j - c_i|}{c_i} \\
&= \max_{i,j\in V} \frac{|c_i + \sum_{k\neq 1} a_k^{(i)} a_k^{(j)} \lambda_k^n/c_j - c_i|}{c_i} \\
&\leq \lambda_M^n \max_{i,j\in V} \frac{\sum_{k\neq 1} |a_k^{(i)} a_k^{(j)}/c_j|}{c_i} \leq \lambda_M^n \max_{i,j\in V} \frac{\sum_k |a_k^{(i)} a_k^{(j)}/c_j|}{c_i} \\
&\leq \lambda_M^n \max_{i,j\in V} \frac{\|D^{1/2} g_i\| \|D^{-1/2} g_j\|}{c_i} \leq \frac{\lambda_M^n}{c_{\min}}.
\end{aligned}$$

□

Theorem 7. *For all nonnegative n, the sequence $\{l^{(n)}\}$ satisfies*

$$\max_i \frac{|l_i^{(n)} - \overline{l}_i|}{c_i} \leq \frac{\lambda_M^n}{c_{\min}} \max_i \frac{|l_i^{(0)} - \overline{l}_i|}{c_i}.$$

Proof. Let $m_{ij}^{(n)}$ denote again the element at position (i,j) of the matrix M^n. Then, we have

$$\begin{aligned}
\max_{i\in V} \frac{|l^{(n)} - \overline{l}_i|}{c_i} &= \max_{i,j\in V} \frac{|\sum_j m_{ij}^{(n)} l_j^{(0)} - c_i \sum_j l_j^{(0)}|}{c_i} \\
&= \max_{i,j\in V} \frac{|\sum_j m_{ij}^{(n)} l_j^{(0)} - \sum_j m_{ij}^{(n)} \overline{l}_j + c_i \sum_j l_j^{(0)} - c_i \sum_j l_j^{(0)}|}{c_i} \\
&= \max_{i,j\in V} \frac{|\sum_j (m_{ij}^{(n)} - c_i)(l_j^{(0)} - \overline{l}_j)|}{c_i} \leq \max_{i,j\in V} \frac{|m_{ij}^{(n)} - c_i|}{c_i} \sum_j |l_j^{(0)} - \overline{l}_j| \\
&\leq \frac{\lambda_M^n}{c_{\min}} \sum_j c_j \frac{|l_j^{(0)} - \overline{l}_j|}{c_j} \leq \frac{\lambda_M^n}{c_{\min}} (\sum_j c_j) \max_i \frac{|l_i^{(0)} - \overline{l}_i|}{c_i} = \frac{\lambda_M^n}{c_{\min}} \max_j \frac{|l_j^{(0)} - \overline{l}_j|}{c_j}.
\end{aligned}$$

□

The convergence using the Manhattan distance

Definition 5.1.3. The *Manhattan distance* between two vectors x and y is defined as

$$d_M(x,y) = \sum_i |x_i - y_i|.$$

As in the previous case, we show that the Manhattan distance $\sum_i |l_i^{(n)} - \bar{l}_i|$ decreases or remains the same after each iteration step of a generalized diffusion algorithm. Moreover, after a sufficient number of steps the total load imbalance decreases geometrically.

Lemma 5. *For all nonnegative n, the sequence $\{l^{(n)}\}$ satisfies*

$$\sum_i |l_i^{(n+1)} - \overline{l_i}| \leq \sum_i |l_i^{(n)} - \overline{l_i}|.$$

Proof. We have

$$l_i^{(n+1)} - \bar{l}_i = \sum_j m_{ij} l_j^{(n)} - \sum_j m_{ij} \bar{l}_j = \sum_j m_{ij} (l_j^{(n)} - \bar{l}_j).$$

It follows that

$$|l_i^{(n+1)} - \bar{l}_i| \leq \sum_j m_{ij} |l_j^{(n)} - \bar{l}_j|$$

and

$$\sum_i \sum_j m_{ij} |l_j^{(n)} - \bar{l}_j| = \sum_j (\sum_i m_{ij}) |l_j^{(n)} - \bar{l}_j| = \sum_j |l_j^{(n)} - \bar{l}_j|.$$

Therefore,

$$\sum_i |l_i^{(n+1)} - \bar{l}_i| \leq \sum_j |l_j^{(n)} - \bar{l}_j|.$$

□

Theorem 8. *For all nonnegative n, the sequence $\{l^{(n)}\}$ satisfies*

$$\sum_i |l_i^{(n)} - \overline{l_i}| \leq \frac{\lambda_M^n}{c_{\min}} \sum_i |l_i^{(0)} - \overline{l_i}|.$$

Proof. Let $m_{ij}^{(n)}$ denote the (i,j)-th element of the matrix M^n. We have

$$\begin{aligned}
\sum_i |l_i^{(n)} - \bar{l}_i| &= \sum_i |\sum_j m_{ij}^{(n)} l_j^{(0)} - \sum_j c_i l_j^{(0)}| \\
&= \sum_i |\sum_j m_{ij}^{(n)} l_j^{(0)} - \sum_j m_{ij}^{(n)} \bar{l}_j + c_i \sum_j \bar{l}_j - \sum_j c_i l_j^{(0)}| \\
&= \sum_i |\sum_j (m_{ij}^{(n)} - c_i)(l_j^{(0)} - \bar{l}_j)| \leq \sum_i \sum_j \frac{|m_{ij}^{(n)} - c_i)|}{c_i} c_i |l_j^{(0)} - \bar{l}_j| \\
&\leq \max_{i,j \in V} \frac{|m_{ij}^{(n)} - c_i)|}{c_i} (\sum_i c_i) \sum_j |l_j^{(0)} - \bar{l}_j| \leq \frac{\lambda_M^n}{c_{\min}} \sum_j |l_j^{(0)} - \bar{l}_j|.
\end{aligned}$$

□

The above relations show that after any iteration step, the distance to the fair distribution that is induced by the weighted 2-norm decreases. The distances induced by the weighted ∞-norm and weighted $1-$norm, eventually decrease. The algorithm results in a geometrical reduction of the above quantities. Unfortunately, as seen above, the convergence rate depends tightly on λ_M, whatever metric is used. Therefore, adequate estimations must be found for it. The above results shows that GDA completes in $O(1/|\ln(\lambda_M)|)$ steps.

5.2 Theoretical bounds for the convergence factor

Before deciding to carry out a workload redistribution, one should dispose of some kind of estimation of the cost of application of a dynamic load balancing algorithm. Here, we try to answer to the question *"what is the maximum number of steps that a* GDA *will take to balance the system?"*. It should be noted that faster polynomial schemes can be derived in relation with a GDM [62, 15]. But often such methods use parameters that are dependent on the eigenvalues of the Laplacian of the communication graph and are more appropriate for the use in a static context. However, as the graph induced by the current data distribution is often irregular and can change as an effect of a previous migration or adaption, the eigenvalues of the Laplacian cannot be given in advance and their computation at runtime induces a supplementary overhead.

We are concerned with finding diffusion matrices with an improved convergence factor. Adequate estimations of the convergence factor as a function of the communication graph's characteristics should exist. Bounds can be formulated by analogy with the discrete time reversible Markov chains, as the transpose of a GDM defines a discrete reversible Markov chain. In theory, greater importance is given generally to the second largest eigenvalue rather than to the smallest one. Indeed, as Sinclair pointed out [76], one can consider $M' = (1/2)(I+M)$ instead of M (therefore $\lambda_M = \lambda_2(M)$), without a significant impact on the convergence rate.

If $u_1, \ldots, u_q$ are weights associated with the edges of G and $U = \mathrm{diag}(u)$, the matrix $L = AUA^T$ is the *weighted Laplacian matrix* of the graph G. The *generalized Laplacian* matrix of G is expressed as $\mathcal{L} = D^{-1/2}LD^{-1/2}$. The smallest eigenvalue of both Laplacians is 0, ${c^{1/2}}^T$ and $c^{1/2}$ being left and right eigenvectors corresponding to this eigenvalue. We assume that the eigenvalues of L are indexed so that $\mu_1(L) = 0 \leq \mu_2(L) \leq \ldots \leq \mu_p(L)$ and $\mu_1(\mathcal{L}) = 0 \leq \mu_2(\mathcal{L}) \leq \ldots \leq \mu_p(\mathcal{L})$, respectively.

According to the Courant-Fischer principle, after some simple transformations, the following expressions can be given for the second smallest eigenvalues of L and $\mathcal{L}$:

$$\mu_2(L) = \min_{y \neq 0,\ y^T e = 0} \frac{\sum_{e_k=\{i,j\}\in E} u_k (y_i - y_j)^2}{\sum_i y_i^2}, \tag{5.2.1}$$

$$\mu_2(\mathcal{L}) = \min_{y \neq 0,\ y^T c = 0} \frac{\sum_{e_k=\{i,j\}\in E} u_k (y_i - y_j)^2}{\sum_i c_i y_i^2}. \tag{5.2.2}$$

Because G is connected, we have $\mu_2(L) > 0$ [22].

Bounds can be formulated for $\mu_2(\mathcal{L})$ using the isoperimetric constant [61], defined w.r.t. the generalized Laplacian. This constant is usually described as follows:

$$i(G) = \min \left\{ \frac{\sum_{i\in S} \sum_{j\notin S} u_k}{\sum_{i\in S} c_i} \,\middle|\, S \subset V,\ 0 < \sum_{i\in S} c_i \leq \frac{1}{2}, e_k = \{i,j\} \in E(G) \right\}. \tag{5.2.3}$$

There is a close connection between the spectra of the Laplacian matrices of a graph and that of any GDM of a diffusion on the same graph. Let L be the matrix $D - MD$. Then L is a weighted Laplacian matrix of G and, as the set of conditions (5.0.1) ensures that $m_{ij} > 0$ iff $\{i,j\} \in E(G)$, it can be put in the form $L = AUA^T$, with $u_k = m_{ij}c_j$, for all $e_k = \{i,j\} \in E$. Therefore,

$$M = I - LD^{-1} = I - D^{1/2}\mathcal{L}D^{-1/2}.$$

In this case, the isoperimetric constant coincides with the Cheeger constant of the weighted graph G [12] and with the conductance of the Markov chain defined by M^T [76]. Sinclair and Jerrum [77] showed that the above quantity satisfies the Cheeger inequality:

$$\frac{i^2(G)}{2} \leq \mu_2(\mathcal{L}) = 1 - \lambda_2(M) \leq 2\, i(G). \tag{5.2.4}$$

The bounds for the second smallest eigenvalue obtained via the Cheeger type inequalities like (5.2.4) are often tight [11]. However, as it was pointed out by Lovasz and Mohar [58, 61], it is NP-hard to determine the Cheeger type constants. We shall prefer instead to give alternative results.

Theorem 9. *Let M be a given GDM. Writing L for $D - MD$, $\mathcal{L}$ for $D^{-1/2}LD^{-1/2}$ and $\mu_2(L)$, $\mu_2(\mathcal{L})$ their corresponding smallest nonzero eigenvalues, we have*

$$\mu_2(L) \leq c_{\max}\mu_2(\mathcal{L}),$$

where $c_{\max}$ is the maximum capacity.

Proof. The proof relies on two alternative characterizations of the second smallest eigenvalues of L and $\mathcal{L}$, which can be obtained from (5.2.1) and (5.2.2) using the Lagrange identity:

$$\begin{aligned}\mu_2(L) &= \min_{y\neq 0}\max_{t}\frac{\sum_{(i,j)\in E} m_{ij}c_j(y_i-y_j)^2}{\sum_i (y_i-t)^2}\\ \mu_2(\mathcal{L}) &= \min_{y\neq 0}\max_{t}\frac{\sum_{(i,j)\in E} m_{ij}c_j(y_i-y_j)^2}{\sum_i c_i(y_i-t)^2}.\end{aligned}$$

□

Fiedler proved [21] that for any symmetric doubly stochastic matrix P, i.e. a matrix whose columns and rows sum to 1, the following inequality holds:

$$\lambda_2(P) \leq 1 - 4\,\sigma_P \sin^2\left(\frac{\pi}{2p}\right), \tag{5.2.5}$$

where σ_P is the *measure of irreducibility* of P, i.e. $\sigma_P = \min_{\emptyset\neq S\subset V}\sum_{i\in S, j\notin S} p_{ij}$.

We notice that the matrix M' defined as $M' = I - L = I - D + MD$ is a doubly stochastic matrix. Below, we formulate a more general result, applicable also for non-symmetric stochastic matrices.

Theorem 10. *Let M be a GDM. Then, we have that*

$$\lambda_2(M) \leq 1 - \frac{4}{c_{\max}}\left(\min_{\emptyset\neq S\subset V}\sum_{i\in S, j\notin S} m_{ij}c_j\right)\sin^2\left(\frac{\pi}{2p}\right). \tag{5.2.6}$$

Proof. The following inequality follows directly from the theorem 9:

$$1 - \lambda_2(M') = \mu_2(L) \leq c_{\max}\mu_2(\mathcal{L}) = c_{\max}(1-\lambda_2(M)).$$

Applying now (5.2.5) for M', one gets

$$\lambda_2(M) \leq 1 - \frac{4}{c_{\max}}\sigma_{M'}\sin^2\left(\frac{\pi}{2p}\right).$$

One should note that $\sigma_{M'} > 0$, as G is connected. □

Corollary 1. *Let M be a GDM and $\theta(M) = \min_{(i,j)}\{m_{ij}c_j\}$. Then we have*

$$\lambda_2(M) \leq 1 - 4\frac{e(G)\theta(M)}{c_{\max}}\sin^2\left(\frac{\pi}{2p}\right). \tag{5.2.7}$$

Proof. The proof is straightforward, by the above theorem and the definition of the edge connectivity.

□

In the sequel, we prove another useful bound for the second largest eigenvalue of a GDM.

Theorem 11. *Let M be a GDM,* $\mathrm{diam}(G)$ *the diameter of G and $\theta(M) = \min_{(i,j)}\{m_{ij}c_j\}$. Then*

$$\lambda_2(M) \leq 1 - \frac{\theta(M)}{\mathrm{diam}(G)} \ . \tag{5.2.8}$$

Proof. Let z be a vector such that

$$\mu_2(\mathcal{L}) = \frac{\sum_{(i,j)\in E(G)}(z_i - z_j)^2 m_{ij}c_j}{\sum_i z_i^2 c_i}.$$

Let i_0 such that $|z_{i_0}| = \max_i |z_i|$. As $\sum_i c_i z_i = 0$, there is an index j_0 such that z_{i_0} and z_{j_0} have opposite signs. Let P be a path with minimum length, joining the vertices i_0 and j_0 in the graph G and l its length. It follows that

$$\begin{aligned}
\mu_2(\mathcal{L}) &\geq \frac{\sum_{(i,j)\in E(P)}(z_i - z_j)^2 m_{ij}c_j}{{z_{i_0}}^2} \geq \frac{\theta(M)\sum_{(i,j)\in E(P)}(z_i - z_j)^2}{{z_{i_0}}^2} \\
&\geq \frac{\theta(M)\left(\sum_{(i,j)\in E(P)}(z_i - z_j)\right)^2}{l{z_{i_0}}^2} = \frac{\theta(M)}{l}\frac{(z_{i_0} - z_{j_0})^2}{{z_{i_0}}^2} \geq \frac{\theta(M)}{\mathrm{diam}(G)}
\end{aligned}$$

As $\lambda_2(M) = 1 - \mu_2(\mathcal{L})$, we get (5.2.8). □

5.3 Other estimations for the second largest eigenvalue

In this section we adapt a result proved by Mohar [61] that improves the lower bound in the inequality (5.2.4).

Theorem 12. $\mu_2(\mathcal{L}) \geq 1 - \sqrt{1 - i_M^2(G)}$.

Proof. The matrix M can be put in the form $M = I - \mathcal{L}$, with $\mathcal{L} = AUA^T$, and $u_k = m_{ij}c_j$, for all k such that $e_k = \{i,j\}$. For simplicity, we shall write u_{ij} instead of u_k, for all i,j such that $e_k = \{i,j\}$. Let x be an eigenvector corresponding to $\mu_2(\mathcal{L})$, i.e. $\mu_2(\mathcal{L})x = \mathcal{L}x = D^{-1/2}LD^{1/2}x$. We have that $x^T c^{1/2} = 0$, because x and $c^{1/2}$ are orthogonal. Let $y = D^{-1/2}x$, then

$$\mu_2(\mathcal{L})Dy = Ly \tag{5.3.1}$$

and $y^T c = 0$. Let $S = \{i | y_i > 0\}$ and z a vector having the entries $z_i = y_i$, if $i \in S$ and $z_i = 0$, otherwise. We may consider that $c(S) = \sum_{i\in S} c_i \leq 1/2$, since otherwise one can take $-y$ instead of

y. Then by (5.3.1) one has

$$\mu_2(\mathcal{L})\sum_i c_i y_i = \sum_{i\in S}[(\sum_j u_{ij})y_i - \sum_j u_{ij}y_j] = \sum_{i\in S}\sum_j u_{ij}(y_i - y_j) \tag{5.3.2}$$

and

$$\begin{aligned}\mu_2(\mathcal{L})\sum_i c_i z_i^2 &= \mu_2(\mathcal{L})\sum_{i\in S} c_i y_i^2 = \sum_{i\in S}\sum_j u_{ij}(y_i - y_j)y_i \\ &= \sum_{i\in S}\sum_{j\in S} u_{ij}(y_i - y_j)y_i + \sum_{i\in S}\sum_{j\notin S} u_{ij}(y_i - y_j)y_i \\ &\geq \sum_{i\in S}\sum_{j\in S} u_{ij}(y_i - y_j)y_i + \sum_{i\in S}\sum_{j\notin S} u_{ij}y_i^2 \\ &= \sum_{ij\in E} u_{ij}(z_i - z_j)^2 = z^T L z.\end{aligned}$$

Let us note $\alpha = z^T Lz/z^T Dz$. Then, we have

$$\mu_2(\mathcal{L}) \geq \alpha.$$

On the other hand,

$$\begin{aligned}&\sum_{ij\in E} u_{ij}(z_i + z_j)^2 = 2\sum_{ij\in E} u_{ij}(z_i^2 + z_j^2) - \sum_{ij\in E} u_{ij}(z_i - z_j)^2 \\ &= 2\sum_i\sum_j u_{ij}z_i^2 - z^T Lz = 2z^T Dz - z^T Lz \geq (2-\alpha)z^T Dz\end{aligned}$$

and

$$\alpha = \frac{\sum_{ij\in E} u_{ij}(z_i - z_j)^2 \sum_{ij\in E} u_{ij}(z_i + z_j)^2}{(z^T Dz)\sum_{ij\in E} u_{ij}(z_i + z_j)^2} \geq \frac{(\sum_{ij\in E} u_{ij}|z_i^2 - z_j^2|)^2}{(2-\alpha)(z^T Dz)^2}.$$

Let $0 = t_0 < t_1 < t_2 < \ldots < t_r$ be all distinct values of z_i, for all $1 \leq i \leq p$. Let $V_k = \{i \in V | t_k \leq z_i\}$. Note that $0 < c(V_k) \leq c(S) \leq 1/2$. Then, we have

$$\begin{aligned}&\sum_{ij\in E} u_{ij}|z_i^2 - z_j^2| = \sum_{k=1,r}\sum_{ij\in E, z_j < z_i = t_k} u_{ij}(z_i^2 - z_j^2) \\ =\ &\sum_{k=1,r}\sum_{z_i = t_k, z_j = t_l, l<k} u_{ij}(t_k^2 - t_{k-1}^2 + t_{k-1}^2 - t_{k-2}^2 + \cdots + t_{l+1}^2 - t_l^2) \\ =\ &\sum_{k=1,r}(\sum_{i\in V_k}\sum_{j\notin V_k} u_{ij})(t_k^2 - t_{k-1}^2) \geq i_M(G)\sum_{k=1,r} c(V_k)(t_k^2 - t_{k-1}^2) \\ =\ &i_M(G)\sum_{k=1,r} t_k^2(c(V_k) - c(V_{k+1})) = i_M(G)z^T Dz.\end{aligned}$$

Consequently, $\alpha \geq i_M^2(G)/(2-\alpha)$. It follows that $\mu(\mathcal{L}) \geq 1 - \sqrt{1 - i_M^2(G)} = i_M^2(G)/(1 + \sqrt{1 - i_M^2(G)}) \geq i_M^2(G)/2$. □

5.4 The balancing flow generated by GDA

Usually, the flow generated by various nearest-neighbor schemes is characterized and analyzed indirectly, via its properties. Here, a direct explicit formula for the balancing flow computed by a GDA as a function of the generalized Laplacian and of the parameters of the assumed theoretical model is given.

Let M be an arbitrary GDM, a natural question that arise is what is the flow generated by the corresponding GDA and what are its properties? Let u the vector of size $q = |E(G)|$, such that $u_k = m_{ij}c_j$, for all $\{i,j\} \in E(G)$ and $U = \text{diag}(u)$. Because M satisfies the set of conditions (5.0.1), it can be easily verified that it can be put in the form $M = I - AUA^TD^{-1}$.

Lemma 6. *Let M be a GDM and $B = M - ce^T$. Then, we have:*

1. $B^n = M^n - ce^T$ *for all* $n \geq 1$;
2. $\rho(B) = \lambda_M$;
3. $(I-B)^{-1}$ *exists, with* e^T *and* c *as left and right eigenvectors corresponding to the eigenvalue* 1.

Proof.

1. For $n = 1$, by definition, the relation is true. Suppose that we have $B^n = M^n - ce^T$. Then $B^{n+1} = (M - ce^T)(M^n - ce^T) = M^{n+1} - ce^T$, because e^T and c are left, respectively right, eigenvectors of M, corresponding to the eigenvalue 1.
2. Let $\Lambda(A)$ denote the set of eigenvalues of the square matrix A. The proof relies on the fact that $\Lambda(B) = \Lambda(M) \setminus \{1\} \bigcup \{0\}$.
3. The matrix $I - B$ has the smallest eigenvalue $1 - \rho(B) = 1 - \lambda_M > 0$. Therefore $I - B$ is nonsingular. The second part relies on the fact that $e^TB = 0$ and $Bc = 0$.

□

As the algorithm 3 indicates, in each iteration step n, an amount

$$\delta_{ij}^{(n)} = m_{ji}l_i^{(n)} - m_{ij}l_j^{(n)}c_j = m_{ij}c_j\left(\frac{l_i^{(n)}}{c_i} - \frac{l_j^{(n)}}{c_j}\right)$$

is transferred across the edge $e_k = \{i,j\}$. This transfer is considered to take place from node i to node j if $\delta_{ij}^{(n)} \geq 0$ and in the opposite direction if $\delta_{ij}^{(n)} < 0$. The *migration flow generated at step n*

is the vector $x^{(n)} \in \mathbb{R}^q$ with the entries $x_k^{(n)} = \delta_{ij}^{(n)}$, for each edge e_k with the extremities i and j. In fact,

$$x_k^{(n)} = u_k \left(\frac{l_i^{(n)}}{c_i} - \frac{l_j^{(n)}}{c_j} \right), \forall\ e_k = \{i,j\} \in E(G).$$

In a condensed form, it can be rewritten as

$$x^{(n)} = UA^T D^{-1} l^{(n)}. \tag{5.4.1}$$

The *migration flow* generated by a GDA is then $f = \sum_{n\geq 0} x^{(n)}$. The following result identifies the balancing flow generated by a GDA:

Lemma 7. *The flow generated by a GDA has the expression*

$$f = UA^T D^{-1} (I - B)^{-1} l^{(0)}.$$

Proof. First, we notice that $UA^T D^{-1} ce^T l^{(0)} = 0$, because $A^T e = 0$. As $\rho(B) < 1$, $I - B$ is nonsingular and $(I - B)^{-1} = \sum_{n\geq 0} B^n$. Therefore, the migration flow can be expressed as

$$\begin{aligned} f &= \sum_{n\geq 0} x^{(n)} = \sum_{n\geq 0} UA^T D^{-1} l^{(n)} = \sum_{n\geq 0} UA^T D^{-1} M^n l^{(0)} \\ &= \sum_{n\geq 0} UA^T D^{-1} (B^n + ce^T) l^{(0)} = \sum_{n\geq 0} UA^T D^{-1} B^n l^{(0)} \\ &= UA^T D^{-1} (\sum_{n\geq 0} B^n) l^{(0)} = UA^T D^{-1} (I - B)^{-1} l^{(0)}. \end{aligned}$$

□

Theorem 13. *The migration flow generated by a GDA is a balancing flow.*

Proof. By lemma 6, e^T is a left eigenvalue of $(I - B)^{-1}$. Therefore,

$$\begin{aligned} Af &= AUA^T D^{-1} (I - M + ce^T)^{-1} l^{(0)} = (I - M)(I - M + ce^T)^{-1} l^{(0)} \\ &= l^{(0)} - ce^T (I - B)^{-1} l^{(0)} = l^{(0)} - ce^T l^{(0)} = l^{(0)} - \bar{l}. \end{aligned}$$

□

Lemma 8. *Let P be the matrix defined as*

$$P = U^{1/2} A^T D^{-1} (I - B)^{-1} A U^{1/2}.$$

Then P is a projection matrix.

Proof. Clearly, $P = P^T$. On the other hand, by Lemma 6, we have

$$\begin{aligned} P^2 &= U^{1/2}A^TD^{-1}(I-B)^{-1}AUA^TD^{-1}(I-B)^{-1}AU^{1/2} \\ &= U^{1/2}A^TD^{-1}(I-B)^{-1}(I-B-ce^T)(I-B)^{-1}AU^{1/2} \\ &= P - U^{1/2}A^TD^{-1}(I-B)^{-1}ce^T(I-B)^{-1}AU^{1/2} = P \end{aligned}$$

□

This allows us to formulate a more important result:

Theorem 14. *Let $\mathcal{F}(H)$ denote the set of all balancing flows in $H = (G, l, c, w)$, f the flow generated by an arbitrary GDA on H, and P the projection matrix defined above. Then*

$$f = U^{1/2}PU^{-1/2}\,x,\ \forall\, x \in \mathcal{F}(H).$$

Proof. Let x be an arbitrary balancing flow in H. Replacing $l^{(0)}$ by $A\,x + \bar{l}$ in the expression of f one gets

$$U^{-1/2}f = U^{1/2}A^TD^{-1}(I-B)^{-1}Ax.$$

Therefore, $f = U^{1/2}PU^{-1/2}\,x$. □

It is well known that in the homogeneous case the diffusion schemes generate a balancing flow that is minimal in the 2-norm [45, 15]. In the heterogeneous case, a weighted 2-norm is minimized [66, 18]. Here, we proved a more powerful result which states that the flow generated by a GDA is a scaled projection of all balancing flows. The minimality with respect to the norm $\|\cdot\|_{2,U^{-1/2}}$ of the flow generated by a GDA appears as a direct consequence of the above theorem (as $\|P\|_2 = 1$).

Let $f^{(n)} = \sum_{k=0}^{n-1} x^{(k)}$ be the partial migration flow generated after n steps by a GDA. Then

Lemma 9. *If f and $f^{(n)}$ are the total and, respectively, the partial migration flow after n iterations of a GDA, one has that*

$$\|f - f^{(n)}\|_2 \le \sqrt{1-\lambda_p}\,\sqrt{\frac{c_{\max}}{c_{\min}}}\,\frac{\lambda_M^n}{1-\lambda_M}\,\|l^{(0)} - \bar{l}\|_2. \tag{5.4.2}$$

Proof. First, we notice that $0 = WA^TD^{-1}(\sum_{k\geq 0} B^k)\bar{l}$, as $\bar{l} = c(e^T l^{(0)})$, $Bc = 0$, and $A^T e = 0$. Let $U = W^{1/2}A^TD^{-1/2}$ and $B' = D^{-1/2}BD^{1/2}$. Then

$$\begin{aligned} f - f^{(n)} &= W^{1/2}UD^{-1/2}(\sum_{k\geq 0} B^k)(l^{(0)} - \bar{l}) - W^{1/2}UD^{-1/2}(\sum_{k=0}^{n-1} B^k)(l^{(0)} - \bar{l}) \\ &= W^{1/2}U\sum_{k\geq 0}(B')^k(B')^nD^{-1/2}(l^{(0)} - \bar{l}) \\ &= W^{1/2}U(I - B')^{-1}(B')^nD^{-1/2}(l^{(0)} - \bar{l}). \end{aligned}$$

The norm of U is evaluated as follows:

$$\|U\|_2^2 = \max_{x\neq 0}\frac{\|Ux\|_2^2}{\|x\|_2^2} = \max_{x\neq 0}\frac{x^TD^{-1/2}AWA^TD^{-1/2}x}{x^Tx}$$

$$= \max_{x\neq 0}\frac{x^T(I - D^{-1/2}MD^{1/2})x}{x^Tx} = 1 - \lambda_p.$$

On the other hand, B and B' are similar and B' is real-symmetric. It follows, on the basis of Lemma 6 that $\|B'\|_2 = \lambda_M$ and that $(I - B')^{-1}$ exists, having the euclidian norm $1/(1 - \lambda_M)$. Therefore,

$$\|f - f^{(n)}\|_2 \leq \sqrt{1 - \lambda_p}\sqrt{\frac{c_{\max}}{c_{\min}}}\frac{\lambda_M^n}{1 - \lambda_M}\|l^{(0)} - \bar{l}\|_2.$$

□

5.5 Analysis of a family of generalized diffusion algorithms

We are interested in finding diffusion matrices with a small convergence factor. For the diffusion depending on a single parameter, optimal matrices have been found in the homogeneous case for common topologies as the n-dimensional meshes or the n-dimensional tori [93]. In the heterogeneous case, due to the fact that the capacities are different and the diffusion matrix is non-symmetric, finding optimal diffusion matrices is much more difficult, even for simple cases.

The question is how to define diffusion matrices with an improved convergence factor. Such matrices should take into account both the network and the processor characteristics. In the homogeneous case, a common practice is to consider either $m_{ij} = 1/(\Delta + 1)$ [13] or $m_{ij} = 1/(\max\{\delta_i, \delta_j\} + 1)$ [6], for all edges $\{i, j\} \in E(G)$. However, these approaches generally do not take into account neither the communication heterogeneity nor the processor heterogeneity, but efforts have been done in this

direction. Diekmann et al. tackled the problem in environments that are heterogeneous with respect to the communication costs [15, 45]. We generalized the diffusion algorithm proposed by Boillat for the case when only the processors have different speeds [66, 67]. Elsässer et al. extended the diffusion algorithm proposed by Cybenko to the case when the processors as well as the network have different performances [19].

We consider diffusion matrices of the type $M = I - ASWA^TD^{-1}$ with S a diagonal matrix of scalars such that $S = \mathrm{diag}(s)$, where s is a vector of size q. The generalized diffusion described by Elsässer et al. [18, 19] corresponds to the case when $S = \alpha I$ and is a generalization of the diffusion described by Cybenko in [13]. The best choice for α is usually considered $2/(\mu_2(\mathcal{L}) + \mu_p(\mathcal{L}))$. However, in a dynamic context in which the communication topology changes often so that the algorithm must be applied several times, these eigenvalues must be computed at runtime, which introduces a supplementary overhead.

When given a vector w of size q, representing weights associated with the edges of G and a vector c of size p, representing the array of capacities, the problem is how to choose a vector of scalars s, of size q, so that the conditions (5.0.1) are satisfied and M has a minimal convergence factor. We notice that, as c is a right eigenvector of M, one should have

$$c_i = \sum_{j \in \mathcal{N}_i} m_{ij} c_j \; = \sum_{e_k = \{i,j\}} s_k w_k. \tag{5.5.1}$$

Therefore, $s_{\min} \leq c_i/\delta_i^w$, for all $1 \leq i \leq p$. The bounds established in the previous section are improving when $\theta_{\min}(M)$ is larger. As the weights on the edges depend on the network communication parameters, s must be chosen so that (5.5.1) holds and s_{min} is as large as possible. For arbitrary $\epsilon \geq 0$ we consider for any edge e_k the quantity

$$s_k(\epsilon) = \min_{e_k = \{i,j\}} \left\{ \frac{c_i}{\delta_i^w + \epsilon}, \frac{c_j}{\delta_j^w + \epsilon} \right\}. \tag{5.5.2}$$

The matrix $M(\epsilon)$ defined as

$$m_{ij}(\epsilon) = \begin{cases} \frac{s_k(\epsilon)\, w_k}{c_j}, \text{ if } e_k = \{ij\} \in E, \\ 0, \text{ if } \{i,j\} \notin E, i \neq j, \\ 1 - \sum_{k:\{k,j\} \in E(G)} m_{kj}, \text{ if } i = j, \end{cases}$$

is a GDM.

This is clearly a generalization of the diffusion proposed by Boillat [6]. Indeed, in the homogeneous case one has $c_i = 1/p$ for all $1 \leq i \leq p$ and $w_k = 1$, for all $1 \leq k \leq q$. For $\epsilon = 1$, one gets

$$m_{ij} = \begin{cases} \min\left\{\frac{1}{\delta_i+1}, \frac{1}{\delta_j+1}\right\}, \text{ if } ij \in E, \\ 0, \text{ if } \{i,j\} \notin E, i \neq j, \\ 1 - \sum_{\{k,j\}\in E} m_{kj}(\epsilon), \text{ if } i = j, \end{cases}$$

which is exactly the diffusion considered in the above paper. For the generalized diffusion matrix $M(\epsilon)$ defined as above we have

$$\theta(M(\epsilon)) = \min_i\{m_{ij}(\epsilon)c_j\} = \min_k \left\{ \min_{e_k=\{i,j\}} \left\{ \frac{c_i}{\delta_i^w + \epsilon}, \frac{c_j}{\delta_j^w + \epsilon} \right\} w_k \right\}.$$

Clearly, $\theta(M) \geq c_{\min} w_{\min}/(\Delta^w + \epsilon)$. With these notations, using corollary 1, one gets

$$\lambda_2(M(\epsilon)) \leq 1 - 4\, e(G)\, \frac{w_{\min}}{\Delta^w + \epsilon}\, \frac{c_{\min}}{c_{\max}}\, \sin^2\left(\frac{\pi}{2p}\right). \tag{5.5.3}$$

Lemma 10. *For all $\epsilon \geq 0$, if $\lambda_p(M(\epsilon)) < 0$ then*

$$|\lambda_p(M(\epsilon)| \leq 1 - 2\frac{\epsilon}{\Delta^w + \epsilon}. \tag{5.5.4}$$

Proof. For simplicity, we write λ_p instead of $\lambda_p(M(\epsilon))$. It is well known [82] that all the eigenvalues of $M(\epsilon)$ lie in the union of the Gershgorin disks. Therefore, there is an index i such that

$$|\lambda_p - m(\epsilon)_{ii}| \leq \Lambda_i, \tag{5.5.5}$$

where $\Lambda_i = \sum_{j\neq i} m(\epsilon)_{ji}$. The matrix $M(\epsilon)$ being column stochastic and λ_p being negative, the above inequality is equivalent to $|\lambda_p| = -\lambda_p \leq 1 - 2m(\epsilon)_{ii} = -1 + 2(1 - m(\epsilon)_{ii})$. On the other hand, we have that

$$1 - m(\epsilon)_{ii} = \sum_{k:e_k=\{j,i\}\in E} \min\left\{ \frac{c_i}{\delta_i^w + \epsilon}, \frac{c_j}{\delta_j^w + \epsilon} \right\} \frac{w_{ij}}{c_j} \leq \frac{\delta_i^w}{\delta_i^w + \epsilon}.$$

Hence, it follows that

$$|\lambda_p| \leq 1 - 2\frac{\epsilon}{\delta_i^w + \epsilon} \leq 1 - 2\frac{\epsilon}{\Delta^w + \epsilon}.$$

□

Corollary 2. *Let $\epsilon_0 = 2e(G)w_{\min}(c_{\min}/c_{\max})\sin^2(\pi/(2p))$. For all $\epsilon \geq \epsilon_0$,*

$$\lambda_{M(\epsilon)} \leq 1 - 4\, e(G)\, \frac{w_{\min}}{\Delta^w + \epsilon}\, \frac{c_{\min}}{c_{\max}}\, \sin^2\left(\frac{\pi}{2p}\right). \tag{5.5.6}$$

Proof. The result follows directly from (5.5.3) and (5.5.4). □

The upper bound fixed by the above inequality is minimized when $\epsilon = \epsilon_0$. Therefore, the diffusion based on $M(\epsilon_0)$ may converge faster than an algorithm based on $M(\epsilon)$, with $\epsilon > \epsilon_0$.

Theorem 15. *The maximum number of iterations needed by the GDA based on $M(\epsilon_0)$ to balance the system belongs to $O((c_{\max}\,\Delta^w\,p^2)/(c_{\min}\,w_{\min}\,e(G))$.*

Proof. Suppose that we want to reduce the weighted 2-norm of the load imbalance to τ, sufficiently small so that the workload distribution, after a number of steps n, is close to the fair distribution. On the basis of (5.5.6), it is sufficient to take n so that

$$\|l^{(n)} - \bar{l}\|_{2,D^{-1/2}} \leq \left(1 - 4\,e(G)\,\frac{w_{\min}}{\Delta^w + \epsilon_0}\frac{c_{\min}}{c_{\max}}\sin^2\left(\frac{\pi}{2p}\right)\right)^n \|l^{(0)} - \bar{l}\|_{2,D^{-1/2}} \simeq \tau.$$

As for large p one has

$$\ln\left(1 - 2\epsilon_0/(\Delta^w + \epsilon_0)\right) \simeq 2\epsilon_0/(\Delta^w + \epsilon_0) \text{ and } \sin\left(\frac{\pi}{2p}\right) \simeq \frac{\pi}{2p},$$

we conclude that the maximum number of steps necessary to reduce the weighted 2-norm of the load imbalance to τ is

$$\begin{aligned} n &\simeq \ln\left(\frac{1}{\tau}\|l^{(0)} - \bar{l}\|_{2,D^{-1/2}}\right)\frac{1}{\pi^2\,e(G)}\frac{c_{\max}}{c_{\min}}\frac{\Delta^w + \epsilon_0}{w_{\min}}\,p^2 \\ &= \ln\left(\frac{1}{\tau}\|l^{(0)} - \bar{l}\|_{2,D^{-1/2}}\right)\frac{1}{\pi^2\,e(G)}\frac{c_{\max}}{c_{\min}}\frac{\Delta^w}{w_{\min}}\,p^2 + \frac{1}{2}\ln(\frac{1}{\tau}\|l^{(0)} - \bar{l}\|_{2,D^{-1/2}}). \end{aligned}$$

□

The above estimation is consistent with the result obtained by Boillat in the homogeneous case where it was shown that the diffusion takes $O(p^2)$ communication steps [6]. In the homogeneous case, when $c_i = c_j$ for all $i \neq j$ and $w_k = 1$, for all $1 \leq k \leq q$, the diffusion matrix proposed by Boillat can be expressed in our terms as $M(1)$. Particularizing the results obtained above, a homogeneous diffusion based on $M(\epsilon_0)$ may converge faster than the algorithm given in the cited paper, as ϵ_0 improves the bound for the convergence factor. This is confirmed in part by the theoretical results obtained by Xu et al. [93] and by the tests we have done.

5.6 Examples, comparisons, experimental results

We show that the complexity bounds formulated in the previous section are tight in the sense that, in the homogeneous case, for some common topologies, $M(\epsilon_0)$ is close to the optimum.

Let us consider now two general classes of graphs: the n-dimensional tori and the n-dimensional meshes.

We shall denote $M^\star = M^1(\epsilon_0)$, with M^1 defined as in the equation 5.6.1.

A) The n-dimensional tori.

A n-dimensional torus, $T_{k_1,k_2,\ldots,k_n}$ is the Cartesian product of n cycles, i.e. $T_{k_1,k_2,\ldots,k_n} = C_{k_1} \times C_{k_2} \times \ldots \times C_{k_n}$, where C_{k_i} is a cycle with k_i vertices. For this type of graphs we have $\Delta = 2n$.

In a homogeneous context, if L is the corresponding unweighted Laplacian, one has that $M^\star(T_{k_1,k_2,\ldots,k_n}) = I - 1/(\Delta + \epsilon_0)L$ and

$$\lambda_{M^\star} = 1 - \frac{4}{\Delta + \epsilon_0} \sin^2\left(\frac{\pi}{k}\right),$$

with $k = \max_{i=1,n} k_i$. Also, $e(T_{k_1,k_2,\ldots,k_n}) = \Delta$ and $\epsilon_0 = 2\Delta \sin^2(\pi/(2\prod_{i=1}^n k_i))$. Therefore, in the homogeneous case, the inequality (5.2.7) reads as

$$\lambda_{M^\star} \leq 1 - \frac{4\Delta}{\Delta + \epsilon_0} \sin^2\left(\frac{\pi}{2\prod_{i=1}^n k_i}\right).$$

The given bound is tight for large k, namely in the case of a chain ($n = 1$ and $k = p$).

For the particular case of a ring, Xu et al. [93] found optimal diffusion matrices that, in our notation, can be expressed as $M_{opt} = M^\star(\epsilon_{opt})$ with

$$\epsilon_{opt} = \begin{cases} 2\sin^2(\frac{\pi}{p}) & \text{if } p \text{ is even} \\ 2\sin(\frac{\pi}{2p})\sin(3\frac{\pi}{2p}) & \text{if } p \text{ is odd.} \end{cases}$$

We see that in this case $\epsilon_0 = 4\sin^2(\pi/(2p))$ is very close to ϵ_{opt}(they are in the class $\Theta(1/p^2)$), compared to the situation when $\epsilon_0 = 1$. Generally, for an n-dimensional torus $T_{k_1,k_2,\ldots,k_n}$ the optimal matrices found by the same authors can be put in the form $M_{opt} = M^\star(\epsilon_{opt})$, with $\epsilon_{opt} = 2\sin^2(\pi/k)$ when all k_i's are larger than 2 and all even and $\epsilon_{opt} = \min\{0, \sin^2(\pi/k) - \sum_{i=1,n} \sin^2(\pi/(2k_i))\}$ when all k_i's are odd.

B) The n-dimensional meshes.

A n-dimensional mesh, $M_{k_1,k_2,\ldots,k_n}$ is the Cartesian product of n paths, i.e. $M_{k_1,k_2,\ldots,k_n} = P_{k_1} \times P_{k_2} \times \ldots \times P_{k_n}$, where P_{k_i} is a path with k_i vertices. For this type of graph we have $\Delta = 2n$ and $e(M_{k_1,k_2,\ldots,k_n}) = n$. The analysis for this class of graphs can be pursued in an similar manner. When the communication costs are similar, as well as the computational capacities, $m^{\star}_{ij} = 1/(\max\{d_i, d_j\} + \epsilon_0)$ for every edge (i,j) and for a chain of order p, $M^{\star} = I - 1/(2+\epsilon_0)L$, L being the corresponding unweighted Laplacian. Therefore

$$\lambda_{M^{\star}} = 1 - 4\frac{1}{2+\epsilon_0}\sin^2\left(\frac{\pi}{2p}\right).$$

On the other hand, in this case the inequality (5.2.7) reads as

$$\lambda_{M^{\star}} \leq 1 - 4\frac{1}{2+\epsilon_0}\sin^2\left(\frac{\pi}{2p}\right).$$

In this case, the upper bound in the inequality (5.2.7) gives the exact value of the convergence factor of the diffusion in a chain. In the case of n-dimensional meshes, Xu et. al. [93] found optimal matrices, for the diffusion with a single parameter, which are identical to $M^1(0)$ in all the cases.

However, when the capacities and the communication parameters differ, it is difficult to give analytic characterizations for the convergence factor, even for simple cases. From this point of view, the general bounds obtained in the section 5.2 are useful.

We simulated and tested various cases of heterogeneous diffusion. In a first step, we considered different artificially generated topologies, capacities, communication weights and initial loads. The

following diffusion types were used

$$m_{ij}^1(\epsilon) = \begin{cases} \min\left\{\frac{c_i}{\delta_i^w+\epsilon}\frac{c_j}{\delta_j^w+\epsilon}\right\}\frac{w_k}{c_j}, \text{ if } e_k=\{ij\}\in E, \\ 0, \text{ if } \{j,i\}\notin E(G), i\neq j, \\ 1-\sum_{j:\{j,i\}\in E(G)} m_{ji}, \text{ if } i=j. \end{cases} \tag{5.6.1}$$

$$m_{ij}^2(\epsilon) = \begin{cases} \frac{c_{\min}}{c_j}\frac{w_k}{\Delta^w+\epsilon}, \text{ if } e_k=\{ij\}\in E, \\ 0, \text{ if } \{j,i\}\notin E(G), i\neq j, \\ 1-\sum_{j:\{j,i\}\in E(G)} m_{ji}, \text{ if } i=j. \end{cases} \tag{5.6.2}$$

$$m_{ij}^3(\epsilon) = \begin{cases} \sqrt{\frac{c_{\min}}{c_{\max}}}\sqrt{\frac{c_i}{c_j}}\frac{w_k}{\Delta^w+\epsilon}, \text{if } e_k=\{ij\}\in E, \\ 0, \text{ if } \{j,i\}\notin E(G), i\neq j, \\ 1-\sum_{j:\{j,i\}\in E(G)} m_{ji}, \text{ if } i=j. \end{cases} \tag{5.6.3}$$

$$m_{ij}^4(\epsilon) = \begin{cases} \frac{c_i}{c_{\max}}\frac{w_k}{\Delta^w+\epsilon}, \text{ if } e_k=\{ij\}\in E, \\ 0, \text{ if } \{j,i\}\notin E(G), i\neq j, \\ 1-\sum_{j:\{j,i\}\in E(G)} m_{ji}, \text{ if } i=j. \end{cases} \tag{5.6.4}$$

$$m_{ij}^5 = \begin{cases} c_i, \text{ if } e_k=\{ij\}\in E, \\ 0, \text{ if } \{j,i\}\notin E(G), i\neq j, \\ 1-\sum_{j:\{j,i\}\in E(G)} m_{ji}, \text{ if } i=j. \end{cases} \tag{5.6.5}$$

Additionally, we considered

$$m_{ij}^6 = \begin{cases} \alpha\frac{w_k}{c_j}, \text{ if } e_k=\{ij\}\in E, \\ 0, \text{ if } \{j,i\}\notin E(G), i\neq j, \\ 1-\sum_{j:\{j,i\}\in E(G)} m_{ji}, \text{ if } i=j, \end{cases} \tag{5.6.6}$$

with $\alpha = 2/(\mu_2(\mathcal{L})+\mu_p(\mathcal{L}))$

The tables 5.6 and 5.6 report results for the case when the ratio $\alpha = c_{\max}/c_{\min} \in \{2,5,10\}$, $w_k = 1, k = 1, |E|$ and $\epsilon \in \{1, \epsilon_0\}$. The N_i represent the number of iterations performed until $\|l^{(n)} - \bar{l}\|_{2,D^{-1/2}} < 1$ by the diffusion algorithms defined by the relations (5.6.1) to (5.6.5).

Tests performed for different intermediate values of ϵ and other different communication topologies showed that in each case the diffusion $M^1(\epsilon_0)$ needed the fewest number of local communication steps.

Clearly, the convergence of any GDA is slower when the number of processors is large, when the ratio between the maximum capacity and the minimum capacity is large and when the underlying graph has poor connectivity. The diffusion defined by (5.6.5) shows how important is in practice the choice of a diffusion matrix; it serves as a very bad example.

Table 5.1: Number of steps until convergence of a GDA on binary trees with 15, 31 and 63 nodes

p	α	ϵ	N1	N2	N3	N4	N5
15	2	1	496	507	653	864	2275
		ϵ_0	371	380	489	648	2275
	5	1	781	943	1364	2339	4330
		ϵ_0	568	707	1024	1756	4330
	10	1	1325	1646	2912	6784	10517
		ϵ_0	963	1235	2186	5095	10517
31	2	1	1117	1138	1435	1873	9847
		ϵ_0	837	853	1075	1404	9847
	5	1	1483	1538	2921	5871	18206
		ϵ_0	1111	1153	2191	4404	18206
	10	1	1623	2307	2450	4395	11761
		ϵ_0	1119	1730	1837	3296	11761
63	2	1	2127	2196	2659	3328	34974
		ϵ_0	1595	1646	1994	2496	34974
	5	1	3044	3588	5203	8734	50675
		ϵ_0	2224	2691	3902	6550	50675
	10	1	4853	5505	11482	27410	116505
		ϵ_0	3549	4128	8612	20558	116505

Table 5.2: Number of steps until convergence of a GDA on paths with 16, 32 and 64 nodes

p	α	ϵ	N1	N2	N3	N4	N5
16	2	1	804	826	889	942	3461
		ϵ_0	536	551	593	628	3461
	5	1	931	1119	1349	1918	4350
		ϵ_0	621	746	900	1280	4350
	10	1	1439	2064	2259	2677	5356
		ϵ_0	960	1378	1508	1787	5356
32	2	1	3819	3876	4551	5542	39744
		ϵ_0	2546	2584	3034	3695	39744
	5	1	4754	5133	7076	10924	45896
		ϵ_0	3170	3423	4718	7285	45896
	10	1	6132	7681	11527	21755	71800
		ϵ_0	4089	5122	7687	14509	71800
64	2	1	12825	13158	15112	18015	255258
		ϵ_0	8550	8772	10075	12010	255258
	5	1	13591	14157	23488	41731	269877
		ϵ_0	9061	9438	15659	27823	269877
	10	1	13767	14401	30294	69426	266141
		ϵ_0	9179	9601	20197	46289	266141

When M is of the form $M = I - ASWA^TD^{-1}$, with $S = \alpha\, I$, the best choice suggested for α is $2/(\mu_2(\mathcal{L}) + \mu_2(\mathcal{L}))$, when $\mathcal{L} = D^{-1/2}AWA^TD^{-1/2}$ [15, 18]. We shall remark however that this choice does not guarantee that M is always nonnegative, as the following counterexample proves. Indeed, for a graph of type star with 4 vertices, in the simple case, $w_k = 1$, $k = 1, 3$ and $c_i = 1/4$, $i = 1, 4$. $\mathcal{L}$ has the form indicated below and its eigenvalues are $\{0, 4, 4, 16\}$. Therefore, $\alpha = 1/10$ and M is as indicated below:

$$\mathcal{L} = \begin{bmatrix} 12 & -4 & -4 & -4 \\ -4 & 4 & 0 & 0 \\ -4 & 0 & 4 & 0 \\ -4 & 0 & 0 & 4 \end{bmatrix}, \quad M = \begin{bmatrix} -0.2 & 0.4 & 0.4 & 0.4 \\ 0.4 & 0.6 & 0.0 & 0.0 \\ 0.4 & 0.0 & 0.6 & 0.0 \\ 0.4 & 0.0 & 0.0 & 0.6 \end{bmatrix}$$

We proceeded to a comparison between the generalized diffusion algorithms that use the GDM's $M^1(1)$, $M^1(\epsilon_0)$ and M^6. We refer to these algorithms as GDA1, GDA0 and GDA6, respectively.

It is well known that the diffusion algorithms suffer of slow convergence on graphs of type path. For such topologies, when choosing ϵ_0 as indicated in the previous sections, i.e. $\epsilon_0 = 2e(G)w_{min}\,(c_{min})/(c_{max})\,\sin^2(\pi/(2p))$, the convergence factors of $M^1(\epsilon_0)$ and M^6 are very close. In

Table 5.6, the convergence factors of GDA6, GDA0 and GDA1 are shown, on paths of sizes varying between 53 and 60, in a heterogeneous environment in which the speeds were set to $r \mod 4 + 1$, where r is the rank of a processor, and the edge weights were set to $k \mod 2 + 1$, k being the index of an edge. The table 5.6 shows that, in this context, GDA0 has a better convergence factor than GDA1 in all the cases and that the convergence factor of GDA0 is close to that of GDA6. The differences are higher between GDA0 and GDA1 than those between GDA0 and GDA6.

Tests were pursued with the three algorithms in order to compare the number of iterations until convergence, for communication topologies of type path and binary tree, both in homogeneous and heterogeneous contexts. We considered two kinds of load distributions:

- HIGH: all processors have the same load except one whose load is 10 times greater than of the others;
- HALF: the processors are divided into two subsets, the processors in each subset have the same load but the ratio between the loads of two processors in different subsets is 1 to 5.

Regarding the heterogeneity of the computational environment, three test cases were considered:

- HOMO: the system is homogeneous both w.r.t the processors' performances and the communication costs;
- HCUW: heterogeneous environment characterized by non-uniform speeds and uniform communication costs. More specifically,
  - the speed of a processor was considered proportional to $r \mod 4+1$, r denoting the rank of a processor;
  - the weights on the edges were set to 1.
- HCHW: heterogeneous environment characterized by non-uniform speeds and non-uniform communication costs. More specifically,
  - the speed of a processor was considered proportional to $r \mod 4+1$, r denoting the rank of a processor;
  - the weights on the edges were set to $k \mod 4 + 1$, k denoting the index of an edge.

Table 5.3: Convergence factors of GDA6, GDA0 and GDA1 on paths of size p, with $54 \leq p \leq 60$

GDA6	0.998885	0.998935	0.998996	0.999008	0.999036	0.999075	0.999123
GDA0	0.999017	0.99906	0.999112	0.999125	0.999149	0.999184	0.999227
GDA1	0.999344	0.999373	0.999408	0.999416	0.999432	0.999456	0.999484

The used convergence criterion was $\|l^{(n)} - \overline{l}\|_2 < 1$. With these settings, the number of iterations of the three algorithms for the considered communication topologies, computational environments and load distributions are as shown in the figures 5.1-5.6.

The tests showed that for a communication topology of type path and for the indicated load distributions, the algorithms GDA0 and GDA6 perform similarly in a homogeneous environment, as the figures 5.1 and 5.2 illustrate. Also, the same figures show that GDA0 performs much better than GDA1, which means that ϵ_0 is a better choice than $\epsilon = 1$ (proposed by Boillat). In the heterogeneous case, for the same type of communication topology and load distributions it was observed that GDA6 performs better than GDA0, but the differences are not very important when compared to GDA1, as shown in the figures 5.3-5.6 (left hand side). In the same context, GDA0 needed a significant fewer number of iterations than GDA1.

For a communication topology of type partial binary tree, all the considered diffusion algorithms performed faster than in the case of a path. This is motivated partly by the Theorem 11, which establishes an upper bound for the second largest eigenvalue of an arbitrary generalized diffusion matrix. Conforming to this, a GDA may converge faster on graphs with small diameters than on graphs with large diameters. A GDA takes the largest number of steps to converge on paths because they have the largest diameter among all the graphs with the same number of vertices. It was also observed that for a topology of type partial binary tree, in all the considered cases GDA0 performed better than GDA1, as expected. Also, GDA6 performed worse than GDA0 and even than GDA1, as shown in the figures 5.1-5.6(right hand side). The differences were significant especially in the case of graphs with a large number of vertices. A possible explanation could be that GDA6 uses a single scalar, while the other two algorithms based on $M^1(\epsilon_0)$ and $M^1(1)$ use multiple scalars (vectors of scalars). The restriction imposed by GDA6, that all scalars are identical, may be the cause of these differences. Another explanation could be that the accuracy of the estimations of the eigenvalues, and therefore of the scalar α, may have an important impact to the convergence rate when the diffusion matrices have large dimensions.

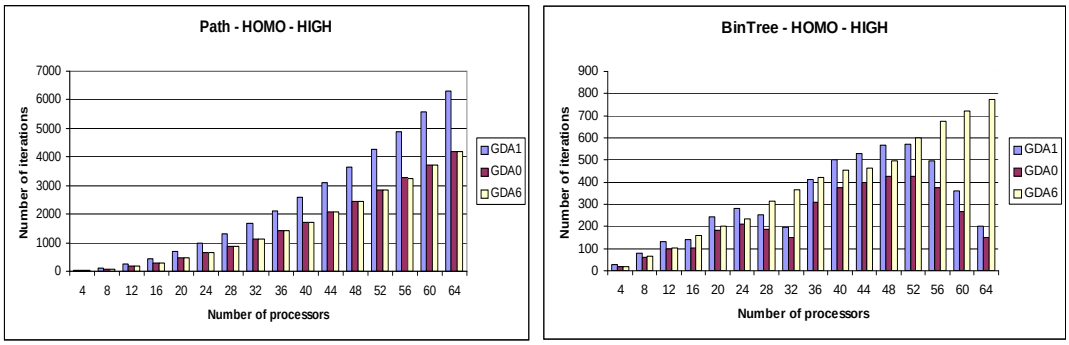


Figure 5.1: Number of iterations of GDA0, GDA1 and GDA6 for a homogeneous environment, on paths with with up to 64 vertices (left) and partial binary trees with up to 64 vertices (right) and a load distribution of type HIGH

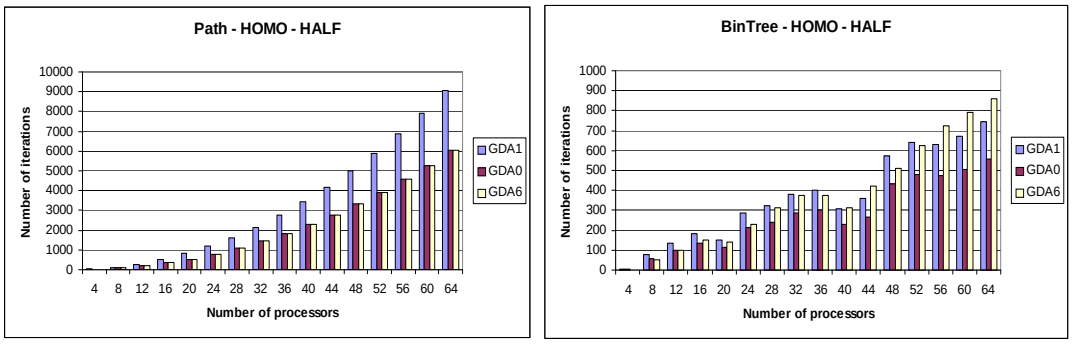


Figure 5.2: Number of iterations of GDA0, GDA1 and GDA6 for a homogeneous environment, on paths with with up to 64 vertices (left) and partial binary trees with up to 64 vertices (right) and a load distribution of type HALF

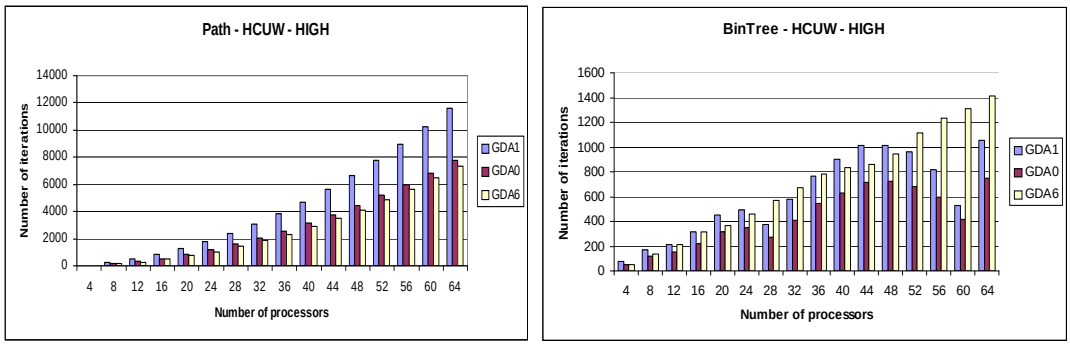


Figure 5.3: Number of iterations of GDA0, GDA1 and GDA6 for a heterogeneous environment (HCUW), on paths with with up to 64 vertices (left) and partial binary trees with up to 64 vertices (right) and a load distribution of type HIGH

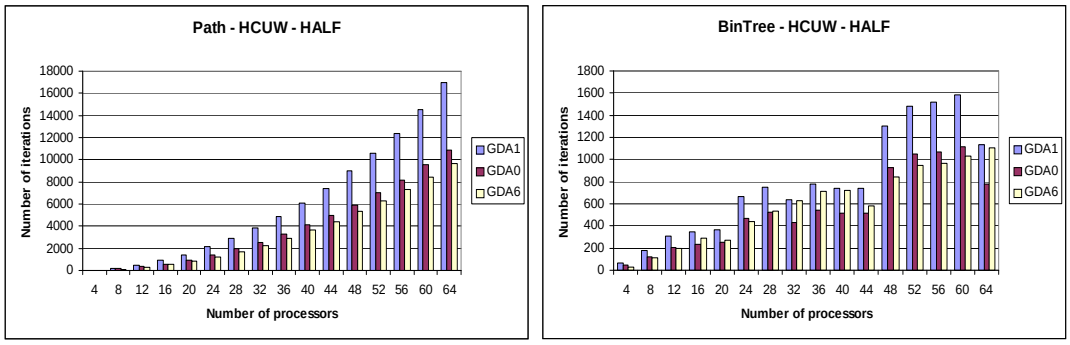


Figure 5.4: Number of iterations of GDA0, GDA1 and GDA6 for a heterogeneous environment (HCUW), on paths with with up to 64 vertices (left) and partial binary trees with up to 64 vertices (right) and a load distribution of type HALF

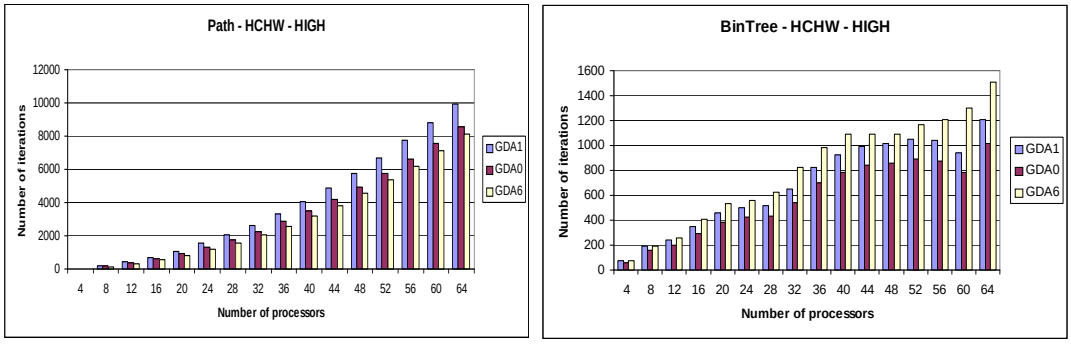


Figure 5.5: Number of iterations of GDA0, GDA1 and GDA6 for a heterogeneous environment (HCHW), on paths with with up to 64 vertices (left) and partial binary trees with up to 64 vertices (right) and a load distribution of type HIGH

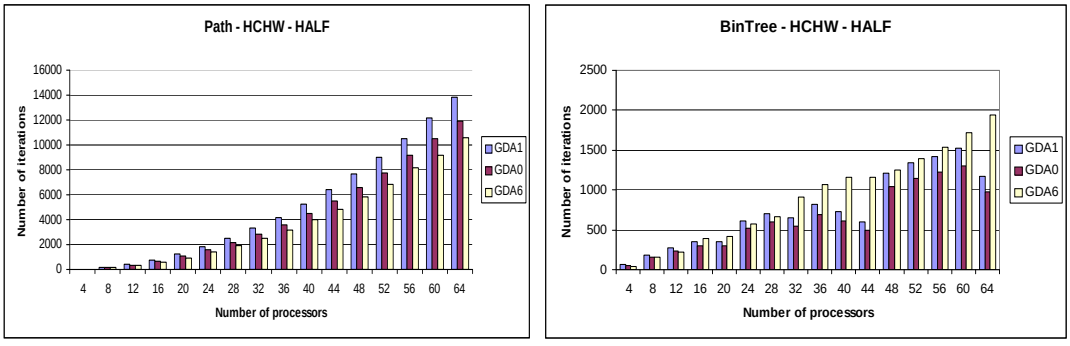


Figure 5.6: Number of iterations of GDA0, GDA1 and GDA6 for a heterogeneous environment (HCHW), on paths with with up to 64 vertices (left) and partial binary trees with up to 64 vertices (right) and a load distribution of type HALF

Table 5.4: Heterogeneous computing environment used for performing tests.

Machine name	OS	processor type	number	CPU	RAM
PC-151 .. PC-165	Solaris 2.7	i386	16	166 Mhz	32 M
peridot	Solaris 2.7	sparcv9	1	167 MHz	64 M
jargon	Solaris 2.7	sparc	1	110 MHz	64 M
outremer	Solaris 2.7	sparc	1	110 MHz	64 M
zircon	Solaris 2.8	sparc	1	70 MHz	32 M
tourmaline	Solaris 2.8	sparc	1	60 MHz	32 M
girasol	Solaris 2.8	sparc	1	75 MHz	32 M

We performed experiments on a heterogeneous environment built using commodity hardware available in our institute. This consists in a collection of heterogeneous machines with the characteristics given in the table 5.4. On these machines LAM/MPI [40] and Network Weather Service 2.0.6 (NWS) [92] were installed. NWS provided online values for network and processor parameters (the available CPU fraction, latency and TCP bandwidth). It should be noted that the differences are quite important and cannot be ignored. In our case it was observed that the bandwidth ranges between 3.5 and 17.8 and the normalized relative speeds between 0.036 to 0.120.

Tests were also performed with various load distribution and communication topologies, either artificially generated (path, ring, star) or corresponding to subdomain graphs of meshes usually used in engineering fields like Computational Fluid Dynamics (whitaker, airfoil, grid20x19 and shock). The diffusion algorithms defined by equations (5.6.1) to (5.6.5) were used. The tables 5.5 and 5.6 report the execution times (expressed in seconds) and the number of number of load index exchange phases for a highly imbalanced load distribution across the machines given in the table 5.4. The tolerated convergence error was set to 0.01.

5.7 Summary and Conclusions

In this chapter we dealt with aspects of the generalization of diffusion algorithms for dynamic load balancing in heterogeneous environments. Such algorithms may constitute a good approach because they are distributed, they require only local synchronization and because the distance to the fair distribution decreases after each iteration step. Globally, these algorithms result in a geometrical reduction of the load imbalance.

Table 5.5: Execution times in hundredths of seconds for different communication topologies on the heterogeneous environment described in 5.4.

Topology	$M^1(\epsilon_0)$	$M^2(\epsilon_0)$	$M^3(\epsilon_0)$	$M^4(\epsilon_0)$	M^5
Path	19.82	20.41	36.48	66.63	300.24
Ring	6.69	6.25	11.5	18.66	80.06
Star	8.69	8.96	9.60	9.93	4.00
SD_whitaker	2.40	2.60	5.90	10.46	13.14
SD_airfoil	3.55	6.02	9.60	16.26	20.45
SD_grid20x19	2.05	2.90	4.96	5.80	6.11
SD_shock	3.96	6.85	8.66	11.21	11.82

Table 5.6: Number of iterations until convergence on the heterogeneous environment described in 5.4.

Topology	$M^1(\epsilon_0)$	$M^2(\epsilon_0)$	$M^3(\epsilon_0)$	$M^4(\epsilon_0)$	M^5
Path	1828	1950	3284	5668	23898
Ring	512	521	869	1536	6390
Star	961	983	995	1086	425
SD_whitaker	221	258	562	998	1183
SD_airfoil	386	650	1003	1638	1936
SD_grid20x19	141	213	296	431	508
SD_shock	313	521	660	856	885

The balancing flow generated by the diffusion algorithms is usually approximated iteratively by summing fractions of load differences that should be exchanged at each step. We showed that they converge to a unique balancing flow in a number of steps that is dependent on the second smallest eigenvalue in absolute value, too. We gave an explicit formula for this unique flow as a function of the diffusion matrix and the vector of workloads and we showed that it has an important property: it is the projection of any other balancing flow. This result is more general than the minimality with respect to a weighted 2-norm, as the latter result is a natural consequence of the former.

We proposed and analyzed a generalization of the variant of diffusion proposed by Boillat [6]. General bounds were given for the second largest eigenvalue of a generalized diffusion matrix. These bounds allowed us to estimate the maximum number of steps that such an iterative process takes to balance the system.

It was illustrated that for critical communication topologies (the path for example) the proposed algorithm performs better than the original one proposed by Boillat. Generally, the tests performed with several types of diffusion algorithms in a heterogeneous environment showed that the algorithm

that we propose performs better on two dimensions: for different ϵ parameters and for different maximal capacity over minimal capacity ratios. Finally, when compared to the algorithm based on M^6 [19], which appears as a generalization of the variant of diffusion proposed by Cybenko [13], it was shown that for critical topologies (for which the convergence is the slowest) the convergence factors do not differ too much. The algorithm that we propose has the advantage that in a dynamic context it can be applied straightforward, while the other one needs the computation at runtime of the eigenvalues of the generalized Laplacian, which means a supplementary overhead.

Chapter 6

Comparison with the hydrodynamic algorithm

A hydrodynamic approach attempting to solve the load balancing problem in heterogeneous systems was proposed by Hui and Chanson [47, 48]. The computational environment was considered heterogeneous only with respect to the processor performances. They viewed the computing nodes as cylinders of different capacities that communicate through pipes with uniform characteristics. The liquid flow through a pipe represents a load transfer between processors. The key element that was used in the analysis of the time needed for reaching the equilibrium state was the concept of *global potential energy*. Denoting the workload of a node n_i by l_i and its capacity by c_i, the authors defined the *height* of a node n_i as $h_i = l_i/c_i$, *the potential energy of the liquid column* in n_i as $\mathrm{PE}(n_i) = 1/2 \cdot c_i h_i^2$, and the global potential energy as $\mathrm{GPE}(G) = \sum_{n_i \in V} \mathrm{PE}(n_i)$. They showed that the state of fairness is achieved when $\mathrm{GPE}(G)$ is minimal. If $\mathrm{GPE}^{(n)}$ denotes the global potential energy of the system after the n-th step and $\mathrm{GPE}^{\min}$ the minimum global potential energy corresponding to the equilibrium state, the following relation takes place:

$$\mathrm{GPE}^{(n)} < \eta^n \mathrm{GPE}^{(0)} + (1 - \eta^n)\mathrm{GPE}^{\min}, \tag{6.0.1}$$

where

$$\eta = 1 - \frac{1}{2(p-1)\mathrm{diam}(G)^2} \frac{c_{\min}^2}{c_{\max} \sum_i c_i}$$

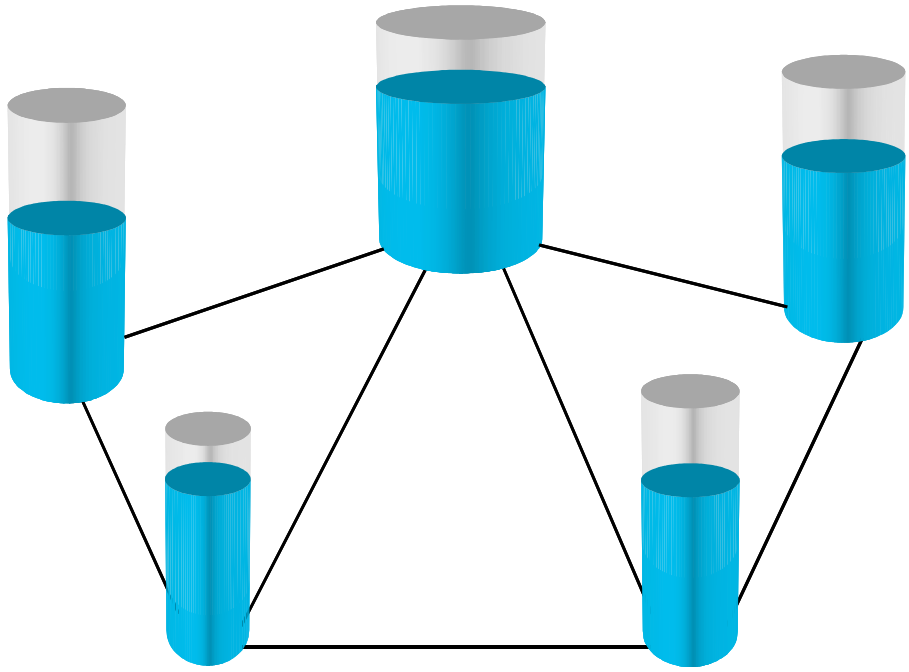


Figure 6.1: Hydrodynamic system

for all $n \geq 0$. The variable η indicates the convergence rate.

Our key observation is that the difference $\mathrm{GPE}^{(n)} - \mathrm{GPE}^{\min}$ is nothing but the square of the distance induced by the weighted 2-norm, between the workload vector corresponding to the current state at the n-th step and the workload vector in the fairness state. Indeed, the quantities $\mathrm{GPE}^{(n)}$ and $\mathrm{GPE}^{\min}$ can be rewritten as

$$\mathrm{GPE}^{(n)} = \frac{1}{2}\sum_i \frac{(l_i^{(n)})^2}{c_i}, \qquad \mathrm{GPE}^{\min} = \frac{1}{2}\sum_i \frac{\overline{l}_i^{\,2}}{c_i}$$

On the other hand, we have

$$\|l^{(n)} - \overline{l}\|^2_{D^{-1/2}} = \sum_i \frac{(l_i^{(n)} - \overline{l}_i)^2}{c_i} = \sum_i \frac{(l_i^{(n)})^2}{c_i} + \sum_i \frac{\overline{l}_i^{\,2}}{c_i} - 2\sum_i \frac{l_i^{(n)}\overline{l}_i}{c_i},$$

and

$$\sum_i \frac{l_i^{(n)}\overline{l}_i}{c_i} = \sum_i l_i^{(n)} \sum_j l_j^{(0)} = (\sum_j l_j^{(0)})^2 =$$

$$= \sum_i c_i (\sum_j l_j^{(0)})^2 = \sum_i \frac{c_i^2 (\sum_j l_j^{(0)})^2}{c_i} = \sum_i \frac{\overline{l}_i^{\,2}}{c_i}.$$

It follows that

$$\|l^{(n)} - \overline{l}\|^2_{D^{-1/2}} = \sum_i \frac{(l_i^{(n)})^2}{c_i} - \sum_i \frac{\overline{l}_i{}^2}{c_i} = 2(\mathrm{GPE}^{(n)} - \mathrm{GPE}^{\min}).$$

Therefore, the inequality (6.0.1) can be rewritten as

$$\|l^{(n)} - \overline{l}\|^2_{D^{-1/2}} < \eta^n \|l^{(0)} - \overline{l}\|^2_{D^{-1/2}}. \tag{6.0.2}$$

On the other hand, as it was shown in section 5.1, the generalized diffusion algorithm satisfies the relation

$$\|l^{(n)} - \overline{l}\|^2_{D^{-1/2}} \leq \lambda_M^{2n} \|l^{(0)} - \overline{l}\|^2_{D^{-1/2}}. \tag{6.0.3}$$

We show that a generalized diffusion algorithm has a better convergence factor and consequently, that it possibly converges faster.

Theorem 16. *In a heterogeneous environment $H = (G, l, c, e)$, a generalized diffusion based on $1/2\ (I + M(\epsilon_0))$ has a better convergence factor than the hydrodynamic algorithm.*

Proof. The communication was assumed to be uniform, so $w_i = 1$, for all i. Therefore, $\delta_i^w = \delta_i$ for all i and $\Delta^w = \Delta$. The reason for considering $M' = 1/2\ (I + M(\epsilon_0))$ is that this one has only nonnegative eigenvalues and in this case $\lambda_{M'} = \lambda_2(M') = 1/2\ (1 + \lambda_{M(\epsilon_0)})$. We shall show that the convergence factor of a diffusion based on $1/2\ (I + M(\epsilon_0))$ is always smaller than η. For this purpose we use theorem 11. We therefore have

$$\lambda_{M(\epsilon_0)} \leq 1 - \frac{c_{\min}}{\mathrm{diam}(G)\,(\Delta + \epsilon_0)}. \tag{6.0.4}$$

For proving that $\lambda_{M'} \leq \eta$ it is sufficient to show that

$$1 - \frac{1}{2}\,\frac{c_{\min}}{\mathrm{diam}(G)\,(\Delta + \epsilon_0)} \leq 1 - \frac{1}{2(p-1)\mathrm{diam}(G)^2}\,\frac{c_{\min}^2}{c_{\max}\sum_i c_i}.$$

As we consider normalized capacities, $\sum_i c_i = 1$. The last inequality is obviously true since $c_{\max} \geq c_{\min}$, and $(p-1)\mathrm{diam}(G) \geq \Delta + \epsilon_0$.

Therefore, we have $\lambda_{M'}^2 < \lambda_{M'} < \eta$. Looking at (6.0.2) and (6.0.3) we conclude that the GDA potentially converges faster than the hydrodynamic algorithm as its convergence factor is smaller. □

Chapter 7

Polynomial iterative schemes

7.1 Heterogeneous polynomial iterative schemes

As it was seen in the previous chapters, the diffusion algorithm converges rather slowly on some classes of communication topologies with low connectivity. Some efforts have been done to improve it. A natural idea emerging from the matrix iterative analysis [82] is to use polynomial acceleration methods [62]. This leads to homogeneous polynomial diffusion schemes. The key idea behind these schemes is that if M denotes a generalized diffusion matrix, one could find polynomials of M with a smaller second largest eigenvalue in absolute value than that of M. In the heterogeneous case, one can naturally extend such schemes as follows:

$$l^{(n)} = P_n(M)\, l^{(0)}, \tag{7.1.1}$$

with $M = I - D^{1/2}\mathcal{L}D^{-1/2}$ and P_n a polynomial of degree n, $P_n(x) = \sum_{k=0}^{n} a_k x^{n-k}$, with $P_n(1) = 1$. The last condition guarantees the conservation of the load during the load balancing process.

7.1.1 The balancing flow generated by the heterogeneous polynomial schemes

In this subsection we investigate the flow generated by a scheme of type (7.1.1).

Let us consider B as defined in lemma 6, i.e. $B = M - ce^T$. It can be easily verified that

equation (7.1.1) can be put in the following equivalent form

$$l^{(n)} - \overline{l} = P_n(B)(l^{(0)} - \overline{l}). \tag{7.1.2}$$

On the other hand, because 1 is a root of the polynomial $R_n(x) = P_n(x) - P_n(1)$, there is a polynomial Q_{n-1} of degree $n-1$ such that $R_n(x) = (x-1)Q_{n-1}(x)$. Therefore, P_n must be of the form

$$P_n(x) = 1 + (x-1)Q_{n-1}(x). \tag{7.1.3}$$

The next theorem expresses the migration flow generated by the heterogeneous polynomial schemes:

Theorem 17. *Let $H = (G, l, c, w)$ be a heterogeneous model, $M = I - AUA^T D^{-1}$ a GDM (with $U = \mathrm{diag}(u)$) and P_n an arbitrary polynomial such that $P_n(1) = 1$. Then, the balancing flow generated by a scheme of type* (7.1.1) *is*

$$f = UA^T D^{-1}(I - B)^{-1} l^{(0)}, \tag{7.1.4}$$

with $B = M - ce^T$.

Proof. Indeed, equation (7.1.2) can be put in the following equivalent form:

$$\begin{aligned} l^{(n)} &= l^{(0)} - (I - B)Q_{n-1}(B)(l^{(0)} - \overline{l}) \\ &= l^{(0)} - (I - M)Q_{n-1}(B)(l^{(0)} - \overline{l}) \\ &= l^{(0)} - AUA^T D^{-1} Q_{n-1}(B)(l^{(0)} - \overline{l}). \end{aligned}$$

The partial flow generated after the n-th iteration is

$$f^{(n)} = UA^T D^{-1} Q_{n-1}(B)(l^{(0)} - \overline{l}). \tag{7.1.5}$$

On the other hand one has

$$\begin{aligned} l^{(0)} - l^{(n)} &= (I - P_n(B))(l^{(0)} - \overline{l}) \\ &= (I - B)Q_{n-1}(B)(l^{(0)} - \overline{l}). \end{aligned}$$

It follows that

$$Q_{n-1}(B)(l^{(0)} - \overline{l}) = (I - B)^{-1}(l^{(0)} - l^{(n)})$$

and

$$f^{(n)} = UA^T D^{-1}(I - B)^{-1}(l^{(0)} - l^{(n)}).$$

Clearly, when $l^{(n)}$ tends to $\overline{l}$, $f^{(n)}$ tends to f. □

Therefore, such schemes generate the same flow as the corresponding generalized diffusion algorithm. In the homogeneous case, the polynomial schemes were studied by different authors [62, 43, 15]. Extensions to the heterogeneous case of these results were also suggested by Elsässer et al. [19].

7.1.2 Second order schemes

In this subsection we investigate a particular case of polynomial schemes, the second order schemes (SOS for short). To make the difference, the diffusion is often referred to as a first order scheme (FOS for short) [62, 19], while a second order scheme refers to an iterative process of type

$$\begin{cases} l^{(1)} &= Ml^{(0)}, \\ l^{(k)} &= \omega M l^{(k-1)} + (1-\omega) l^{(k-2)}, \ k \geq 2 \end{cases}$$

In the homogeneous case, such schemes were investigated in the literature [62, 15, 16]. Their extension to the heterogeneous case follows naturally and some related work has been done [19].

The load vector at the step n can be expressed as $l^{(n)} = P_n(M)\, l^{(0)}$, with the polynomials P_n satisfying the following recurrence:

$$\begin{cases} P_0(x) &= 1, \\ P_1(x) &= x, \\ P_k(x) &= \omega\, x\, P_{k-1}(x) + (1-\omega)\, P_{k-2}(x), k \geq 2, \end{cases}$$

for all x.

Now, taking into consideration the form of P_n expressed in the equation (7.1.3), i.e. $P_n(x) = 1 + (x-1)Q_{n-1}(x)$, and replacing this in the above recurrence, one gets, after some simplifications,

$$Q_{n-1}(x) = \omega Q_{n-2}(x) + (1-\omega)Q_{n-3}(x) + \omega P_n(x), \forall x.$$

We have shown in the previous section that the partial flow generated at step n by the polynomial schemes is as given in the equation (7.1.5), i.e. $f^{(n)} = UA^T D^{-1} Q_{n-1}(B)(l^{(0)} - \overline{l})$. Hence, one gets the following recurrence involving the partial flow between i and j at the step n:

$$f_{ij}^{(n)} = \omega f_{ij}^{(n-1)} + (1-\omega) f_{ij}^{(n-2)} + \delta_{ij}^{(n)},$$

where

$$\delta_{ij}^{(n)} = \omega\, u_{ij}\, (l_i^{(n)}/c_i - l_j^{(n)}/c_j)$$

is the fraction of load differences to be exchanged at step n between the processors i and j. One may go further and write

$$f_{ij}^{(n)} = f_{ij}^{(n-1)} + \psi_{ij}^{(n-1)},$$

with

$$\psi_{ij}^{(n)} = -(1-\omega)\psi_{ij}^{(n-1)} + \delta_{ij}^{(n)},$$

where $\psi_{ij}^0 = 0$ and $\phi_{ij}^0 = 0$. This last recurrence asymptotically converges to the flow indicated in theorem 17. With these settings, the algorithm corresponding to a second order scheme is as described in the algorithm 4. A second order scheme based on the generalized diffusion matrix M generates the same flow as the generalized diffusion algorithm based on M.

Algorithm 4 Second order scheme

$k = 0$;
while (not converged) **do**
  for all i **do**
    send $l_i^{(k)}$ to neighbors
    receive $l_j^{(k)}$ from all neighbors j
    for all neighbors j of i **do**
      $\delta_{ij}^{(k)} = \omega\, u_{ij}\, (l_i^{(k)}/c_i - l_j^{(k)}/c_j)$
      $\psi_{ij}^{(k)} = -(1-\omega)\, \psi_{ij}^{(k-1)} + \delta_{ij}^{(k)}$
      $f_{ij}^{(k+1)} = f_{ij}^{(k)} + \psi_{ij}^{(k)}$
      $l_i^{(k+1)} = \omega\, l_i^{(k)} + (1-\omega)\, l_i^{(k-1)} - \sum_{(i,j)} \delta_{ij}^{(k)}$
    end for
  end for
  $k = k + 1$
end while

The flow generated by the algorithm 4 may be rewritten in the following equivalent form:

$$f_{ij}^{(n)} = \sum_{r=0}^{n} \sum_{k=0}^{r} (-1)^k (1-\omega)^k \delta_{ij}^{(r-k)}.$$

The recurrence (7.1.2) converges whenever $\omega \in (0,2)$ [82]. The fastest convergence occurs for

$$\omega_{opt} = \frac{2}{1+\sqrt{1-\lambda_M^2}}.$$

In this case, as shown by Varga [82])

$$\max_{x\in[-\lambda_M,\lambda_M]} |P_k(x)| = (\omega_{opt}-1)^{\frac{k}{2}}(1+k\sqrt{1-\lambda_M^2}).$$

As it was shown above, the polynomial schemes satisfy the relation (7.1.2):

$$l^{(n)} - \overline{l} = P_n(B)(l^{(0)} - \overline{l}), \tag{7.1.6}$$

where $B = M - ce^T$. On the other hand, $D^{-1/2}p_n(B)D^{-1/2}$ is real symmetric and

$$\|D^{-1/2}p_n(B)D^{-1/2}\| = \rho(P_n(B)) = \max_{i=1}^{p} P_n(\lambda_i) \leq \max_{x\in[-\lambda_M,\lambda_M]} |P_k(x)|.$$

Therefore,

$$\|l^{(n)} - \overline{l}\| \leq \frac{c_{\min}}{c_{\max}} (\omega_{opt}-1)^{\frac{n}{2}} \left(1+n\sqrt{1-\lambda_M^2}\right) \|l^{(0)} - \overline{l}\|$$

Muthukrishnan et al. showed [62] that

- if ω is fixed independently of M, such that $0 < \omega < 1$, there are diffusion algorithms for which the corresponding SOS takes longer to ϵ-balance.
- for any diffusion matrix M can be chosen a value $1 < \omega_M < 2$, dependent on M, for which the corresponding SOS ϵ-balances faster than the GDA based on the same M. The optimal value for ω is then $\omega_M = 2/(1+\sqrt{1-\lambda_M^2})$.

However, in a dynamic context in which the diffusion matrix may change often, the computation of λ_M attenuates the advantage offered by a second order scheme that uses ω_M. Generally, the computation of the eigenvalues is costly. However, we propose here another value for ω, which can be computed fastly at runtime. Instead of λ_M, we take the upper bound given in corollary 1 of theorem 10. We propose

$$\omega^\star = \frac{2}{1+\sqrt{K(2-K)}}, \tag{7.1.7}$$

with

$$K = 4\,e(G)\,\frac{w_{\min}}{\Delta^w+\epsilon}\,\frac{c_{\min}}{c_{\max}}\,\sin^2\left(\frac{\pi}{2p}\right).$$

Clearly, we have $1 < \omega^\star < 2$. As was shown by Muthukrishnan et al. [62], the SOS using this value theoretically converges faster than the corresponding simple diffusion algorithm.

Other polynomial iterative schemes were proposed by different authors [43, 15, 17]. Unfortunately, they suffer of the same flaw: they rely upon parameters that are dependent on the eigenvalues of the underlying diffusion matrix. As the communication topology, the network and the processor parameters may change often at runtime, the computation of these eigenvalues constitutes a serious overhead that should be taken into account.

7.2 Heterogeneous implicit schemes

In the homogeneous case, alternative iterative approaches were studied, as for example that using implicit schemes of the following type [28]:

$$(I + AWA^T)l^{(k+1)} = l^{(k)}. \tag{7.2.1}$$

Watts and Taylor proposed an implicit diffusion scheme for the case when the processors may have different capacities. Unfortunately, they ignored the impact of network parameters. Their scheme corresponds to that described in the algorithm 5 [89, 87]. They considered a measure for load balance, that they called efficiency, defined in as $eff = U_{bal}/U_{max}$, where in our terms, $U_i = l_i/c_i$ and $U_{bal} = \sum_i U_i / \sum_i c_i$. The authors did not analyze the quality of the generated balancing flow. We consider here a more general scheme which is able to take into consideration also the communication costs. More specifically, we consider and analyze the following more general scheme:

$$(I + D^{1/2}\mathcal{L}D^{-1/2})l^{(k+1)} = l^{(k)}, \tag{7.2.2}$$

with $\mathcal{L} = D^{-1/2}LD^{-1/2}$ and $L = AUA^T$ and U, S, W $q \times q$ diagonal matrices such that $U = \text{diag}(u), S = \text{diag}(s), W = \text{diag}(w)$ and $u_k = s_k\, w_k$, for all $1 \leq k \leq q$, where u is an array of edge weights, s an array of scalars and w the array of communication costs. It must be noted that the above algorithm is just a particular case of this scheme, for $w_k = \alpha$ and $s_k = \frac{c_i c_j}{c_i + c_j}$, for all $1 \leq k \leq q$ and $e_k = \{i, j\}$.

The iterative scheme (7.2.2) converges to the fair workload vector. The proof relies on the fact

Algorithm 5 Implicit diffusion scheme

$f_{ij} = 0, \text{for all } j \in \mathcal{N}_i$

$\beta_i = 1 + \alpha \sum_{j \in \mathcal{N}_i} \frac{c_j}{c_i + c_j}$

$T_{ij} = \alpha \frac{c_i}{c_i + c_j}$

$m = \left\lceil \frac{\ln(\alpha)}{\ln(\max_i(\beta_i^{-1} \sum_{j \in \mathcal{N}_i} T_{ij}))} \right\rceil$

while $eff < eff_{min}$ **do**

$l_j^{(0)} = l_j$

for $k = 1$ to m **do**

$l_i^{(k)} = \beta_j^{-1}(l_i^{(0)} + \sum_{j \in \mathcal{N}_i} T_{ij}\, l_j^{(k-1)})$

end for

$f_{ij} = f_{ij} + \beta_j^{-1} T_{ij}\, (\frac{c_j}{c_i}\, l_i^{(0)} - l_j^{(m-1)})$, for all $j \in \mathcal{N}_i$

$l_i = l_i^{(m)}$

end while

Algorithm 6 Generalized implicit scheme

$f_{ij} = 0, \text{for all } j \in \mathcal{N}_i$

$\beta_i = 1 + \sum_{j, e_k = \{j,i\}} \frac{s_k\, w_k}{c_j}$

$T_{ij} = \frac{s_k\, w_k}{c_j}$, for all $e_k = \{i, j\}$

$m = \left\lceil \frac{\ln(\alpha)}{\ln(\max_i(\beta_i^{-1} \sum_{j \in \mathcal{N}_i} T_{ij}))} \right\rceil$

while $eff < eff_{min}$ **do**

$l_j^{(0)} = l_j$

for $k = 1$ to m **do**

$l_i^{(k)} = \beta_j^{-1}(l_i^{(0)} + \sum_{j \in \mathcal{N}_i} T_{ij}\, l_j^{(k-1)})$

end for

$f_{ij} = f_{ij} + \beta_j^{-1} T_{ij}\, (\frac{c_j}{c_i}\, l_i^{(0)} - l_j^{(m-1)})$, for all $j \in \mathcal{N}_i$

$l_i = l_i^{(m)}$

end while

that the inverse of $I + D^{1/2}\mathcal{L}D^{-1/2}$ exists and has the eigenvalues

$$0 < \frac{1}{\mu_p + 1} \leq \ldots \leq \frac{1}{\mu_2 + 1} < \frac{1}{\mu_1 + 1} = 1, \tag{7.2.3}$$

with e^T and c left and respectively right eigenvectors corresponding to the largest eigenvalue 1. If we consider

$$B' = (I + D^{1/2}\mathcal{L}D^{-1/2})^{-1} - ce^T,$$

one can verify that this matrix has only nonnegative and subunitary eigenvalues. As equation (7.2.2) can be rewritten in the equivalent form $l^{(k+1)} - \overline{l} = B'(l^{(k)} - \overline{l})$, it is guaranteed that $(l^{(k)})_{k\geq 0}$ converges to $\overline{l}$.

Simple transformations show that the relation (7.2.2) can be also rewritten as

$$l^{(k+1)} = l^{(k)} - AUA^T D^{-1}(I + D^{1/2}\mathcal{L}D^{-1/2})^{-1} l^{(k)}.$$

Theorem 18. *The balancing flow generated by a heterogeneous implicit scheme is*

$$f = UA^T D^{-1}(I - B)^{-1} l^{(0)}.$$

Proof. The migration flow computed by such a scheme is

$$f = UA^T D^{-1} \sum_{k\geq 1} Q^k l^{(0)},$$

with Q denoting the matrix $(I + D^{1/2}\mathcal{L}D^{-1/2})^{-1}$. Going further, one gets that an implicit scheme defined as above generates the following migration flow:

$$\begin{aligned} f &= UA^T D^{-1} \sum_{k\geq 1} Q^k l^{(0)} \\ &= UA^T D^{-1} Q \sum_{k\geq 0} Q^k l^{(0)} \\ &= UA^T D^{-1} Q \sum_{k\geq 0} (B'^k + ce^T) l^{(0)} \\ &= UA^T D^{-1} Q \sum_{k\geq 0} B'^k l^{(0)} \\ &= UA^T D^{-1} Q (I - B')^{-1} l^{(0)}. \end{aligned}$$

On the other hand, we have

$$\begin{aligned}(I - B')^{-1} &= (I - (I + D^{1/2}\mathcal{L}D^{-1/2})^{-1} + ce^T)^{-1} \\ &= (I - Q + ce^T)^{-1} \\ &= Q^{-1}(D^{1/2}\mathcal{L}D^{-1/2} + ce^T)^{-1} \\ &= Q^{-1}D^{1/2}(\mathcal{L} + c^{1/2}c^{T^{1/2}})^{-1}D^{-1/2}.\end{aligned}$$

Finally, we are left with

$$\begin{aligned}f &= UA^TD^{-1/2}(\mathcal{L} + c^{1/2}c^{T^{1/2}})^{-1}D^{-1/2}l^{(0)} \\ &= UA^TD^{-1}(LD^{-1} + ce^T)^{-1}l^{(0)} \\ &= UA^TD^{-1}(I - B)^{-1}l^{(0)}\end{aligned}$$

□

Therefore, implicit schemes like those given by equation (7.2.2) compute the same balancing flow as the simple diffusion algorithm based on $M = I - AUA^TD^{-1}$.

7.3 Conclusions, issues

In this chapter we put in evidence that the polynomial schemes generate the same balancing flow as the corresponding GDA's in the same heterogeneous computing environment. We further considered implicit diffusion schemes as those proposed by Heirich [28] in the homogeneous case and Watts [89] in the heterogeneous case. They ignored the quality of the balancing flow generated by such schemes. Here, we showed that these schemes generate the same balancing flow as the corresponding generalized diffusion algorithm, too. As a natural consequence, as it was shown in the case of the generalized diffusion, the generated flow is a scaled projection of any other balancing flow and, consequently, it minimizes a weighted 2-norm.

The analysis carried out in the case of generalized diffusion showed that this algorithm is rather slow, although it is better than other techniques (the hydrodynamic algorithm). This is not surprising as it corresponds to the conclusions drawn in the homogeneous case [6]. On the other hand, the

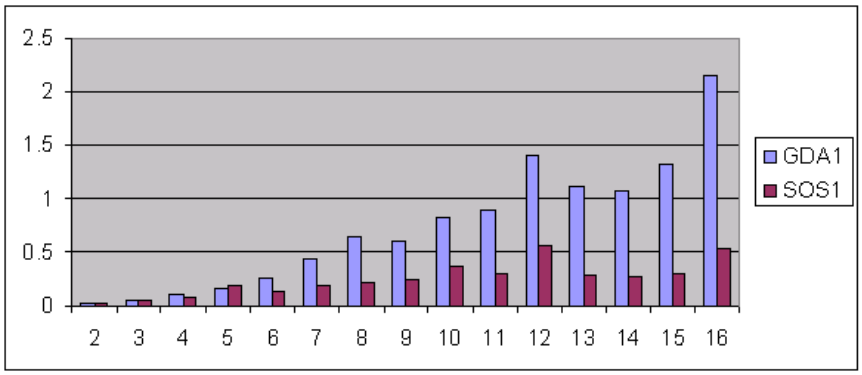


Figure 7.1: Execution times of GDA1 and SOS1, varying with the number of processors, when the communication topology is a path.

polynomial schemes like those described by Diekmann et al. [15] and Hu et al. [43] require knowledge at runtime of the eigenvalues of the generalized Laplacian matrix. These eigenvalues should be recomputed every time the communication graph, the network parameters or the available processor capacities change.

In the case of the second order schemes we suggested the use of another parameter than that depending on the second largest eigenvalue, as is indicated in equation (7.1.7). We performed tests with the generalized diffusion algorithm GDA1 that uses the diffusion matrix defined by the equation (5.6.1) and with the second order scheme SOS1 that uses the same diffusion matrix and the parameter $\omega^\star$. As the figures 7.1, 7.2 and 7.3 illustrate, the second order scheme that uses this parameter is better than the corresponding generalized diffusion algorithm. In this case, we see that the use of the value $\omega^\star$ results in smaller execution times for SOS1 when compared to GDA1. Therefore, the use of the proposed parameter may result in a better execution time for SOS1 as compared to GDA1. This solution has the advantage that it avoids the explicit computation of the eigenvalues for computing the optimal parameter ω.

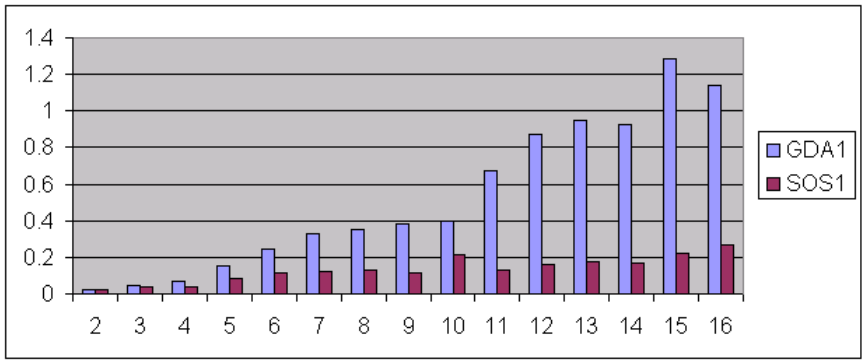


Figure 7.2: Execution times of GDA1 and SOS1, varying with the number of processors, when the communication topology is a binary tree.

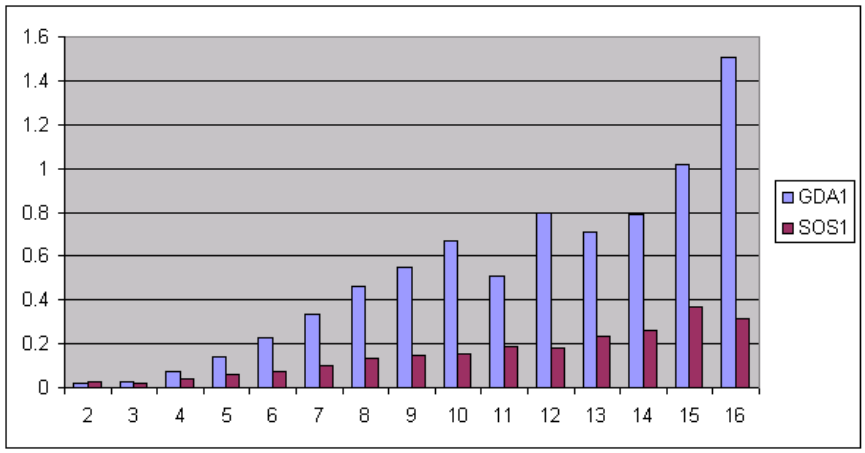


Figure 7.3: Execution times of GDA1 and SOS1, varying with the number of processors, when the communication topology is a 4-tree.

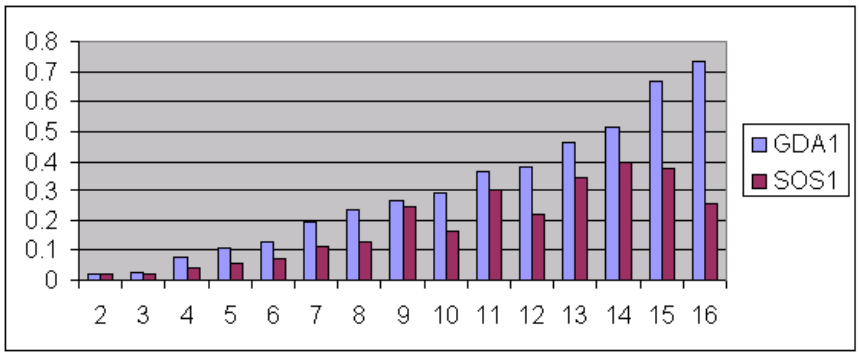


Figure 7.4: Execution times of GDA1 and SOS1, varying with the number of processors, when the communication topology is a star graph.

Chapter 8

Faster dynamic load balancing schemes

8.1 A flow optimization problem

As remarked by Golub and Van Loan, a difficulty associated with the iterative methods of the type of second order schemes and other polynomial schemes is that they depend upon parameters that are sometimes hard to choose properly [26]. They need good estimates of one or more eigenvalues and it may be analytically impossible or computationally expensive to do this. In the case of the dynamic load balancing problem, these schemes iteratively approximate the minimal balancing flow (w.r.t. the indicated norm) in a heterogeneous environment. Although they essentially use local communication, they may converge slowly for certain classes of communication topologies. We try therefore to find faster methods for dynamic load balancing in heterogeneous environments.

The design of a dynamic load balancing method should take into consideration the following important aspects:

- In a distributed environment it is important to avoid as much as possible to perform unnecessary communication and to call primitives that need global communication. Ideally, the communication should restrict to data exchanges between neighbors.

- The communication time needed for the migration must be minimized.

Clearly, several balancing flows may exist in the communication graph induced by a parallel application. As an example, for a computational environment with four processors and a communication topology of type ring with a diagonal, as in figure 8.1, at least two balancing flows can be defined. However, only one minimizes the Euclidian norm (the one on the left hand side of the figure).

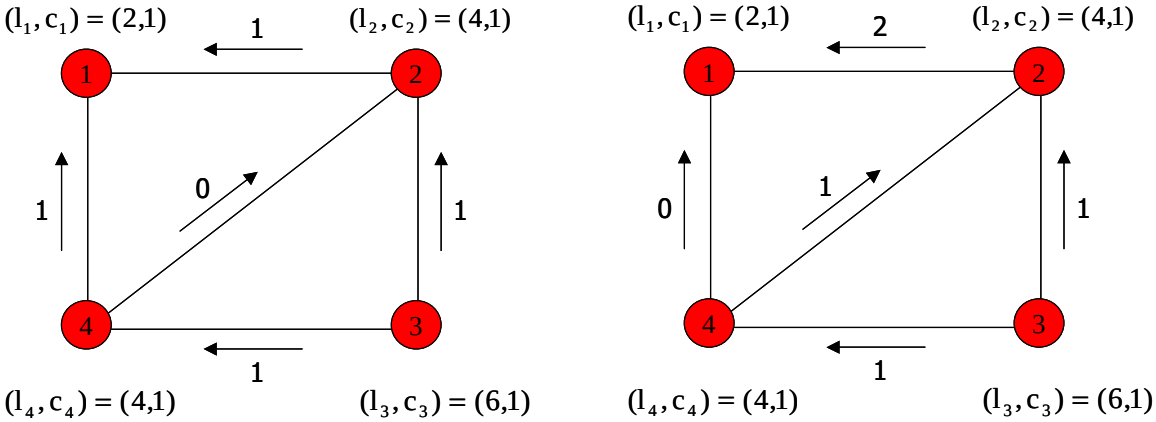


Figure 8.1: On the left, $\|x\|^2 = 4$. On the right, $\|x\|^2 = 7$.

The communication time needed for the migration must be minimized. Typically this can be achieved by keeping as low as possible the maximum cost of data transmission between two processors. Unfortunately, minimizing the maximum norm is hardly feasible. However, it is possible to minimize a weighted 2-norm. As was put in evidence in the previous chapters, the diffusion-like schemes naturally minimize such a norm. It is important to avoid as much as possible the global communication, limiting the load/information exchanges to the neighbors. The problem of finding a minimal balancing flow can be formulated directly as a quadratic programming problem as follows:

$$\begin{aligned} \text{minimize} \quad & \tfrac{1}{2} x^T W^{-1} x \\ \text{subject to} \quad & A\,x = l - \overline{l}. \end{aligned} \tag{8.1.1}$$

In the homogeneous case, the problem was addressed by Hu et al. [45]. A solution based on *Lagrange multipliers* was suggested and a conjugate gradient method was used for computing them.

Here, we try to see how this problem can be adapted and implemented for a heterogeneous computational model.

An important observation is that, as Bertsekas and Tsitsiklis showed [5], the solution of (8.1.1) can be found by solving the associated dual problem

$$\begin{aligned} &\text{minimize} \quad \tfrac{1}{2}u^T AWA^T u + u^T(l - \overline{l}), \\ &u \in \mathbb{R}^p \end{aligned}$$

which rewrites as

$$\begin{aligned} &\text{minimize} \quad \tfrac{1}{2}u^T Lu + u^T(l - \overline{l}). \\ &u \in \mathbb{R}^p \end{aligned} \tag{8.1.2}$$

Assuming that $u^\star$ is the solution of equation (8.1.2), the solution of equation (8.1.1) is

$$x^\star = -WA^T u^\star.$$

We consider now the *generalized Laplacian* of G, $\mathcal{L} = D^{-1/2}AWA^T D^{-1/2}$. It can be easily verified that it has 0 as the smallest eigenvalue, ${c^T}^{1/2}$ and $c^{1/2}$ being the corresponding left and right eigenvectors. Furthermore, as G is a connected graph, the eigenvalue 0 has the algebraic multiplicity 1 [61, 66]. Let $\mu_i, 1 \leq i \leq p$, be the eigenvalues of $\mathcal{L}$; we shall suppose that they are in increasing order:

$$0 = \mu_1 < \mu_2 \leq \mu_3 \leq \ldots \leq \mu_p.$$

The matrix $\mathcal{L} + c^{1/2}{c^T}^{1/2}$ plays a key role as will be shown later. The following result holds:

Lemma 11. *Let $\Lambda(A)$ denote the set of the eigenvalues of a square matrix A. Then,*

$$\Lambda(\mathcal{L} + c^{1/2}{c^T}^{1/2}) = \Lambda(\mathcal{L}) \setminus \{0\} \cup \{1\}.$$

Proof. Let μ be an eigenvalue of $\mathcal{L}$, $\mu \neq 0$, and x a corresponding eigenvector. As ${c^T}^{1/2}$ and $c^{1/2}$ are left and right eigenvectors corresponding to the eigenvalue 0, we have that $0 = {c^T}^{1/2}\mathcal{L}x = \mu{c^T}^{1/2}x$. It follows that $0 = {c^T}^{1/2}\mathcal{L}$ and $(\mathcal{L} + c^{1/2}{c^T}^{1/2})x = \mathcal{L}x = \mu x$. Additionally, we have that $(\mathcal{L} + c^{1/2}{c^T}^{1/2})c^{1/2} = c^{1/2}$, as $\sum_{i=1,p} c_i = 1$. Therefore,

$$\Lambda(\mathcal{L}) \setminus \{0\} \cup \{1\} \subseteq \Lambda(\mathcal{L} + c^{1/2}{c^T}^{1/2}).$$

Conversely, we consider $\{\mu', x'\}$ such that $\mu' \neq 1$ and

$$(\mathcal{L} + c^{1/2}{c^T}^{1/2})x' = \mu' x'. \tag{8.1.3}$$

On the other hand, we know that $(\mathcal{L} + c^{1/2}c^{T\,1/2})c^{1/2} = c^{1/2}$. Multiplying the relation (8.1.3) on the left by $c^{T\,1/2}$ one gets that $c^{T\,1/2}x' = \mu' c^{T\,1/2}x'$, which yields $c^{T\,1/2}x' = 0$. Therefore, equation (8.1.3) reads as $\mathcal{L}x' = \mu' x'$ and we have that $\Lambda(\mathcal{L} + c^{1/2}c^{T\,1/2}) \setminus \{1\} \subseteq \Lambda(\mathcal{L})$. As 1 is an eigenvalue of $\mathcal{L} + c^{1/2}c^{T\,1/2}$, too, we have that $\Lambda(\mathcal{L} + c^{1/2}c^{T\,1/2}) \subseteq \Lambda(\mathcal{L}) \cup \{1\}$. Let us assume that 0 is an eigenvalue of (8.1.3). Then, multiplying the above equation on the left by $c^{T\,1/2}$ one gets that $c^{T\,1/2}x'$ should be 0. This yields that x' should be an eigenvector corresponding to the eigenvalue 0. As it has the multiplicity 1, one gets that $x' = \alpha c^{1/2}$, $\alpha \neq 0$, and $c^{T\,1/2}x' = \alpha \neq 0$, which is a contradiction. Therefore, $\Lambda(\mathcal{L} + c^{1/2}c^{T\,1/2}) \subseteq \Lambda(\mathcal{L}) \setminus 0 \cup \{1\}$. Finally, we conclude that $\Lambda(\mathcal{L} + c^{1/2}c^{T\,1/2}) = \Lambda(\mathcal{L}) \setminus \{0\} \cup \{1\}$. □

As a consequence, because $\mathcal{L} + c^{1/2}c^{T\,1/2}$ has only positive eigenvalues, its inverse exists and has the same eigenvectors. Therefore, $c^{T\,1/2}$ and $c^{1/2}$ are left and right eigenvectors corresponding to the eigenvalue 1 of the inverse.

It is easy to prove, based on the above results, that the solution of (8.1.2) is of the form $u^\star = D^{-1/2}v^\star$, where $v^\star$ is the solution of the problem

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}v^T(\mathcal{L} + c^{1/2}c^{T\,1/2})v + v^T D^{-1/2}(l - \bar{l}). \\ v \in \mathbb{R}^p & \end{array} \tag{8.1.4}$$

Clearly, the matrix $\mathcal{L} + c^{1/2}c^{T\,1/2}$ is symmetric and positive definite. As a consequence, the minimum is achieved in the above problem for $v^\star = -(\mathcal{L} + c^{1/2}c^{T\,1/2})^{-1}D^{-1/2}(l - \bar{l})$ and the solution of (8.1.1) is then

$$x^\star = -WA^T u^\star = -WA^T D^{-1/2}v^\star = WA^T D^{-1/2}(\mathcal{L} + c^{1/2}c^{T\,1/2})^{-1}D^{-1/2}(l - \bar{l}).$$

Theorem 19. *1. In a heterogeneous computational model $H = (G, l, c, w)$, the following vector of size $q = |E(G)|$ is a balancing flow:*

$$\phi = WA^T D^{-1/2}(\mathcal{L} + c^{1/2}c^{T\,1/2})^{-1}D^{-1/2}l. \tag{8.1.5}$$

2. There is a projection matrix P such that for any balancing flow x

$$\phi = W^{-1/2}PW^{1/2}x\,.$$

Proof. First, we show that ϕ is a balancing flow:

$$\begin{aligned} A\phi &= D^{1/2}\mathcal{L}(\mathcal{L}+c^{1/2}c^{T\,1/2})^{-1}D^{-1/2}l \\ &= b - D^{1/2}c^{1/2}c^{T\,1/2}(\mathcal{L}+c^{1/2}c^{T\,1/2})^{-1}D^{-1/2}l \\ &= b - D^{1/2}c^{1/2}c^{T\,1/2}D^{-1/2}l \\ &= l - ce^{T}l = l - \bar{l}. \end{aligned}$$

Let x be a arbitrary balancing flow in H. Replacing l by $Ax + \bar{l}$ in the expression of ϕ one gets

$$W^{-1/2}\phi = W^{1/2}A^{T}D^{-1/2}(\mathcal{L}+c^{1/2}c^{T\,1/2})^{-1}D^{-1/2}Ax.$$

Let P denote the quantity

$$W^{1/2}A^{T}D^{-1/2}(\mathcal{L}+c^{1/2}c^{T\,1/2})^{-1}D^{-1/2}AW^{1/2}.$$

It is easy to verify that $P = P^{T}$ and $P^{2} = P$, i.e. P is a projection matrix. Therefore,

$$\phi = W^{1/2}PW^{-1/2}x$$

and

$$\|W^{-1/2}\phi\|_2^2 = \|W^{-1/2}x\|_2^2 - \|(I-P)W^{-1/2}x\|_2^2.$$

□

Theorem 20. *The balancing flow ϕ given in theorem 19 is the unique balancing flow in $H = (G, l, c, w)$ that minimizes the norm $\|\cdot\|_{2,W^{-1/2}}$.*

Proof. The result is a direct consequence of the above theorem. Indeed, let ϕ denote the balancing flow expressed as in 8.1.5 and $x \in \mathbb{R}^q$ a balancing flow such that

$$\|x\|_{2,W^{-1/2}} = \min_{y\in\mathcal{F}(H)} \|y\|_{2,W^{-1/2}}. \tag{8.1.6}$$

Then,

$$\|W^{-1/2}\phi\|_2^2 + \|(I-P)W^{-1/2}x\|_2^2 = \|W^{-1/2}x\|_2^2.$$

As x is minimal, one should have that $\|(I-P)W^{-1/2}x\|_2 = 0$, which is possible only if $(I-P)W^{-1/2}x = 0$ i.e. $W^{-1/2}x = PW^{-1/2}x$, that is $x = W^{1/2}PW^{-1/2}x = \phi$. □

Another important property for guiding the data migration is the following result that is straightforward to prove:

Theorem 21. *Let ϕ be the solution of equation* (8.1.1) *and D_+ the digraph obtained by orienting the edges of G so that there is an arc (i,j) if and only if $\phi_{ij} > 0$. Then D_+ is acyclic.*

This ensures the absence of circular dependencies during the effective migration phase.

Having established the expression of the minimal balancing flow w.r.t. the indicated norm, the problem is how to compute it efficiently? The iterative methods of the type of the generalized diffusion represent a possibility. Unfortunately, they have the drawback that they may converge slowly. An alternative way, that is a straightforward issue of the above results, is that it can be computed using a parallel conjugate gradient method (PCG for short). The algorithm 7 corresponds to the serial variant described by Golub et al. [26], for the case of a positive definite matrix A and a vector b. The algorithm uses only 4 vectors of size p: x,r,v and w. The method proceeds by generating vector sequences of successive approximations to the solution, residuals corresponding to the approximations, and search directions used in updating these vectors and residuals [2]. In every iteration of the method, two inner products are performed in order to compute update scalars that are defined to make the sequences satisfy certain orthogonality conditions. On a symmetric positive definite linear system these conditions imply that the distance to the solution is reduced in some norm.

In the homogeneous case, an efficient algorithm for dynamic load balancing using this method was described by Hu et al. [45]. According to Golub and al. [26], the number of iterations that this algorithm executes is at most the number of distinct eigenvalues of the matrix A, in our case of the Laplacian matrix ($\leq p$).

In the sequel, we describe a parallel algorithm for computing the minimum balancing flow, which is in fact a parallel version of the conjugate gradient method described by Golub et al. [26]. We implemented it using a SPMD programming model and the MPI communication library. The following data structures are used by each processor:

1. *Load* - workload assigned to the processor itself

2. *Cap* - capacity of the processor itself

Algorithm 7 Conjugate Gradient Algorithm

$x_0 =$ initial guess

$k = 0$

$r = b - A\,x_0$

$\rho_0 = \|r\|_2^2$

while $(\sqrt{\rho_k} > \epsilon\|b\|_2) \wedge (k < k_{max})$ **do**

$k = k + 1$

if $k = 1$ **then**

$u = r$

else

$\beta_k = \rho_{k-1}/\rho_{k-2}$

$u = r + \beta_k u$

end if

$w = A\,u$

$\alpha_k = \rho_{k-1}/u^T w$

$x = x + \alpha_k\,u$

$r = r - \alpha_k\,w$

$\rho_k = \|r\|_2^2$

end while

3. $TotLoad$ - sum of all workloads of the processors
4. $FairLoad$ - fair workload of the processor itself
5. $nbNbrs$ - number of neighbors (the degree of the processor in the communication graph)
6. $arrNb$[0..nbNbrs-1] - array of the neighbors' ids
7. $eWght$[0..nbNbrs-1] - array of edge weights representing the cost of communication with the neighbors
8. $Flow$[0..nbNbrs-1] - array of flows to be sent/received to/from the neighbors
9. $b, r, u, w, bnrm2, rho0, rho1, \alpha, \beta, dotp$ - local real numbers
10. We further make the specification that the semantics of the primitive **allreduce** (a, b, op) is that b is the result of applying a reduction operation op to all local values a.

With these settings, the PCG method is as described in algorithm 8. The algorithm performs at each iteration step a number of local communication operations and a number of global communication operations. Our implementation performs two reduction operations at each iteration step. The global summation normally can be done in $O(\log p)$ parallel steps, therefore the PCG algorithm takes at most $\mathrm{O}(p \log p)$ parallel steps. Compared to a generalized diffusion algorithm, it is theoretically faster. The experimental tests confirmed that PCG is significantly faster than any diffusion algorithm.

8.2 An algorithm based on the Laplacian polynomial

In this section we present an algorithm for dynamic load balancing in heterogeneous environments based on the Laplacian polynomial of the communication graph. It presents the advantage of computing the required balancing flow in only $O(p-1)$ parallel steps as opposed to the generalized diffusion algorithm which may make $O((c_{\max}\,\Delta^w\,p^2)/(c_{\min}\,w_{\min}\,e(G))$ steps. It "exactly" computes this flow. As opposed to the PCG algorithm algorithm, it does not make use of global communication but requires knowledge at runtime of the coefficients of the characteristic polynomial of the communication graph. We show how to efficiently compute these coefficients in parallel.

Algorithm 8 PCG

allreduce $(Load, TotLoad, 1, SUM)$

$FairLoad = Cap \cdot TotLoad$

$b = FairLoad - Load$

allreduce $(b \cdot b, bnrm2, SUM)$

$k = 0$

$w = 0.0$

$r = b - w$

allreduce $(r \cdot r, rho0, SUM)$

$rho1 = rho0$

while $(sqrt(rho1) > \epsilon\, sqrt(bnrm2)) \wedge (k < k_{max})$ **do**

$k = k + 1;$

if $(k == 1)$ **then**

$u = r$

else

$beta = rho1/rho0$

$u = r + beta \cdot u$

end if

send to neighbors the local u value

receive from neighbors the values of u and store them in $u_1, \ldots, u_{nbNbrs}$

for $(i = 0; i < nbNbrs; i = i + 1)$

$w = w + eWght[i]\,(u - u_i)$

end for

allreduce $(u \cdot w, dotp, SUM)$

$alpha = rho1/dotp$

$x = x + alpha \cdot u$

$r = r - alpha \cdot w$

allreduce $(r \cdot r, rho1, SUM)$

end while

send to neighbors the local x value

receive from neighbors the local x values in $x_1, \ldots, x_{nbNbrs}$

for $(i = 0; i < nbNbrs; i = i + 1)$ **do**

$Flow[i] = eWght[i]\,(x - x_i)$

end for

The characteristic polynomial of a square matrix A can be defined as

$$\chi(\mu) = det(\mu I - A) = \sum_{k=0}^{p} a_k \mu^{p-k}.$$

Following an usage introduced by Kelmans et al. [54], we refer to the characteristic polynomial of the generalized Laplacian as the *Laplacian polynomial*. Without loss of generality we may assume that $a_0 = 1$.

As stated in the previous chapters, for a given model (G, l, c, w), 0 is an eigenvalue with the algebraic multiplicity 1 of the generalized Laplacian matrix $\mathcal{L}$. From the Viète's relations, we have that

$$\begin{aligned} a_{p-1} &= \sum_{k=1,p} \prod_{i \neq k} \mu_i, \\ a_p &= \mu_1 \mu_2 \cdots \mu_p = 0. \end{aligned}$$

As we assumed a heterogeneous model (G, l, c, w) with G being connected, the eigenvalue 0 has the multiplicity 1 and the other eigenvalues are strictly positive. On the other hand, the Cayley-Hamilton theorem asserts that $\mathcal{L}$ satisfies $\chi(\mathcal{L}) = 0$. With these observations, it follows immediately that

$$(\mathcal{L} + c^{1/2} c^{T\,1/2})(\mathcal{L}^{p-1} + a_1 \mathcal{L}^{p-2} + \ldots + a_{p-1} I) \;=\; a_{p-1} c^{1/2} c^{T\,1/2}. \tag{8.2.1}$$

As the inverse of $\mathcal{L} + c^{1/2} c^{T\,1/2}$ exists and has the properties mentioned in the previous section, one has

$$\mathcal{L}^{p-1} + a_1 \mathcal{L}^{p-2} + \ldots + a_{p-1} I \;=\; a_{p-1} (\mathcal{L} + c^{1/2} c^{T\,1/2})^{-1} c^{1/2} c^{T\,1/2}, \tag{8.2.2}$$

and

$$a_{p-1} (\mathcal{L} + c^{1/2} c^{T\,1/2})^{-1} c^{1/2} c^{T\,1/2} \;=\; a_{p-1} c^{1/2} c^{T\,1/2}. \tag{8.2.3}$$

Going further, one gets

$$\overline{l} \;=\; l^{(0)} + A W A^T D^{-1/2} \; \frac{\mathcal{L}^{p-2} + a_1 \mathcal{L}^{p-3} + \ldots + a_{p-2} I}{a_{p-1}} \; D^{-1/2} l^{(0)}. \tag{8.2.4}$$

The last result constitutes the basis of algorithm 9.

Algorithm 9 LPS

for all i **do**
  $u_i = 0$
  $s_i^{(0)} = l_i^{(0)} / \sqrt{c_i}$
end for
for all i **do**
  for $t = 0$ to $p - 2$ **do**
    $u_i = u_i + a_{p-2-t} \cdot s_i^{(t)}$
    send $s_i^{(t)}$ to neighbors
    $s_i^{(t+1)} = \mathcal{L}_{ii} \cdot s_i^{(t)}$
    for all $j \in \mathcal{N}(i)$ **do**
      receive $s_j^{(t)}$ from j
      $s_i^{(t+1)} += \mathcal{L}_{ij} \cdot s_j^{(t)}$
    end for
  end for
end for
for all i **do**
  send u_i to neighbors
  receive u_j from all $j \in \mathcal{N}(i)$
  for all $j \in \mathcal{N}(i)$ **do**
    $x_{ij} = -w_{ij}/a_{p-1} \cdot (u_i/\sqrt{c_i} - u_j/\sqrt{c_j})$
    if $(x_{ij} > 0)$ **then**
      send x_{ij} units to j
    else
      receive x_{ij} units from j
    end if
  end for
end for

The above algorithm needs knowledge at runtime of the coefficients of the characteristic polynomial. We indicate in the next subsection several ways to efficiently compute the coefficients of the Laplacian polynomial in parallel. An optimal scheme that requires at most p steps, too, was proposed by Diekmann et al. [15], but this method requires the computation of all the eigenvalues of the Laplacian matrix and no method is suggested how to efficiently do this at runtime in a dynamic context that asks for several applications of a dynamic load balancing method. Our method requires knowledge of the coefficients of the Laplacian polynomial and we show in the sequel how to compute them efficiently in parallel.

8.2.1 Parallel algorithms for the computation of the coefficients of the Laplacian polynomial

A possible way of computing the coefficients of the characteristic polynomial of a square matrix A consists in using *Newton's identities*:

$$s_k + \sum_{i=1}^{k-1} a_{k-i} s_i + k a_k = 0, 1 \leq k \leq p, \tag{8.2.5}$$

where $s_k = \texttt{trace}(A^k)$, $\texttt{trace}(M)$ denoting the trace of matrix M, i.e. the sum of its diagonal elements. Finding the coefficients of the characteristic polynomial reduces therefore to solving the following linear system:

$$\begin{bmatrix} 1 & 0 & 0 & \dots & \dots & & 0 \\ s_1 & 2 & 0 & \dots & \dots & & \vdots \\ s_2 & s_1 & 3 & \dots & \dots & & \vdots \\ s_3 & s_2 & s_1 & 4 & \dots & & \\ \vdots & & \ddots & \ddots & \ddots & & \\ & & & \ddots & \ddots & \ddots & \vdots \\ s_{p-1} & \dots & & s_3 & s_2 & s_1 & p \end{bmatrix} \begin{bmatrix} a_{p-1} \\ a_{p-2} \\ a_{p-3} \\ \vdots \\ \vdots \\ a_1 \\ a_0 \end{bmatrix} = - \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ \vdots \\ \vdots \\ s_{p-1} \\ s_p \end{bmatrix}$$

The algorithm 10 can be used for computing the coefficients a_i.

Another way of computing these coefficients consists in using Berkowitz's algorithm. It computes the coefficients of the characteristic polynomial of a matrix A, $\chi(x) = \det(xI - A)$, by computing iterated matrix products. The main idea of this algorithm is the Samuelson's identity, which relates

Algorithm 10 Algorithm for computing the coefficients of the characteristic polynomial of a matrix

compute $A^2, A^3, \ldots, A^{p-1}$

compute $s_k = \texttt{trace}\,(A^k)$

solve the triangular system given by the Newton's relations

the characteristic polynomial of a matrix to the characteristic polynomial of its principal submatrix [50, 79]. The coefficients of the characteristic polynomial of a $p \times p$ matrix A are computed recursively. Assuming that M is the principal submatrix of A, we have

$$A = \begin{pmatrix} a_{11} & u^T \\ v & M \end{pmatrix}$$

where u^T, v and M are $1 \times (p-1)$, $(p-1) \times 1$ and $(p-1) \times (p-1)$ sub-matrices, respectively.

In the following we shall use the following notation:

- $A[k|l]$ denotes the matrix obtained from A by deleting the k-th row and the l-th column
- $A[-|l]$ denotes the matrix obtained from A by deleting l-th column
- $A[k|-]$ denotes the matrix obtained from A by deleting the k-th row
- $\mathrm{adj}(A)$ denotes the adjoint of a matrix A, i.e. $\mathrm{adj}(A)_{ij} = (-1)^{i+j} \det(A[j|i])$.

Lemma 12 (Samuelson's identity). *Let $\chi(x)$ and $Q(x)$ be the characteristic polynomials of A and M, respectively. Then: $\chi(x) = (x - a_{11})Q(x) - u^T\, adj(xI - M)\, v$*

Lemma 13. *If $Q(x) = Q_{p-1}x^{p-1} + \cdots + Q_1 x + Q_0$ is the characteristic polynomial of M, and if*

$$B(x) = \sum_{k=2}^{p} (Q_{p-1}M^{k-2} + \cdots + Q_{p-k+1}I)x^{p-k}, \tag{8.2.6}$$

then $B(x) = adj(xI - M)$.

The following identity, which is the basis for Berkowitz's algorithm, follows from lemma 12 and lemma 13:

$$\chi(x) = (x - a_{11})Q(x) - u^T\, B(x)\, v \tag{8.2.7}$$

Using equation (8.2.7), we can express the characteristic polynomial of a matrix as a matrix product.

Definition 8.2.1. A $p \times m$ matrix is *Toeplitz* if the values on each diagonal are the same. A matrix is *upper triangular* if all the values below the main diagonal are zero. A matrix is *lower triangular* if all the values above the main diagonal are zero.

Expressing equation (8.2.7) in a matrix form, one gets that

$$\chi = C_1 Q. \tag{8.2.8}$$

where C_1 is an $(p+1) \times p$ Toeplitz lower triangular matrix, and where the entries in the first column are defined as follows:

$$c_{i1} = \begin{cases} 1, & \text{if } i = 1 \\ -a_{11}, & \text{if } i = 2 \\ -(u^T M^{i-3} v), & \text{if } i \geq 3 \end{cases} \tag{8.2.9}$$

As an example, if A is a 6×6 matrix, then $z = C_1\, y$ is given by:

$$\begin{pmatrix} z_5 \\ z_4 \\ z_3 \\ z_2 \\ z_1 \\ z_0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ -a_{11} & 1 & 0 & 0 & 0 \\ -u^T v & -a_{11} & 1 & 0 & 0 \\ -u^T M v & -u^T v & -a_{11} & 1 & 0 \\ -u^T M^2 v & -u^T M v & -u^T v & -a_{11} & 1 \\ -u^T M^3 v & -u^T M^2 v & -u^T M v & -u^T v & -a_{11} \end{pmatrix} \begin{pmatrix} y_4 \\ y_3 \\ y_2 \\ y_1 \\ y_0 \end{pmatrix}$$

With this observations, given an $p \times p$ matrix A, over any field K, *Berkowitz's algorithms* computes a $(p+1) \times 1$ column vector χ as follows:

Let C_j be a $(p+2-j) \times (p+1-j)$ Toeplitz and lower-triangular matrix, with the entries in the first column are

$$\begin{cases} 1, & \text{if } i = 1 \\ -a_{jj}, & \text{if } i = 2 \\ -u_j^T M_j^{i-3} v_j, & \text{if } 3 \leq i \leq p+2-j \end{cases} \tag{8.2.10}$$

where M_j is the j-th principal submatrix ($M_1 = A[1|1]$, $M_2 = M_1[1|1]$, ..., $M_{k+1} = M_k[1|1]$) and u_j^T and v_j^T are given by:

$$\begin{pmatrix} a_{j(j+1)} & a_{j(j+2)} & \cdots & a_{jp} \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} a_{(j+1)j} & a_{(j+2)j} & \cdots & a_{pj} \end{pmatrix}$$

respectively. Then:

$$\chi = C_1 C_2 \cdots C_p \tag{8.2.11}$$

Therefore, the vector of the coefficients of the characteristic polynomial χ can be expressed as a product of matrices. Each of the Toeplitz matrices C_i can be computed independently of the other C_j's, so we have a sequence of $C_1, C_2, \ldots, C_p$ matrices, independently computed. The entries of each C_i can also be computed using matrix products, independently of each other. Being Toeplitz matrices, in fact one has to compute only the first column each time. Efficient methods for computing the product of Toeplitz matrices in parallel exist as indicated in [49] and a prefix method can be employed for computing the whole product.

8.2.2 The migration flow computed by the LPS algorithm

Lemma 14. *The flow computed by the LPS algorithm in (G, l, c, w) is*

$$x = -WA^T D^{-1/2} \frac{\mathcal{L}^{p-2} + a_1 \mathcal{L}^{p-3} + \ldots + a_{p-2} I}{a_{p-1}} D^{-1/2} l^{(0)}.$$

Proof. Let

$$u = (\mathcal{L}^{p-2} + a_1 \mathcal{L}^{p-3} + \ldots + a_{p-2} I)\ D^{-1/2} l^{(0)}$$

and

$$x = -\frac{1}{a_{p-1}} WA^T D^{-1/2} u.$$

For each edge (i, j), $x_{ij} = -w_{ij}/a_{p-1} \cdot (u_i/\sqrt{c_i} - u_j/\sqrt{c_j})$.
On the other hand, $u = \sum_{t=0}^{p-2} a_{p-2-t} s^{(t)}$ and $s^{(t)}$ satisfies the recurrence

$$\begin{cases} s^{(0)} & = & D^{-1/2} l^{(0)} \\ s^{(t+1)} & = & \mathcal{L} s^{(t)}, \text{ for } t \in \{0, \ldots, p-2\} \end{cases}.$$

It follows that $u = (\sum_{t=0}^{p-2} a_{p-2-t} \mathcal{L}^t) s^{(0)}$. □

Theorem 22. *The LPS algorithm generates the same balancing flow as a GDA in a given heterogeneous computational model (G, l, c, w).*

Proof. Let f be the minimal balancing flow with respect to $\|\cdot\|_{2,W^{-1/2}}$ in (G, l, c, W) and x the flow computed by the above algorithm based on the characteristic polynomial of generalized Laplacian.

Using the relations (8.2.2) and (8.2.3) one gets

$$\begin{aligned} f - x &= WA^T D^{-1/2} \left[(\mathcal{L} + c^{1/2} c^{T\,1/2})^{-1} + \frac{\mathcal{L}^{p-2} + a_1 \mathcal{L}^{p-3} + \ldots + a_{p-2} I}{a_{p-1}} \right] D^{-1/2} l^{(0)} \\ &= WA^T D^{-1/2} \left[\frac{\mathcal{L}^{p-1} + a_1 \mathcal{L}^{p-2} + \ldots + a_{p-1}}{a_{p-1}} (\mathcal{L} + c^{1/2} c^{T\,1/2})^{-1} \right] D^{-1/2} l^{(0)} \\ &= WA^T D^{-1/2} c^{1/2} c^{T\,1/2} D^{-1/2} l^{(0)} = WA^T e e^T l^{(0)} = 0. \end{aligned}$$

Therefore, the two flows are identical. □

As a consequence, we have immediately:

Corollary 3. *The LPS algorithm computes the unique minimal balancing flow with respect to the norm* $\|\cdot\|_{2,W^{-1/2}}$ *in* (G, l, c, W).

8.2.3 Experiments with LPS

The LPS algorithm is effective when the communication topology as well as the capacities of the processors do not vary often at runtime. The modification of the communication patterns, of the network parameters as well as the changes of capacities impose the re-computation of the coefficients of the Laplacian polynomial. The LPS algorithm requires only $p-1$ communication phases to balance the system, provided the coefficients of the Laplacian polynomial are known, whereas the generalized diffusion algorithm needs $O((c_{\max}\,\Delta^w\,p^2)/(c_{\min}\,w_{\min}\,e(G))$ steps. Furthermore, it requires only communication between neighbors. Large transfers along the edges of the communication graph for which the communication is costly can be prohibited by choosing the weight of the corresponding link inverse proportionally to the real cost of communication between the corresponding processors. The algorithm is comparable with the optimal scheme described by Diekmann et al. [15] and adapted by Elsässer et al. [18] what the number of steps is concerned and both present no global communication. However, their scheme requires the computation of the spectra of the Laplacian matrix and no efficient method was specified for doing this, what is needed for applying it in a dynamic context. The LPS algorithm requires the computation of the coefficients of the Laplacian polynomial but efficient ways of how to efficiently compute them in parallel were indicated.

The described algorithm generates a balancing flow that has the property that it is the scaled projection of all other balancing flows and that minimizes a weighted 2-norm. We implemented and

Hostname	CPU	memory	model
jargon	110 MHz	64 MB	SPARCStation-4
opale	70 MHz	32 MB	SPARCStation-5
outremer	110 MHz	64 MB	SPARCStation-4
peridot	167 MHz	128 MB	UltraSPARC
sanguine	200 MHz	64 MB	UltraSPARC
spinelle	70 MHz	32 MB	SPARCStation-5
turquoise	70 MHz	32 MB	SPARCStation-5
zircon	70 MHz	64 MB	SPARCStation-5
girasol	75 MHz	64 MB	SPARCStation-20
cristal	75 MHz	64MB	SPARCStation-4
tourmaline	75 MHz	64MB	SPARCStation-4

Table 8.1: heterogeneous cluster of SUN workstations used

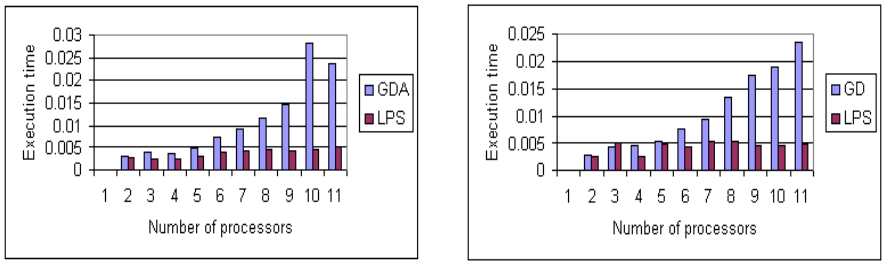


Figure 8.2: Execution time in seconds for GDA and LPS on paths (in the left figure) and on rings (in the right figure)

run the described technique on a heterogeneous cluster of SUN workstations running SunOS 5.7 with the structure given in table 8.1. In figure 8.2 execution times for GDA1 and LPS for communication topologies of type path and ring of length up to 11 are shown.

Tests were also performed for various subdomain graphs obtained using the Metis [53] library. In figure 8.3 the execution times (in seconds) for GDA1 and LPS on subdomain graphs of some well known graphs.

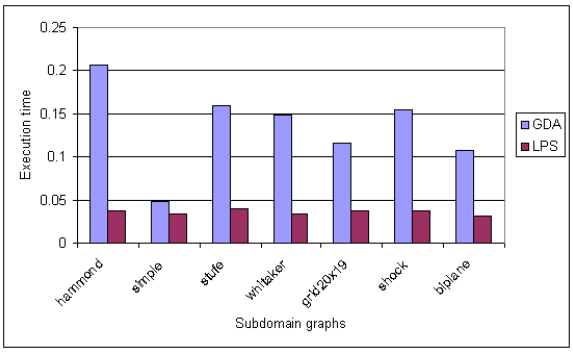


Figure 8.3: Execution time in seconds for GDA and LPS on a heterogeneous cluster of 11 processors for various subdomain graphs

8.3 An improved Laplacian polynomial based algorithm

Examining the explicit expression of the unique balancing flow that is the solution of (8.1.1), one can remark that ϕ can be put in the form $\phi = \Theta l$, with

$$\Theta = W A^T D^{-1/2} (\mathcal{L} + c^{1/2} c^{T^{1/2}})^{-1} D^{-1/2} = W A^T D^{-1} (L D^{-1} + c e^T)^{-1}.$$

We use the following alternative characterization for Θ [68]:

Theorem 23. *Let $a_1, \ldots, a_p$ be the coefficients of the characteristic polynomial of LD^{-1} and $\Theta = W A^T D^{-1} (L D^{-1} + c e^T)^{-1}$. Then Θ can also be expressed in the following equivalent form:*

$$\Theta = -W A^T D^{-1} U,$$

with

$$U = \frac{(L D^{-1})^{p-2} + a_1 (L D^{-1})^{p-3} + \ldots + a_{p-2} I}{a_{p-1}}.$$

The coefficients of characteristic polynomial may be computed directly using the Fadeev-LeVerrier algorithm [60]:

We propose in the sequel an efficient parallel algorithm for computing the minimal balancing flow in a dynamic heterogeneous environment. A SPMD model is used and the following local data structures are used by the processors:

1. *rank* - rank of the processor itself

Algorithm 11 The Fadeev-LeVerrier algorithm

$P^1 = L\,D^{-1}$

for $i = 1, p$ **do**

$a_i =$ `-trace`$(P^i)/i$

$P^{i+1} = (P^i + a_i\,I)\,LD^{-1}$.

end for

2. *Load* - workload of the processor itself
3. *Cap* - capacity of the processor itself
4. *arr*[0..p-1] - local array that denotes the *rank*-th line in U
5. *arrS*[0..p-1] - backup local copy of *arr*
6. *nbNbrs* - number of logical neighbors (the degree of the processor in the communication graph) of the processor itself
7. *arrNb*[0..nbNbrs-1] - array of the neighbors' ids
8. *CapNb*[0..nbNbrs-1] - array of the capacities of neighboring processors
9. *eWght*[0..nbNbrs-1] - array of weights representing communication costs along the links with the neighbors
10. *Flow*[0..nbNbrs-1] - array of flows to be sent/received to/from the neighbors
11. *arrLoad*[0..p-1] - array of the processors' loads
12. **allgather** $(larr, lv)$ - denotes a global operation of type allgather: all processors gather the local values lv in the array $larr$, the value received from a processor i being stored in $larr[i]$

With these settings, the parallel method that we propose is described in algorithm 12.

In a distributed environment, the communication is strongly influenced by two network parameters: the latency and the bandwidth. Generally, it is assumed that the communication time between two processors i and j obeys the following law:

$$T_{ij}^{comm} = \alpha_{ij} + \beta_{ij}\,L_{ij},$$

Algorithm 12 LPA

for $(i = 0; i < p; i = i + 1)$ **do**

$arr[i] = 0$

end for

$arr[rank] = 1$

for $(k = 0; k < p; k = k + 1)$ **do**

if $(k == p - 2)$ **then**

for $(i = 0; i < p; i = i + 1)$ **do**

$arrS[i] = arr[i]$

end for

end if

send arr to neighbors

receive from neighbors the local copies of arr, in $arrNb_0, \ldots, arrNb_{nbNbrs-1}$

for $(i = 0; i < p; i = i + 1)$ **do**

$\alpha = arr[i]/Cap$

for all neighbors j **do**

$arr[i] = arr[i] + eWght[j] \cdot (\alpha - arrNb_j[i]/CapNb_j)$

end for

end for

allreduce $(arr[myrank], T, SUM)$

$arr[myrank] = arr[myrank] - T/(k + 1)$

end for

for $(i = 0; i < p; i = i + 1)$ **do**

$arr[i] = -(k + 1)/T \cdot arrS[i]$

end for

for $(i = 0; i < p; i = i + 1$ **do**

$arrS[i] = arrS[i]/(T \cdot Cap)$

end for

send $arrS$ to neighbors

receive from neighbors the local copies of $arrS$, in $arrNb_0, \ldots, arrNb_{nbNbrs-1}$

allgather $(arrLoad, Load)$

for all $neighbors\ j$ **do**

for $(i = 0; i < p; i = i + 1)$ **do**

$Flow[j]^{new} = Flow[j]^{new} + eWght[j] \cdot (arrS[i] - arrNb_j[i]) \cdot arrLoad[i]$

end for

end for

where

- T_{ij}^{comm} denotes the communication time for transferring L_{ij} bytes between the processors i and j.
- α_{ij} is the latency of the communication between i and j.
- $\beta_{ij} = 1/b_{ij}$, where b_{ij} is the communication bandwidth between i and j.
- L_{ij} is the data size expressed in bytes.

The redistribution time between two logically adjacent processors i and j can be estimated as

$$T_{ij} = \alpha_{ij} M_{ij} + \gamma \beta_{ij} |\phi_{ij}|$$

where

- α_{ij}, β_{ij} have the same meaning as above:
- $|\phi_{ij}|$ is the absolute value of the balancing flow to be transferred between the processors i, j;
- M_{ij} represents the number of messages exchanged between the processors i and j.
- γ is a constant.

We shall make the specification that in the course of the implementation we have made use of non blocking point to point communication primitives.

Thus, the total redistribution time is

$$T_r = \sum_{ij \in E} \left(\alpha_{ij} M_{ij} + \gamma \beta_{ij} |\phi_{ij}| \right).$$

On the other hand, the above algorithm minimize the quantity $\|\phi\|_{W^{-1/2}}$. One may consider $w_k = 1/\beta_{ij}^2$, where i an j denote the extremities of the edge e_k. Therefore, the above algorithms minimize $\sum_{k=1}^{m} \phi_k^2 / w_k$. By the inequality of Cauchy-Schwarz-Bunyakowski one has

$$q \sum_{k=1}^{m} \frac{\phi_k^2}{w_k} \geq \sum_{k=1}^{m} \left(\beta_k |\phi_k| \right)^2.$$

This means that the above algorithms minimize an upper bound for the total redistribution time. We used as experimental environment a computational grid that was built using commodity hardware

Machine	OS	processor	CPU	RAM
PC151,..,PC166	Solaris 2.7	i386	166 Mhz	32 M
peridot	Solaris 2.7	sparcv9	167 MHz	64 M
jargon	Solaris 2.7	sparc	110 MHz	64 M
outremer	Solaris 2.7	sparc	110 MHz	64 M
zircon	Solaris 2.8	sparc	70 MHz	32 M
tourmaline	Solaris 2.8	sparc	60 MHz	32 M
girasol	Solaris 2.8	sparc	75 MHz	32 M

Table 8.2: Heterogeneous computing environment used for performing tests.

Machines	normalized relative speed
PC151,...,PC166	0.0358
peridot	0.1201
jargon	0.0843
outremer	0.0765
zircon	0.0358
tourmaline	0.0627
girasol	0.0560

Table 8.3: Heterogeneous computing environment used for performing tests.

available in our institute. It consists of a collection of heterogeneous machines as indicated in table 8.2. Table 8.3 shows the normalized relative speeds of these machines. LAM/MPI and Network Weather Service 2.0.6 [92] (NWS) were installed. NWS provided online values for network and processor parameters. These measurements were afterwards merged into values of the parameters of the assumed theoretical model.

8.4 Experiments with PCG and LPA

We compared generalized diffusion with PCG and LPA algorithms. Several communication topologies were considered. Generalized diffusion algorithms based on the following diffusion operators

Table 8.4: The initial loads assigned to processors

$load_1 = 600$	$load_7 = 2000$	$load_{13} = 3000$	$load_{19} = 500$
$load_2 = 3000$	$load_8 = 3000$	$load_{14} = 4000$	$load_{20} = 7000$
$load_3 = 600$	$load_9 = 4000$	$load_{15} = 1000$	$load_{21} = 500$
$load_4 = 2000$	$load_{10} = 5000$	$load_{16} = 2000$	$load_{22} = 800$
$load_5 = 300$	$load_{11} = 200$	$load_{17} = 1000$	
$load_6 = 1000$	$load_{12} = 200$	$load_{18} = 1000$	

were used:

$$m_{ij}^1(\epsilon) = \begin{cases} \min\left\{\frac{c_i}{\delta_i^w+\epsilon}\frac{c_j}{\delta_j^w+\epsilon}\right\}\frac{w_k}{c_j}, \text{ if } e_k=\{ij\}\in E, \\ 0, \text{ if } ij\notin E, i\neq j, \\ 1-\sum_{j:ji\in E} m_{ji}, \text{ , if } i=j. \end{cases} \quad (8.4.1)$$

$$m_{ij}^2(\epsilon) = \begin{cases} \frac{c_{\min}}{c_j}\frac{w_k}{\Delta^w+\epsilon}, \text{ if } e_k=\{ij\}\in E, \\ 0, \text{ if } ij\notin E, i\neq j, \\ 1-\sum_{j:ji\in E} m_{ji}, \text{ , if } i=j. \end{cases} \quad (8.4.2)$$

$$m_{ij}^3(\epsilon) = \begin{cases} \sqrt{\frac{c_{\min}}{c_{\max}}}\sqrt{\frac{c_i}{c_j}}\frac{w_k}{\Delta^w+\epsilon} \text{ ,if } e_k=\{ij\}\in E, \\ 0, \text{ if } ij\notin E, i\neq j, \\ 1-\sum_{j:ji\in E} m_{ji}, \text{ , if } i=j. \end{cases} \quad (8.4.3)$$

$$m_{ij}^4(\epsilon) = \begin{cases} \frac{c_i}{c_{\max}}\frac{w_k}{\Delta^w+\epsilon}, \text{ if } e_k=\{ij\}\in E, \\ 0, \text{ if } ij\notin E, i\neq j, \\ 1-\sum_{j:ji\in E} m_{ji}, \text{ , if } i=j. \end{cases} \quad (8.4.4)$$

$$m_{ij}^5(\epsilon) = \begin{cases} \frac{c_i c_j}{w_k}, \text{ if } e_k=\{ij\}\in E, \\ 0, \text{ if } ij\notin E, i\neq j, \\ 1-\sum_{j:ji\in E} m_{ji}, \text{ , if } i=j. \end{cases} \quad (8.4.5)$$

Tables 8.5 and 8.6 report results in terms of execution times (expressed in seconds) and the number of iterations for a load distribution as expressed in table 8.4 and a tolerated convergence error error $= 0.01$. The first column corresponds to the algorithm described in the previous section and the other five to the diffusion algorithms based on the diffusion matrices described above. If we express the load imbalance as $\max_i\{l_i/\bar{l}_i\}$, the above load distribution is characterized by an imbalance factor of 3.27.

The figures below report execution times for different communication topologies when the number of processors vary, for a load imbalance factor of 3.2. We further considered the efficiency defined

Topology	LPA	GD1	GD2	GD3	GD4	GD5
Path	0.32	19.82	20.41	37.48	66.73	300.24
Ring	0.27	6.69	6.25	11.5	18.67	80.06
Star	0.20	8.79	8.96	9.60	9.93	4.00

Table 8.5: Execution times for different communication topologies and redistribution methods on the heterogeneous environment described in 8.2.

Topology	LPA	GD1	GD2	GD3	GD4	GD5
Path	21	1828	1950	3284	5678	23898
Ring	21	512	521	879	1537	6390
Star	21	961	983	995	1086	425

Table 8.6: Number of iterations until convergence of the considered redistribution methods on the heterogeneous environment described in 8.2.

as the ratio $T_{p,1}/(pT_{p,p})$, where $T_{p,1}$ is the time the algorithm takes for solving the problem of size p on a single processor, and $T_{p,p}$ is the time the algorithm takes for solving the problem of size p on p processors.

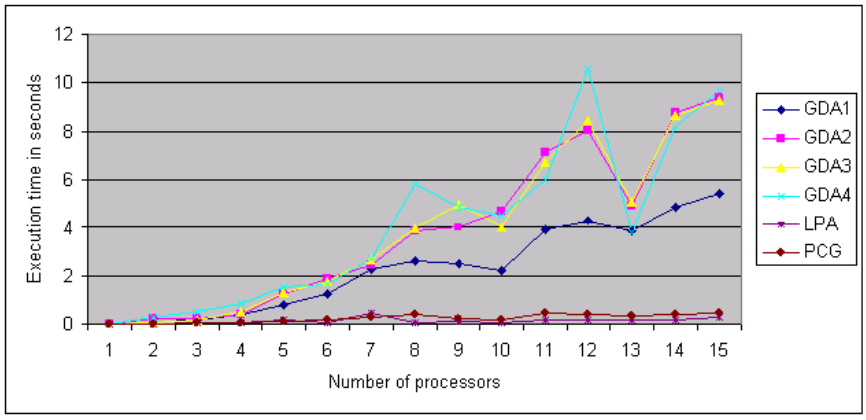


Figure 8.4: Execution times of the tested algorithms when the communication topology is a binary tree.

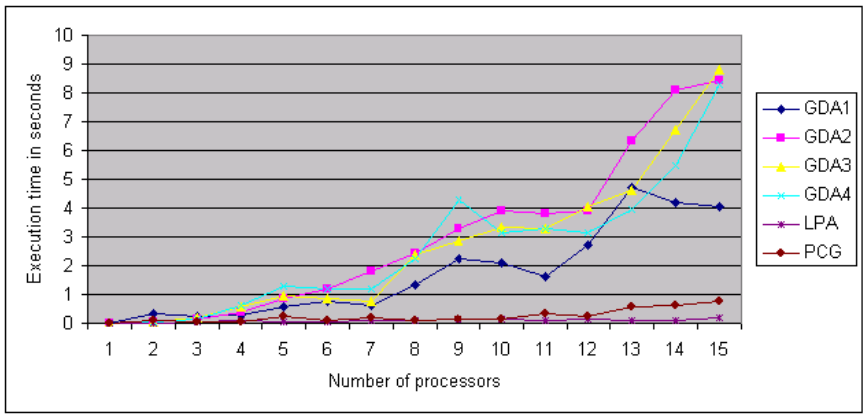


Figure 8.5: Execution times of the tested algorithms when the communication topology is a path.

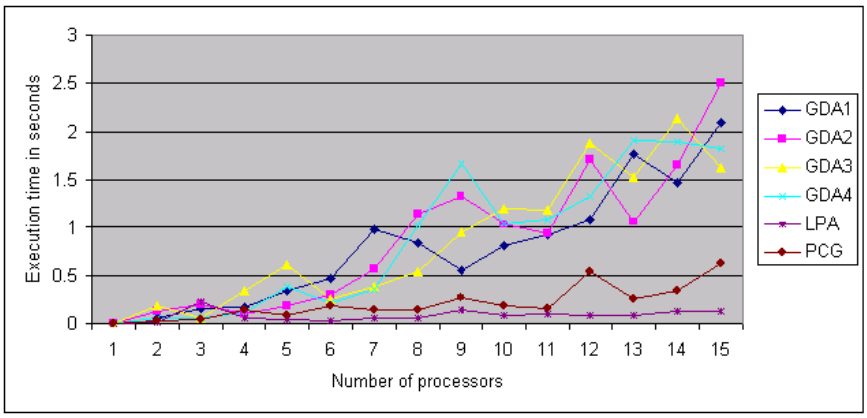


Figure 8.6: Execution times of the tested algorithms when the communication topology is a ring.

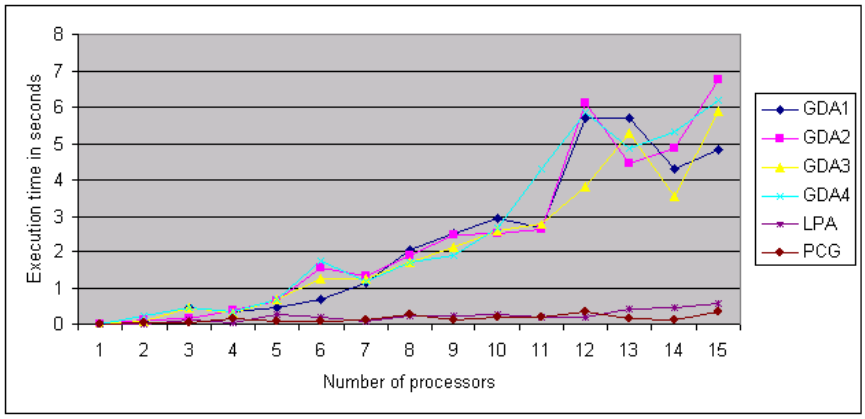


Figure 8.7: Execution times of the tested algorithms when the communication topology is of type star.

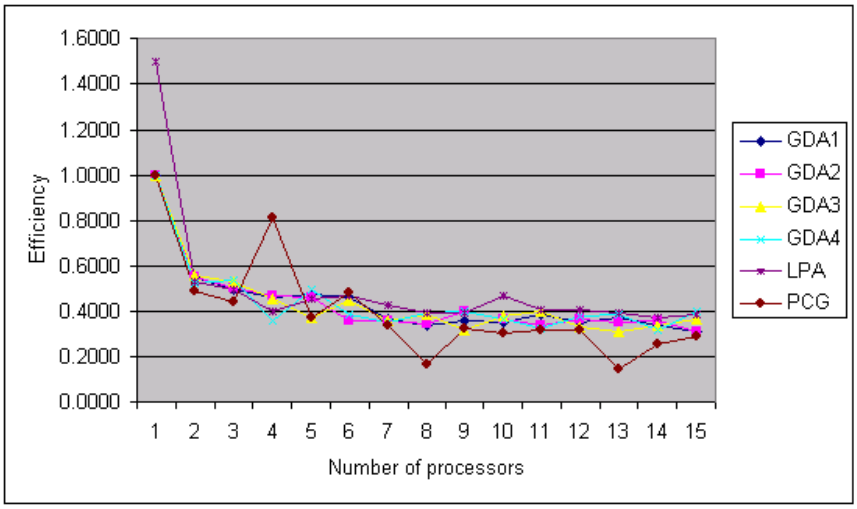


Figure 8.8: Efficiency of the tested algorithms when the communication topology is of type path.

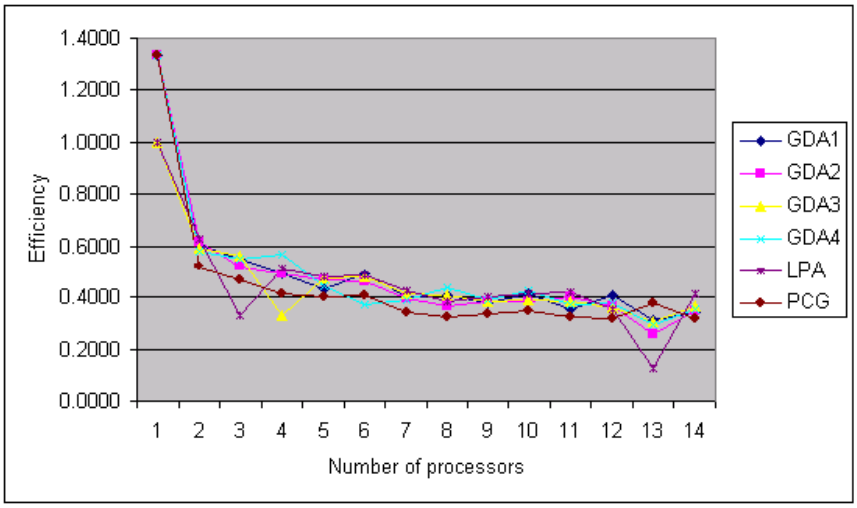


Figure 8.9: Efficiency of the tested algorithms when the communication topology is of type ring.

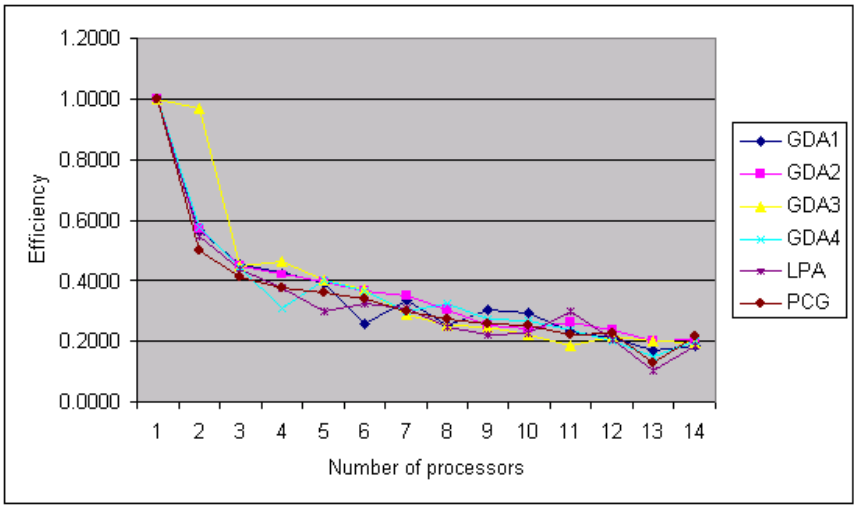


Figure 8.10: Efficiency of the tested algorithms when the communication topology is of type star.

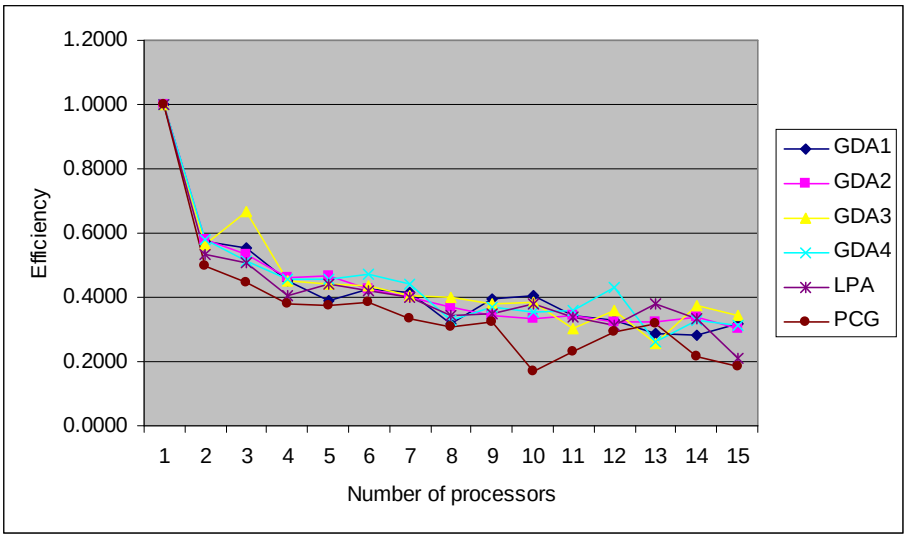


Figure 8.11: Efficiency of the tested algorithms when the communication topology is a binary tree.

8.5 An algorithm based on the minimum polynomial

In the previous sections we gave an algorithm based on the Laplacian polynomial that is better than the generalized diffusion algorithm (in terms of number of communication phases) and than the algorithm based on the conjugate gradient method (in terms of number of number of global operations). The analysis carried out with the algorithm LPA did not make use of any special properties of the characteristic polynomial, but only of the fact that $\mathcal{L}$ nullifies it. One may take advantage of this by using a polynomial with a lower degree, in order to speed up the algorithm. It is therefore natural to use the *minimum generating polynomial* of the generalized Laplacian matrix. Clearly, the minimum polynomial is a divisor of the Laplacian polynomial. Therefore, the related dynamic load balancing methods may be faster, at least in the case when the degree of the minimum polynomial is significantly lower than that of the characteristic polynomial. In this section we present an algorithm for dynamic load balancing that relies on the minimum generating polynomial of the Laplacian matrix. This algorithm relies upon an idea of Wiedemann [90].

Let V be a vector space over the field K, and let $\{a_i\}_{i=0}^{\infty}$ be an infinite sequence with elements $a_i \in V$. The sequence $\{a_i\}_{i=0}^{\infty}$ is linearly generated over K if there exist $c_0, c_1, \ldots c_n \in K$ with $n \geq 0$ and $c_k \neq 0$ for some integer $k \in [0..n]$, such that

$$c_0 a_j + \ldots + c_n a_{j+n} = 0, \forall j \geq 0.$$

If $\{a_i\}_{i=0}^{\infty}$ is a linearly generated sequence, then the polynomial $c_0 x^n + c_1 x^{n-1} + \ldots + c_{n-1} x + c_n$ is called a generating polynomial for this sequence. There exists an unique monic generator of minimal degree of a sequence $\{a_i\}_{i=0}^{\infty}$ and any other generating polynomial is a multiple of this minimum polynomial.

Let us consider now $A \in K^{n \times n}$ a square matrix over a field K. Conforming to the Cayley-Hamilton theorem, there is a non-null polynomial χ, the characteristic polynomial, such that $\chi(A) = 0$. Following the above definition, this means that the sequence $\{A^i\}_{i=0}^{\infty}$ is linearly generated. Its minimum generating polynomial is the minimum polynomial of A and will be denoted f^A. For any column vector b, the sequence $\{A^i b\}_{i=0}^{\infty}$ is also generated by f^A and its minimum generating polynomial $f^{A,b}$ is a divisor of f^A. For a column vector u, the sequence $\{u^T A^i b\}_{i=0}^{\infty}$ is linearly generated as well, and its minimum generating polynomial, denoted $f_u^{A,b}$ is a divisor of $f^{A,b}$.

Wiedemann presented an algorithm for solving a sparse linear system [90]. Relying upon the same idea, Kaltofen and Saunders gave an algorithm for computing the minimum generating polynomial of a sequence $\{A^i b\}_{i=0}^{\infty}$ that is identical to that described in the algorithm 13 [52].

Algorithm 13 Algorithm Minimum Polynomial

Input $A \in R^{n \times n}, b \in R^n$, and $d = n$

Output $f^{A,b}$

Step 1: Pick a random vector u and compute

$a_0 \longleftarrow ub, a_1 \longleftarrow uAb, \ldots, a_2 \longleftarrow uA^k b, \ldots \longleftarrow uA^{2d-1}b.$

Step 2: Determine $f_u^{A,b}$ by applying the Berlekamp/Massey algorithm

to the sequence $a_0, \ldots, a_{2d-1}$

Step 3: Check if $f_u^{A,b}$ is a proper divisor of $f^{A,b}$

if $(d = \deg(f_u^{A,b}))$ **then**

  Return $f^{A,b} \longleftarrow f_u^{A,b}$

else

  $b' \longleftarrow f_u^{A,b}(A)\, b$

  if $(b' = 0)$ **then**

    Return $f^{A,b} \longleftarrow f_u^{A,b}$

  else

    Call the algorithm recursively with A, b' and $d - \deg(f_u^{A,b})$

    Return $f^{A,b} \longleftarrow f_u^{A,b} \times f^{A,b'}$

  end if

end if

Algorithm 13 requires the application of the Berlekamp/Massey algorithm that is described in algorithm 14 [59]. Originally, this was used in relation with the decoding of Reed-Solomon (RS), and more generally, Bose-Chaudhuri-Hocquenghem (BCH) block error-control codes. For an input sequence of size n it has time complexity $O(2n)$.

We shall use these ideas for deriving a parallel algorithm that computes a minimal balancing flow. First of all, let $\alpha_0 X^k + \alpha_1 X^{k-1} + \ldots + \alpha_{k-1} X + \alpha_k$ be the minimum generating polynomial of the sequence $\{b_i\}_{i=0}^{\infty}$, with $b_i = (LD^{-1})^i l$. Without loss of generality, we may assume that $\alpha_0 = 1$.

Algorithm 14 The Berlekamp/Massey algorithm

$\sigma(X) = 1$

$\beta(X) = 0$

$l = 0$

for $k = 1$ to $2\,n$ **do**

  $d = \sum_{i=0}^{l} \sigma_i \, S_{k-i}$

  if $(d \neq 0)$ **then**

    $\sigma' = \sigma(X) - d\,X\,\beta(X)$

    if $(2l < k)$ **then**

      $\beta(X) = \delta^{-1}\,\sigma(X)$

      $l = k - l$

    else

      $\beta(X) = X\,\beta(X)$

    end if

    $\sigma(X) = \sigma'(X)$

  else

    $\beta(X) = X\,\beta(X)$

  end if

end for

Then, we have that

$$\alpha_0 b_k + \alpha_1 b_{k-1} + \ldots + \alpha_{k-1} b_1 + \alpha_k b_0 = 0.$$

Two remarks should be made. First, because e^T is a left eigenvector that corresponds to the smallest eigenvalue 0 of $L = AWA^T$, necessarily α_k should be 0. Second, we have that $\alpha_{k-1} \neq 0$, because LD^{-1} has only nonnegative eigenvalues and the eigenvalue 0 has the algebraic multiplicity 1 (G being connected) and because the minimum polynomial divides the characteristic polynomial. As the array of capacities c is a right eigenvector of LD^{-1} corresponding to the eigenvalue 0, we deduce that

$$\alpha_0 b_{k-1} + \alpha_1 b_{k-2} + \ldots + \alpha_{k-1} b_0 = \beta c,$$

with $\beta = \alpha_{k-1} e^T l$.

Therefore, we have

$$A\phi = l - \overline{l},$$

where

$$\phi = -WA^T D^{-1} \frac{\alpha_0 b_{k-2} + \alpha_1 b_{k-3} + \ldots + \alpha_{k-2} b_0}{\alpha_{k-1}}.$$

As in the case of the characteristic polynomial, it can be shown similarly that ϕ is the unique minimal balancing flow that minimizes the norm $\| \cdot \|_{W^{-1/2}}$.

In the sequel we describe a parallel algorithm for dynamic load balancing using the minimum generating polynomial. As usually, we use a SPMD model. The data structures used by each task are the following:

1. *rank* - the index of the processor itself
2. *Load* - workload of the processor itself
3. *Cap* - capacity of the processor itself
4. *nbNbrs* - number of neighbors in the communication graph of the processor itself
5. *eWght*[0..nbNbrs-1] - array of weights representing communication costs along the links with the neighbors
6. *Flow*[0..nbNbrs-1] - array of flows to be sent/received to/from the neighbors

7. Deg - degree of the minimum polynomial
8. $arrC[0..Deg-1]$ - array of the coefficients of the minimum polynomial
9. $gidNghb$ - global index of a neighbor
10. $arrRecv[0..nbNbrs-1]$ - array used for receiving temporary information received from the neighbors
11. $v[0..2p]$ - array whose entries are defined as $v[i]$ = the $rank$-th entry of the matrix vector product $LD^{-1}l$
12. u - local potential
13. $S[0..2p]$ - auxiliary array

With these settings, the parallel algorithm MPA is identical to that described in the algorithm 15.

Algorithm 15 MPA

$sumWghts = 0$

for $(j = 0; j < nbNbrs; j = j + 1)$ **do**

$sumWghts = sumWghts + eWght[j]$

end for

$v[0] = Load$

for $(k = 0; k < 2p; k = k + 1)$ **do**

$vsend = v[k-1]/Cap$

$v[k] = vsend \cdot sumWghts$

send $vsend$ to neighbors

receive from neighbors the $vsend$ values in $arrRecv_0, \ldots, arrRecv_{nbNbrs-1}$

for $(j = 0; j < p; j = j + 1)$ **do**

$v[k] = v[k] - eWght[j] \cdot arrRecv[j]$

end for

end for

barrier

gather the local arrays $v[0..2p+1]$ in $gv[0..(2p+1)p]$

for $(i = 0; i <= 2p; i = i + 1)$ **do**

$S[i] = 0.0$

for $(j = 0; j < p; j = j + 1)$ **do**

$S[i] = S[i] + gu[j] \cdot gv[(2p+1)j + i]$

end for

end for

$Deg =$ `BerlekampMassey`$(S, p, \&arrC)$

$u = 0.0$

for $(i = 0; i < Deg - 1; i = i + 1)$ **do**

$u = u + arrC[i] \cdot v[Deg - 2 - i]$

end for

$u = u/(arrC[Deg-1] \cdot Cap)$

send u to neighbors

receive the local arrays u from neighbors in $arrRecv_0, \ldots, arrRecv_{nbNbrs-1}$

for $(j = 0; j < p; j = j + 1)$ **do**

$Flow[j] = -eWght[j] \cdot (u - arrRecv[j])$

end for

topology	diam	Δ	γ
path	15	2	0.13
ring	7	2	0.14
binary tree	3	3	0.13
quaternary tree	2	5	0.13
star	2	14	0.13
wheel	2	14	0.26
complete bipartite	2	8	0.53
bridge	3	8	0.57
complete	1	14	1.00

Table 8.7: Characteristics of the tested communication topologies.

8.6 Experiments

We performed experiments with PCG, LPA and MPA for a variety of communication topologies. We considered graphs that have different characteristics. More specifically, we took into consideration the diameter, the maximum degree and the connectivity. The *connectivity* of a graph $G = (V, E)$ with n vertices is defined as

$$\gamma = \frac{2|E|}{n(n-1)}.$$

Table 8.7 gives the type of communication topologies that were used during the tests, together with their characteristics, for a fixed number of 15 vertices.

Each of the algorithms PCG, LPA, MPA was executed 50 times for all of the communication topologies indicated in table 8.7 when the number of processors varied from 4 to 16. In the following figures we report for each type of the topologies indicated in table 8.7 the average execution times, the median values and the standard deviation values for 50 executions of the algorithms PCG, LPA and MPA. If $t_1, \ldots, t_N$ are the execution times of an algorithm for N runnings, the average execution time is

$$\bar{t} = \frac{1}{N} \sum_{i=1}^{N} t_i$$

and the standard deviation is

$$\sqrt{\frac{\sum_{i=1}^{N}(t_i - \bar{t})^2}{N-1}} = \sqrt{\frac{N \sum_{i=1}^{N} t_i^2 - (\sum_{i=1}^{N} t_i)^2}{N(N-1)}}$$

Figure 8.12: Path

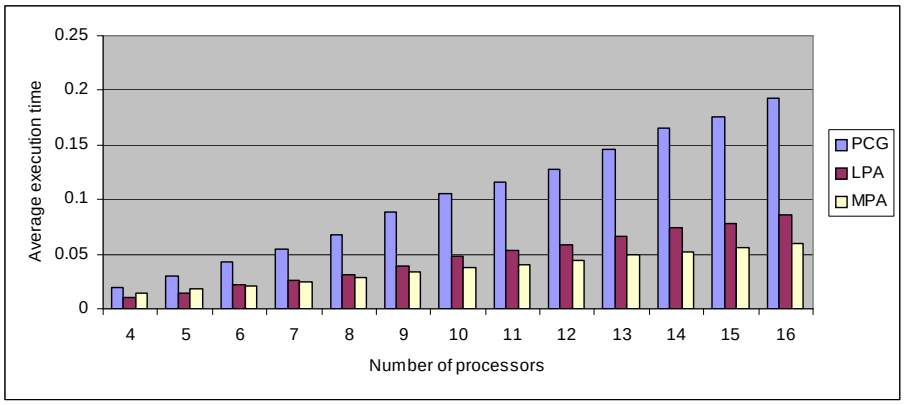


Figure 8.13: Average execution times of LPA, PCG and MPA, varying with the number of processors, when the communication topology is a path.

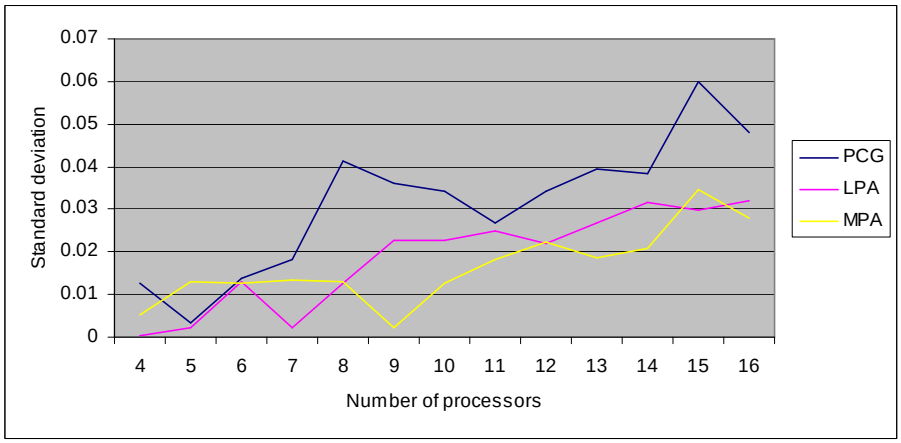


Figure 8.14: Standard deviation values for the execution times of LPA, PCG and MPA, varying with the number of processors, when the communication topology is a path.

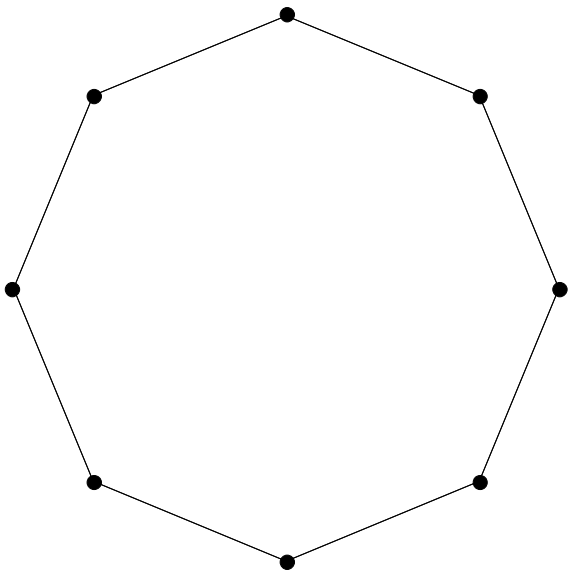


Figure 8.15: Ring

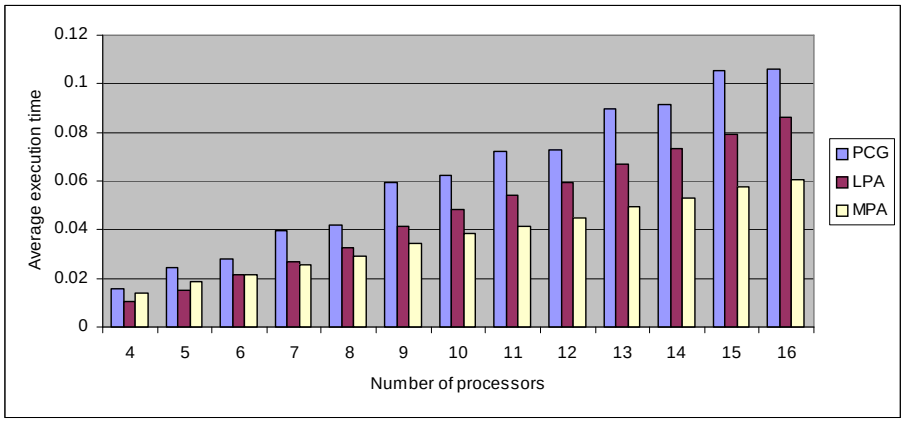


Figure 8.16: Average execution times of LPA, PCG and MPA, varying with the number of processors, when the communication topology is a ring.

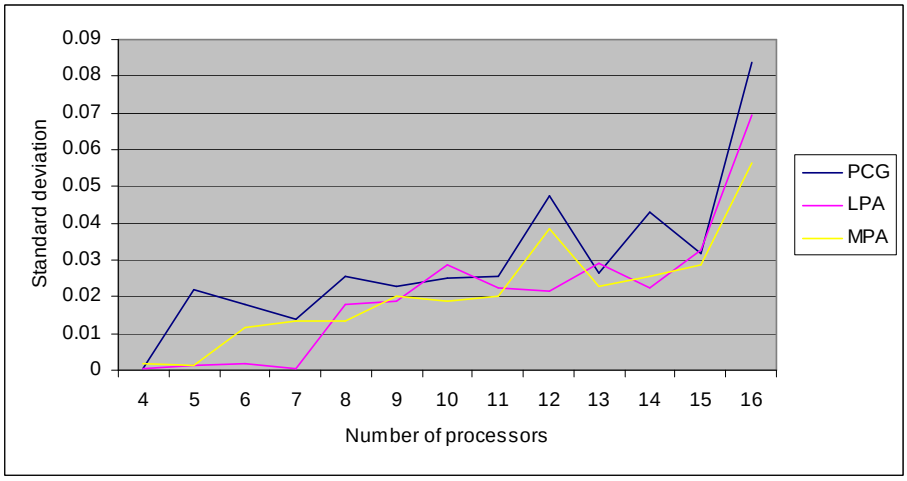


Figure 8.17: Standard deviation values for the execution times of LPA, PCG and MPA, varying with the number of processors, when the communication topology is a ring.

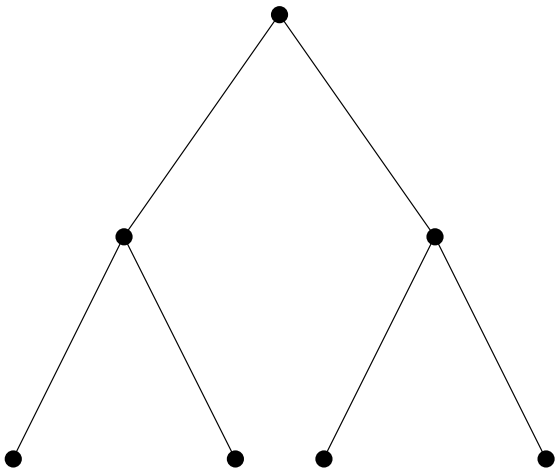


Figure 8.18: Binary tree

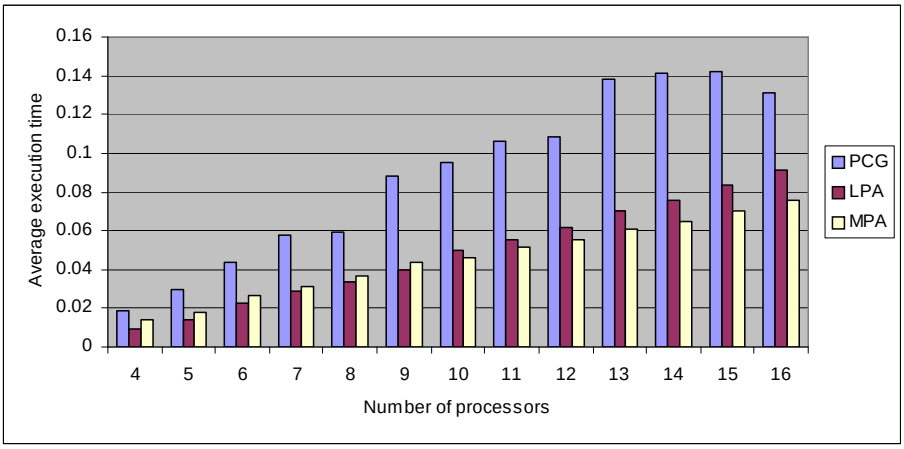


Figure 8.19: Average execution times of LPA, PCG and MPA, varying with the number of processors, when the communication topology is a binary tree.

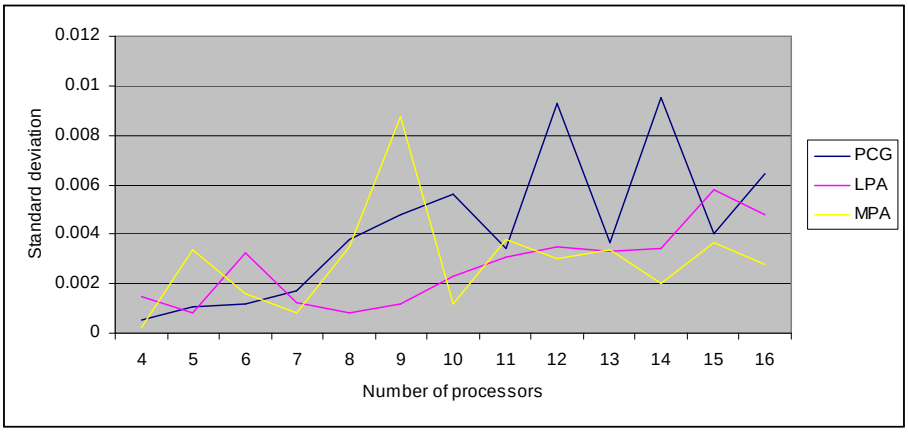


Figure 8.20: Standard deviation values for the execution times of LPA, PCG and MPA, varying with the number of processors, when the communication topology is a binary tree.

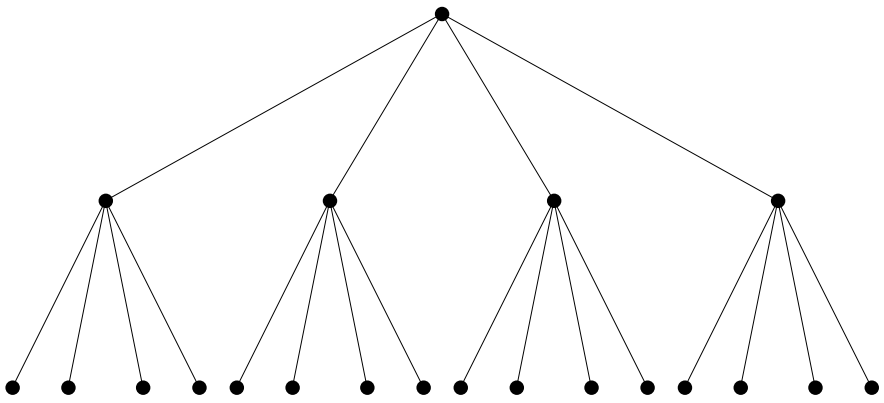


Figure 8.21: Quaternary tree

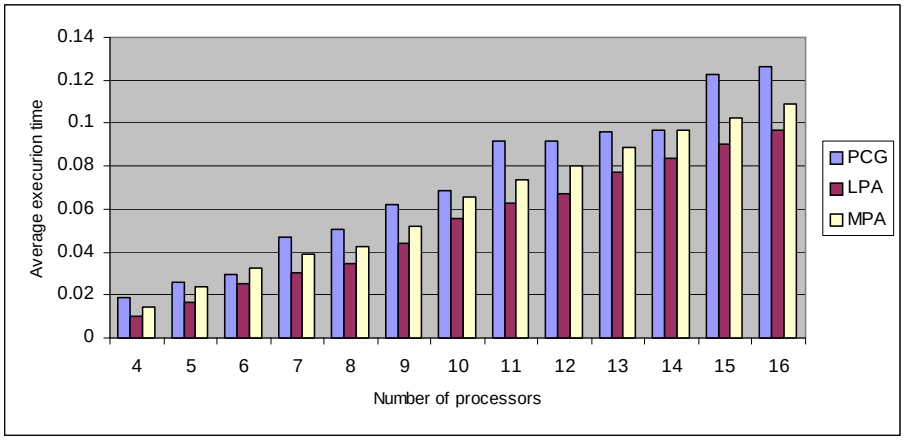


Figure 8.22: Average execution times of LPA, PCG and MPA, varying with the number of processors, when the communication topology is a quaternary tree.

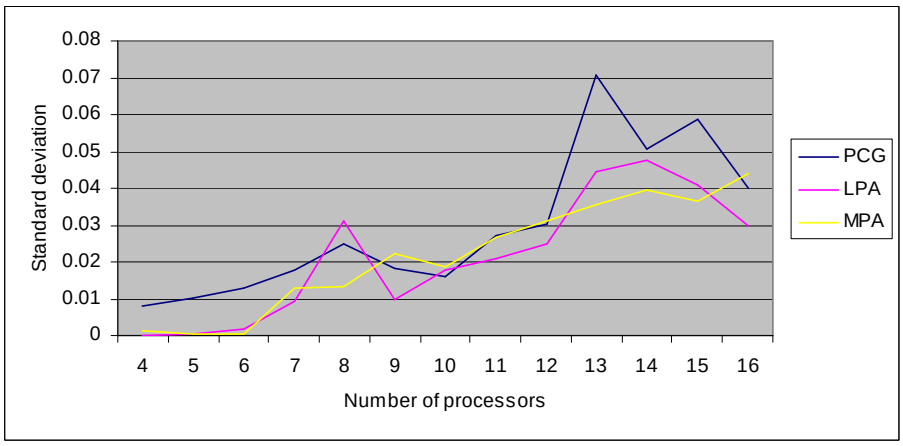


Figure 8.23: Standard deviation values for the execution times of LPA, PCG and MPA, varying with the number of processors, when the communication topology is a quaternary tree.

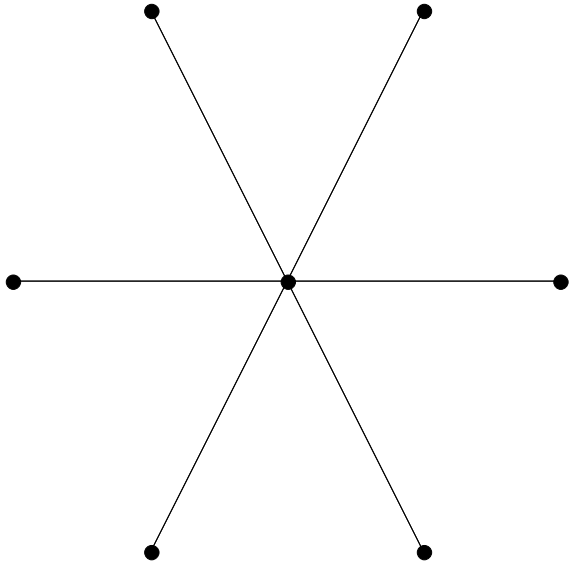


Figure 8.24: Star

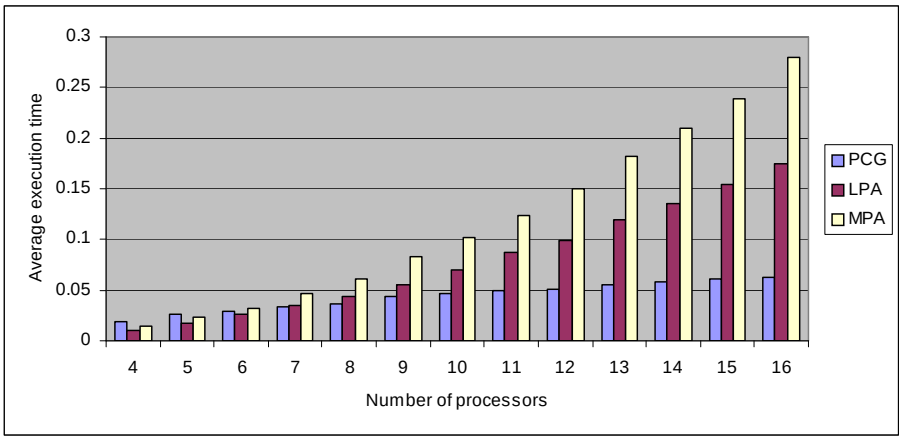


Figure 8.25: Average execution times of LPA, PCG and MPA, varying with the number of processors, when the communication topology is a star.

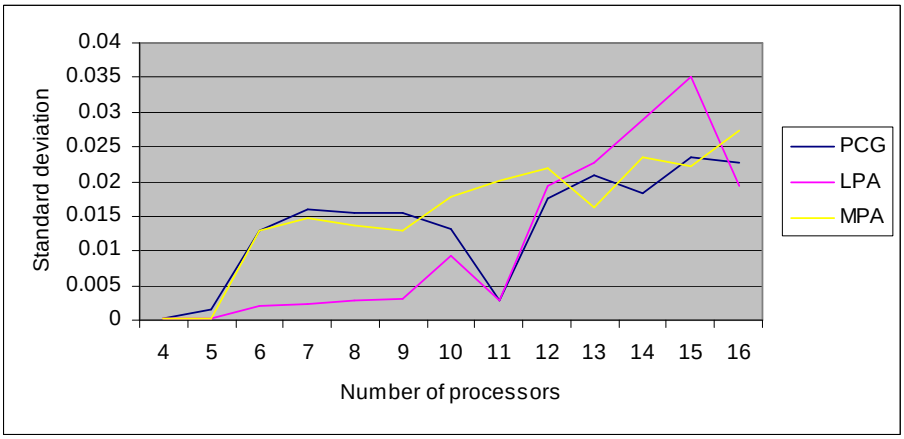


Figure 8.26: Standard deviation values for the execution times of LPA, PCG and MPA, varying with the number of processors, when the communication topology is a star.

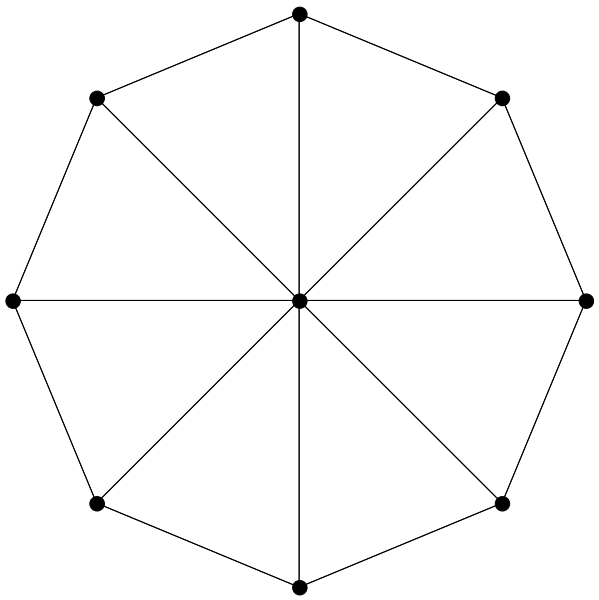


Figure 8.27: Wheel

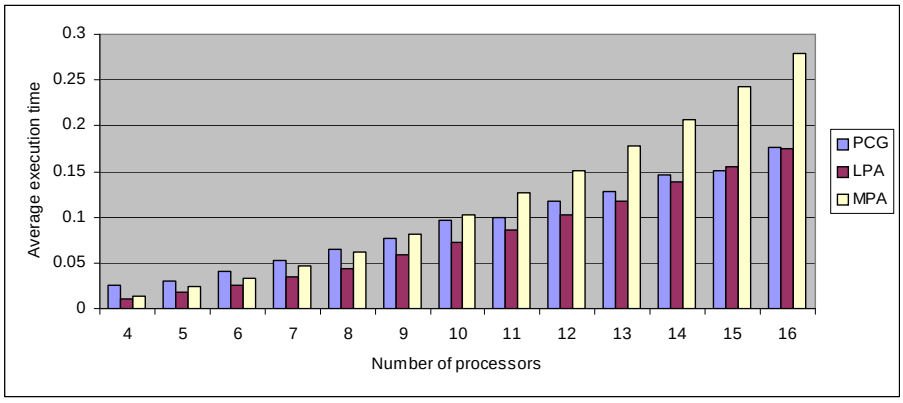


Figure 8.28: Execution times of LPA, PCG and MPA, varying with the number of processors, when the communication topology is a wheel graph.

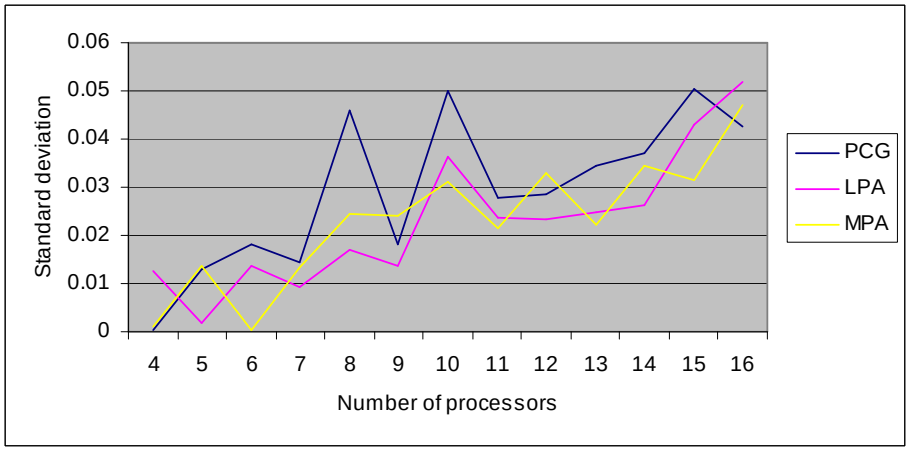


Figure 8.29: Standard deviation values for the execution times of LPA, PCG and MPA, varying with the number of processors, when the communication topology is a complete wheel graph.

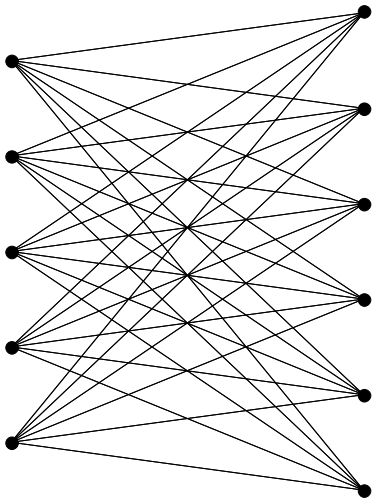


Figure 8.30: Complete bipartite graph.

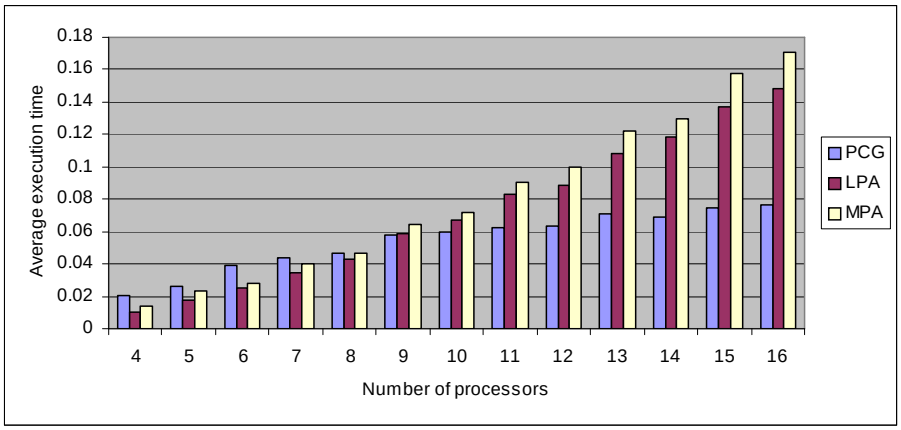


Figure 8.31: Average execution times of LPA, PCG and MPA, varying with the number of processors, when the communication topology is a complete bipartite graph.

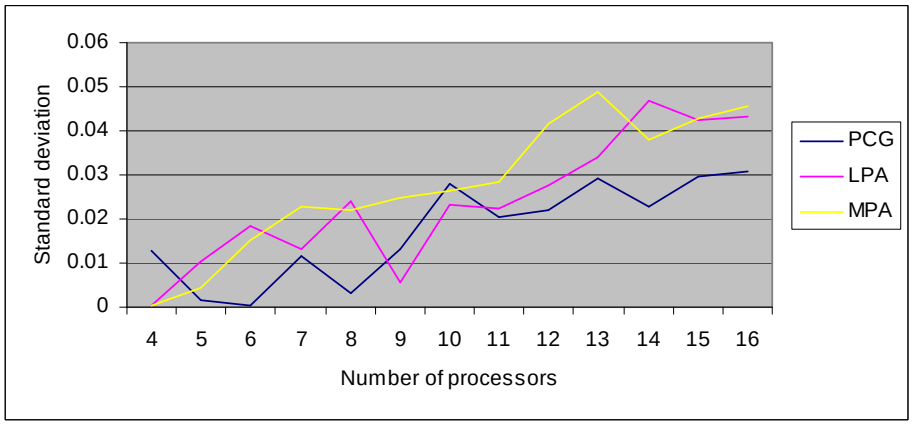


Figure 8.32: Standard deviation values for the execution times of LPA, PCG and MPA, varying with the number of processors, when the communication topology is a complete bipartite graph.

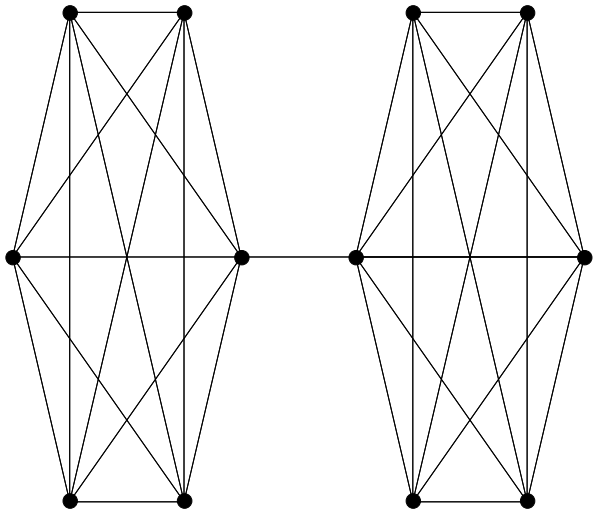


Figure 8.33: Bridge

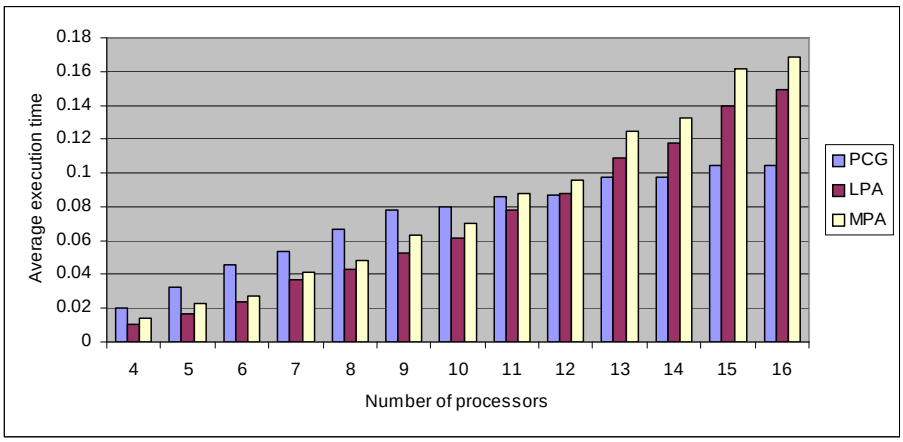


Figure 8.34: Average execution times of LPA, PCG and MPA, varying with the number of processors, when the communication topology is a bridge graph.

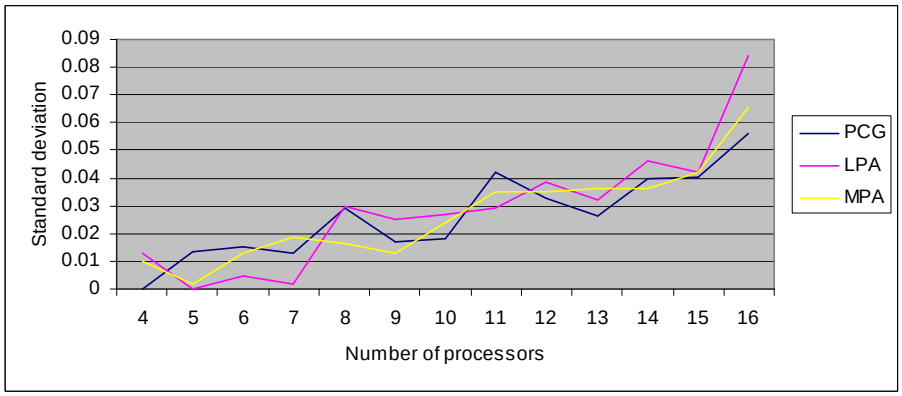


Figure 8.35: Standard deviation values for the execution times of LPA, PCG and MPA, varying with the number of processors, when the communication topology is a bridge graph.

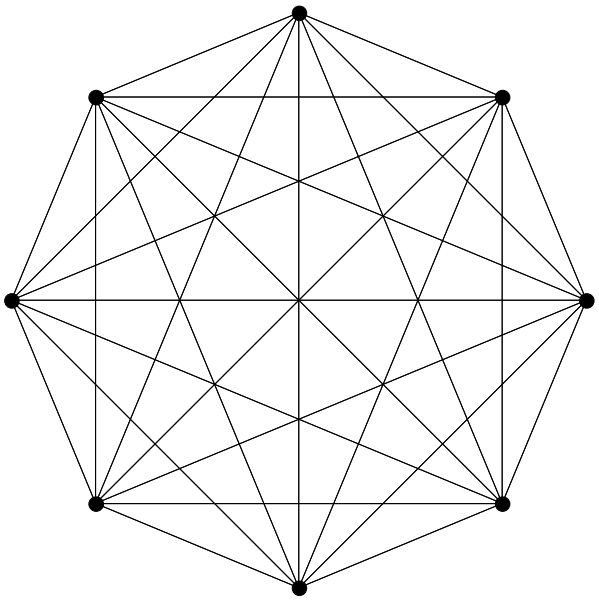


Figure 8.36: Complete graph

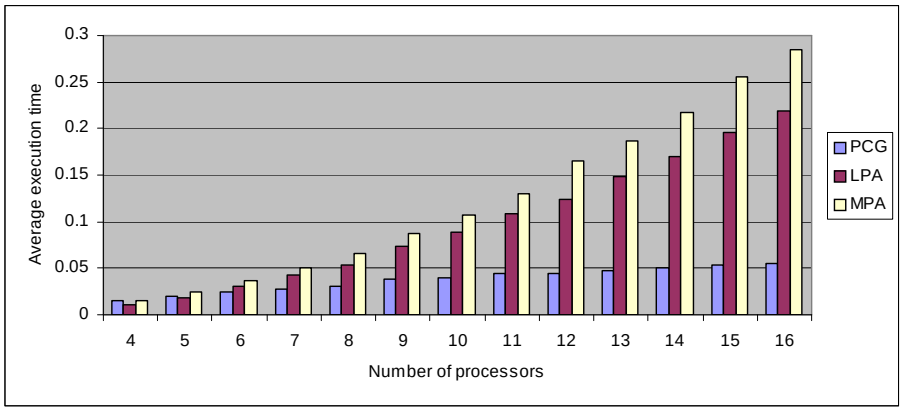


Figure 8.37: Average execution times of LPA, PCG and MPA, varying with the number of processors, when the communication topology is a complete graph.

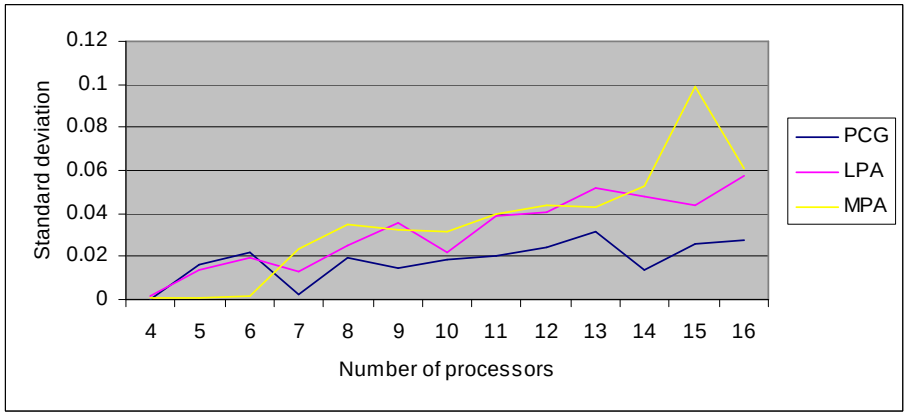


Figure 8.38: Standard deviation values for the execution times of LPA, PCG and MPA, varying with the number of processors, when the communication topology is a complete graph.

The figures show that in the case of a path or a ring the algorithm MPA is the best while the algorithm PCG is the worst. This is also true in the case of a binary tree. In the case of a quaternary tree, LPA is the best and PCG still is the worst. In the case of graphs that are dense or have a small diameter like complete graphs, wheel graphs, complete bipartite graphs or star graphs PCG performs best and the algorithm MPA the worst. This leads to the conclusions that no algorithm perform best in all the situations and that the communication topology has a strong impact on the execution times of these algorithms. In the sequel we shall proceed to a finer analysis of the execution times of the three algorithms.

Examining the algorithms PCG, LPA and MPA, one can remark that the only global operation that is performed inside the loop is global summation. This can be typically performed via a MPI reduction operation and usually can be done with $O(\log(p))$ parallel point to point communication phases [80]. Let T^{ar} denote the time needed for performing a global summation. The algorithm LPA performs further a global operation of type allgather. Let T^{ag} denote the time needed for performing it (usually has the same complexity as a global reduction operation).

- **Estimation of the execution time of PCG:**

  $T_{\text{PCG}} = \max T_i$, where $T_i = T_i^{\text{comp}} + T_i^{\text{lcomm}} + T_i^{\text{gcomm}}$, where T_{PCG} is the execution time of PCG T_i^{comp} is the time spent by the processor i to perform computations, T_i^{lcomm} and T_i^{gcomm} are the times spent by the processor i to perform neighbor-to-neighbor communication and global communication.

  Let N be the number of steps until convergence (N is at most the number of distinct eigenvalues of the communication graph). Then

  - $T_i^{\text{comp}} = h_i\, N\, (\delta_i\, \text{a}_{\text{PCG}} + \text{b}_{\text{PCG}}) + \text{f}_{\text{PCG}}$, where a_{PCG}, b_{PCG} and f_{PCG} are constants and h_i depends on the available capacity of the processor i.
  - $T_i^{\text{lcomm}} = (N+1) \sum_{ji \in E} (\text{d}_{\text{PCG}} \beta_{ij} + \alpha_{ij})$, where $\text{d}_{\text{PCG}} = \text{sizeof(double)}$.
  - $T_i^{\text{gcomm}} = (N+2)\, 2\, T^{ar}$.

  We have therefore,

$$T_i = h_i\, N\, (\delta_i\, \text{a}_{\text{PCG}} + \text{b}_{\text{PCG}}) + \text{f}_{\text{PCG}} + (N+1) \sum_{ji \in E} (\text{d}_{\text{PCG}}\, \beta_{ij} + \alpha_{ij}) + 2\, (N+2)\, T^{ar}$$

- **Estimation of the execution time of LPA:**

  $T_{\text{LPA}} = \max T_i$, where $T_i = T_i^{\text{comp}} + T_i^{\text{lcomm}} + T_i^{\text{gcomm}}$, where T_{LPA} is the execution time of LPA and T_i^{comp},T_i^{lcomm},T_i^{gcomm} have the same meaning as above.

  - $T_i^{\text{comp}} = h_i\,(p-1)\,(\delta_i\,p\,\text{a}_{\text{LPA}} + \text{b}_{\text{LPA}}) + \text{f}_{\text{LPA}}$, where a_{LPA}, b_{LPA} and f_{LPA} are constants and h_i depends on the available capacity of the processor i.
  - $T_i^{\text{lcomm}} = (p+1)\,\sum_{ji \in E}(p\,\text{d}_{\text{LPA}}\,\beta_{ij} + \alpha_{ij})$, where $\text{d}_{\text{LPA}} = \text{sizeof(double)}$.
  - $T_i^{\text{gcomm}} = p\,T^{ar} + T^{ag}$.

  We have therefore,

$$T_i = h_i\,(p-1)\,(\delta_i\,p\,\text{a}_{\text{LPA}} + \text{b}_{\text{LPA}}) + \text{f}_{\text{LPA}} + (p+1)\,\sum_{ji \in E}(p\,\text{d}_{\text{LPA}}\,\beta_{ij} + \alpha_{ij}) + p\,T^{ar} + T^{gs}.$$

- **Estimation of the execution time of MPA:** $T_{\text{MPA}} = \max T_i$, where $T_i = T_i^{\text{comp}} + T_i^{\text{lcomm}} + T_i^{\text{gcomm}}$, where T_{MPA} is the execution time of MPA and T_i^{comp},T_i^{lcomm},T_i^{gcomm} have the same meaning as above.

  - $T_i^{\text{comp}} = h_i\,(2p+1)\,(\text{a}_{\text{MPA}}\,p + \text{b}_{\text{MPA}}) + \text{f}_{\text{MPA}}$, where a_{MPA}, b_{MPA} and f_{MPA} are constants.
  - $T_i^{\text{lcomm}} = (2p+1)\,\sum_{ji \in E}(\text{d}_{\text{MPA}}\,\beta_{ij} + \alpha_{ij})$, where $\text{d}_{\text{MPA}} = \text{sizeof}\,(\textit{double})$.
  - $T_i^{\text{gcomm}} = T^{gs}$.

  We have therefore,

$$T_i = h_i\,(2p+1)\,(\text{a}_{\text{MPA}}\,p + \text{b}_{\text{MPA}}) + \text{f}_{\text{MPA}} + (2p+1)\,\sum_{ji \in E}(\text{d}_{\text{MPA}}\,\beta_{ij} + \alpha_{ij}) + T^{gs}.$$

From the above formulas one can see that the execution times of the considered algorithms depend on the following parameters: number of iterations performed, the actual computational capacities of the processors, the number of neighbors of each processor, the point-to-point communication bandwidths and the startup costs for sending messages.

Normally, as PCG performs more global operations than the other two algorithms, one can expect that this is slower than the others in all the cases. However, as the number of iterations performed depends tightly on the communication graph (it converges after a number of steps that is at most equal to the number of distinct eigenvalues of the generalized Laplacian) it is possible that for highly connected graphs it converges faster. This is indeed the case of a complete graph as it has a small number of distinct eigenvalues (the unweighted Laplacian has only two distinct eigenvalues!). For graphs with small maximum degrees the local communication is very fast, as any processor has a small number of neighbors but the use of collective operations may be costly if these graphs have a large diameter.

In the case of a ring or a path, the number of distinct eigenvalues of the communication graph equals the number of processors. As the degree of each vertex is at most 2, the global communication is the most penalty. It is therefore natural that the algorithm PCG takes more time than the algorithms LPA and MPA. On the other hand, LPA performs in each iteration step a global operation, while MPA needs only local communication at each iteration step. It is therefore natural that the algorithm LPA take more time than the algorithm MPA. On a communication graph of type quaternary tree, LPA performs asymptotically better than the others and PCG the worst. However, one can remark that in the case of communication graphs with large degrees and high connectivity, such as a star or a clique, the algorithm PCG is the best. This can be explained by that these kind of graphs may have few distinct eigenvalues. On the other hand, for graphs that are dense, MPA is the slowest because it performs $O(2p)$ iterations while the others finish in $O(p)$ steps and at each iteration step the communication is costly because the number of neighbors is relatively large.

By extrapolation, one may suggest that

- On graphs with a small maximum degree, poor connectivity and large diameter the MPA algorithm should be used.
- On graphs with medium degree, medium connectivity and diameter, the LPA algorithm should be chosen.
- On graphs with high maximum degrees, large connectivity and small diameter the PCG algorithm should be used.

As a conclusion, no algorithm performs best in all the cases and in practice it is convenient to dispose of a large palette of methods, which should be chosen/applied depending on the context. In the next section we propose an adaptive algorithm that is able to take into account the change of the parameters of the environment and is capable to take advantage of a previously computed flow.

Chapter 9

An incremental system sensitive dynamic load balancing algorithm

In a distributed environment, the main events that can affect the execution time of a parallel application of the type we assumed are:

1. The modification of the communication patterns induced by the actual data distribution (as a result of data migration between processors).
2. The modification of the available computational capacities of the processors (as an effect of external factors like allowing multiple users entering or leaving the system).
3. The modification of the communication costs (as a consequence of the modification of the network parameters like the available bandwidth).
4. The modification of the complexity of the work assigned to processors (as a result of a refinement/de-refinement operation asked for by the application).

9.1 Balancing operator

Definition 9.1.1. Let (G, l, c, w) be the current state of a heterogeneous environment w.r.t. to a given application and A the vertex-edge incidence matrix of the communication graph G induced by

the current data distribution w.r.t a given application. The $q \times p$ matrix Θ is a balancing operator if

$$A\,\Theta = I - ce^T.$$

Theorem 24. *Let (G, l, c, w) the current state of a heterogeneous environment and*

$$\Theta = WA^T D^{-1/2}(\mathcal{L} + c^{1/2}c^{T\,1/2})^{-1}D^{-1/2}.$$

Then

1. *Θ is a balancing operator.*
2. *$\Theta\, c = 0$.*
3. *There exists a $q \times q$ projection matrix U so that $\Theta = W^{1/2}UW^{-1/2}\Theta'$, for any balancing operator Θ'.*

Proof. 1. Clearly, Θ is a balancing operator:

$$\begin{aligned} A\Theta &= D^{1/2}\mathcal{L}(\mathcal{L} + c^{1/2}c^{T\,1/2})^{-1}D^{-1/2} \\ &= I - D^{1/2}c^{1/2}c^{T\,1/2}(\mathcal{L} + c^{1/2}c^{T\,1/2})^{-1}D^{-1/2} \\ &= I - D^{1/2}c^{1/2}c^{T\,1/2}D^{-1/2} = I - ce^T. \end{aligned}$$

2.

$$\begin{aligned} \Theta c &= WA^T D^{-1/2}(\mathcal{L} + c^{1/2}c^{T\,1/2})^{-1}D^{-1/2}c = WA^T D^{-1/2}(\mathcal{L} + c^{1/2}c^{T\,1/2})^{-1}c^{1/2} \\ &= WA^T D^{-1/2}c^{1/2} = WA^T e = 0 \end{aligned}$$

3. Let Θ' an arbitrary balancing operator for (G, l, c, w). Then $I = A\Theta' + ce^T$. Let $U = W^{1/2}A^T D^{-1/2}(\mathcal{L}+c^{1/2}c^{T\,1/2})^{-1}D^{-1/2}AW^{1/2}$. As $c^{T\,1/2}$ and $c^{1/2}$ are left and right eigenvectors

of $\mathcal{L}$ corresponding to the eigenvalue 0, ${c^T}^{1/2} c^{1/2} = 1$ and $A^T e = 0$, one has

$$\begin{aligned}\Theta &= WA^T D^{-1/2}(\mathcal{L} + c^{1/2}{c^T}^{1/2})^{-1} D^{-1/2}(A\Theta' + ce^T) \\ &= WA^T D^{-1/2}(\mathcal{L} + c^{1/2}{c^T}^{1/2})^{-1} D^{-1/2} A\Theta' + WA^T D^{-1/2} c^{1/2} e^T \\ &= W^{1/2} U W^{-1/2} \Theta'.\end{aligned}$$

□

With these settings, a balancing flow can be seen as the result of the application of a balancing operator to a particular workload vector.

Definition 9.1.2. A balancing flow in a heterogeneous environment $H = (G, l, c, w)$ is a vector $\phi \in R^m$ such that $\phi = \Theta\, l$.

9.2 Incremental computation of the minimal balancing flow when the capacities change

Let us assume that the available processor capacities have changed as an effect of tasks competition for the processor resources in the system. The question is how to compute the new minimal balancing flow ϕ' within the new heterogeneous context (G, l, c', w) taking advantage of the already computed balancing flow ϕ in the previous state $H = (G, l, c, w)$. One can notice that ϕ can be expressed as

$$\phi = \Theta\, l = WA^T(L + cc^T)^{-1} l,$$

and

$$\phi' = \Theta' l = WA^T(L + c'{c'}^T)^{-1} l.$$

As e^T is a left eigenvector of L corresponding to th eigenvalue 0 and c, c' are normalized w.r.t to the norm $\|\cdot\|_1$, it can be easily proved that

$$(I - c'e^T)(L + c'{c'}^T) = L \tag{9.2.1}$$

and

$$(L + cc^T)(I - ec^T) = L \tag{9.2.2}$$

Therefore,

$$(L + cc^T)^{-1}(I - c'e^T) = (I - ec^T)(L + c'c'^T)^{-1} \tag{9.2.3}$$

It follows that

$$WA^T(L + cc^T)^{-1}(I - c'e^T) =$$
$$WA^T(I - ec^T)(L + c'c'^T)^{-1} =$$
$$WA^T(L + c'c'^T)^{-1},$$

which means that

$$\Theta' = \Theta(I - c'e^T) = \Theta(I - (c' - c)e^T),$$

where Θ and Θ' are the balancing operators defined in the previous section for (G, l, c, w) and (G, l, c', w), respectively. We deduce that

$$\phi' = \Theta' l = \Theta\,(I - (c' - c)e^T)l = \Theta\,(l - \overline{l'}) = \phi - \Theta\,(c' - c)(e^T l) = \phi - (\sum_i l_i)\,\Theta\,(c' - c).$$

The above expression shows how to compute a new balancing operator from a previous one when a transition of the system from a state (G, l, c, w) to a state (G, l, c', w) occurs, i.e.

$$\begin{aligned} \Theta' &= \Theta - \Theta\,(c' - c)\,e^T \\ \phi' &= \phi - \alpha\,\Theta\,(c' - c) \end{aligned}$$

where $\alpha = \sum_{i=1}^{p} l_i$. Therefore, it suffices that a processor acquire information regarding the differences in capacities, from the processors reporting such modifications, in order to be able to rapidly compute a new balancing flow.

9.3 Incremental computation of the minimal balancing flow when the loads change

The minimal balancing flow in $H = (G, l, c, w)$ is $\phi = \Theta\, l$. When the processor loads change, with the new workload vector l', the minimal balancing flow corresponding to the new situation (G, l', c, w) is $\phi' = \Theta'\, l'$. But the expression of the balancing operator does not depend on the workload vector, therefore, $\Theta' = \Theta$ and

$$\phi' = \Theta\, l + \Theta\,(l' - l) = \phi + \Theta\,(l' - l).$$

When the processor capacities, the communication topology and the communication costs are invariant but only the processor loads change dynamically, this is a very fast way to balance the system. In a preprocessing phase one has just to compute the balancing operator. After that, every time a load change is reported, it suffices to broadcast the load differences and then to apply the operator to the vector of load differences.

9.4 Flow computation when the communication topology or the communication costs change

As long as only the load or the capacities change, the computation of a new balancing flow can be done rapidly by reusing the already computed balancing operator. When the communication topology or the communication costs change, the computation of the balancing operator must be restarted from scratch. Various iterative methods can be used for approximating the inverse of $(\mathcal{L} + c^{1/2}c^{T\,1/2})^{-1}$, which is the part that is hardest to compute that appears in the expression of Θ. As it is a dense matrix, its computation could involve a large amount of computation and global communication. For its computation we use the following characterization [68]:

Theorem 25.

$$\Theta = -WA^T D^{-1/2} \frac{\mathcal{L}^{p-2} + a_1\mathcal{L}^{p-3} + \ldots + a_{p-2}I}{a_{p-1}} D^{-1/2},$$

where $a_1, \ldots, a_p$ are the coefficients of the Laplacian polynomial.

Proof. The key observation is that $\Lambda(\mathcal{L}) = \{1\} \bigcup \Lambda(\mathcal{L} + c^{1/2}c^{T\,1/2}) \setminus \{0\}$, where $\Lambda(A)$ denotes for a given matrix A the set of its eigenvalues. As $c^{1/2}$ is an eigenvector corresponding to the unique eigenvalue 0, one has

$$(\mathcal{L} + c^{1/2}c^{T\,1/2})(\mathcal{L}^{p-1} + a_1\mathcal{L}^{p-2} + \ldots + a_{p-1}I) \;=\; a_{p-1}c^{1/2}c^{T\,1/2}. \tag{9.4.1}$$

On the other hand, as the inverse of $\mathcal{L} + c^{1/2}c^{T\,1/2}$ exists and has the properties mentioned in the section 4.5, one has

$$\mathcal{L}^{p-1} + a_1\mathcal{L}^{p-2} + \ldots + a_{p-1}I \;=\; a_{p-1}(\mathcal{L} + c^{1/2}c^{T\,1/2})^{-1}c^{1/2}c^{T\,1/2} \;=\; a_{p-1}c^{1/2}c^{T\,1/2}. \tag{9.4.2}$$

Multiplying on the left by $D^{1/2}$ and on the right by $D^{-1/2}$, one gets

$$ce^T = I + AWA^T D^{-1/2} \frac{\mathcal{L}^{p-2} + a_1\mathcal{L}^{p-3} + \ldots + a_{p-2}I}{a_{p-1}} D^{-1/2}. \quad (9.4.3)$$

Let us denote by Θ' the expression $-(1/a_{p-1})\, WA^T D^{-1/2} (\mathcal{L}^{p-2} + a_1\mathcal{L}^{p-3} + \ldots + a_{p-2}I)\ D^{-1/2}$. From the above relation, we deduce that $A\Theta' = I - ce^T$ i.e. Θ' is a balancing operator for the system when the current state is (G, l, c, w). We prove now that the above operator coincides with the balancing operator Θ considered in the theorem 24. Indeed, from theorem 24 and equation (9.4.2) we have

$$\begin{aligned}
\Theta - \Theta' &= WA^T D^{-1/2} \left[(\mathcal{L} + c^{1/2}c^{T\,1/2})^{-1} + \frac{\mathcal{L}^{p-2} + a_1\mathcal{L}^{p-3} + \ldots + a_{p-2}I}{a_{p-1}} \right] D^{-1/2} \\
&= WA^T D^{-1/2} (\mathcal{L} + c^{1/2}c^{T\,1/2})^{-1} \left[I + (\mathcal{L} + c^{1/2}c^T) \frac{\mathcal{L}^{p-2} + \ldots + a_{p-1}}{a_{p-1}} \right] D^{-1/2} \\
&= WA^T D^{-1/2} c^{1/2} \left[I + c^{1/2}c^{T\,1/2} - I + \frac{a_{p-2}}{a_{p-1}} c^{1/2}c^{T\,1/2} \right] \\
&= \frac{a_{p-2}}{a_{p-1}} WA^T D^{-1/2} c^{1/2} c^{T\,1/2} D^{-1/2} = \frac{a_{p-2}}{a_{p-1}} WA^T ee^T = 0.
\end{aligned}$$

□

Obviously, using this expression for computing the balancing operator requires that the coefficients of the characteristic polynomial have been already computed. Actually, one can do better and can compute in exactly $p-1$ iterations a new balancing operator by adapting the Fadeev-LeVerrier algorithm for computing the coefficients of the Laplacian polynomial [60]. Thus, the following algorithm computes a new balancing operator:

$U = I$

for $(k = 1; k <= p-2; k++)$ **do**

$U = L\, D^{-1}\, U - 1/k\ Tr(L\, D^{-1}\, U)$

end for

$\alpha = -1/(p-1)\ Tr(L\, D^{-1}\, U)$

$\Theta = -1/\alpha\ W\ A^T\ D^{-1}\ U$

9.5 General algorithm

As seen above, depending on the kind of change that occur in the system, one can synthesize the results obtained above and derive a general parallel incremental algorithm as follows:

Algorithm 16 Adaptive algorithm for multi-user heterogeneous environments

if *(the load changed)* **then**

let l^{new} be the new workload vector

compute the new balancing flow vector ϕ^{new}:

$$\phi^{new} = \phi^{old} + \Theta\,(l^{new} - l^{old})$$

end if

if *(if the processing capacities changed)* **then**

let c^{new} be the new capacity vector

compute the new balancing operator and the new balancing flow:

$$\Theta^{new} = \Theta^{old} - \Theta^{old}\,(c^{new} - c^{old})\,e^T$$

$$\phi^{new} = \phi^{old} - (\textstyle\sum_{i=1}^{p} l_i)\ \Theta^{old}\ (c^{new} - c^{old})$$

end if

if *(the communication patterns changed)* **then**

compute the new incidence matrix A^{new}

compute the new generalized Laplacian matrix:

$$\mathcal{L}^{new} = D^{-1/2} A^{new} W (A^{new})^T D^{-1/2}$$

Compute Θ^{new} using the algorithm (9.4)

$\phi^{new} = \Theta^{new}\, l$

end if

if *(the network parameters changed)* **then**

compute the network communication costs w^{new}

compute the new generalized Laplacian matrix:

$$\mathcal{L}^{new} = D^{-1/2} A W^{new} A^T D^{-1/2}$$

Compute Θ^{new} using the algorithm (9.4)

$\phi^{new} = \Theta^{new}\, l$

end if

9.5.1 Parallel implementation aspects

Clearly, each processor can easily deal with the aspects concerning its capacity, load or the communication with its neighbors. The problem is how to handle the balancing operator. Ideally, the processors should know only those parts of Θ that play a role in the computation of the flow along the edges with its neighbors. The balancing operator can be put in the form $\Theta = WA^TD^{-1}U$, U being a $p \times p$ matrix of reals. Each processor needs only to store the i-th column of U, denoted u_i, its neighbors and the cost of communication with these neighbors.

There are four categories of messages that can initiate a dynamic load procedure:

- the load changed: `MSG_LOADCHGD`
- the processing capacities changed: `MSG_CAPCHGD`
- the topology changed: `MSG_TOPOCHGD`
- the network parameters changed: `MSG_NETCHGD`

A SPMD model is used in the implementation phase. Each processor uses the following local data structures:

1. *Load*, the workload of the processor itself
2. *Cap*, the capacity of the processor itself
3. *arr*, an array of size number of processors
4. *nbNbrs*, the number of neighbors
5. $arrNb[0..NbNbrs-1][0..p-1]$ array of potentials of neighbors
6. $arrLoad[0..p-1]$ - array of the processors' workloads
7. $eWght[0..nbNbrs-1]$, array of size *nbNbrs* representing the communication costs with the neighbors.
8. $Flow[0..nbNbrs-1$, array of size *nbNbrs* with the components representing amounts of data that should be sent/received to/from neighbors

9. $TotLoad$, the sum of the loads of the processors
10. $nLChgd$, the number of processors whose workload changed
11. $nCChgd$, the number of processors whose workload changed

Algorithm 17 AdA

if ($Load^{new} \neq Load^{old}$) **then**

post a message of type `MSG_LOADCHGD`

else if ($Cap^{new} \neq Cap^{old}$) **then**

post a message of type `MSG_CAPCHGD`

else if (Neighborhood changed) **then**

post a message of type `MSG_TOPOCHGD`

else if (Communication costs changed) **then**

post a message of type `MSG_NETCHGD`

end if

Extract a message from the queue

if (MsgType == `MSG_LOADCHGD`) **then**

$Flow = CompFlowLoadChgd()$

end if

if (MsgType == `MSG_CAPCHGD`) **then**

$Flow = CompFlowCapChgd()$

end if

if (MsgType == `MSG_TOPOCHGD`) **then**

$Flow = CompFlowTopoChgd()$

end if

if (MsgType == `MSG_NETCHGD`) **then**

$Flow = CompFlowNetChgd()$

end if

return $Flow$

Function 18 CompFlowLoadChgd

broadcast $\delta = Load^{new} - Load^{old}$

receive $\delta_0, \ldots, \delta_{nLChgd-1}$ from the processors whose load changed, $p_0, \ldots, p_{nLChgd-1}$

send arr to all neighbors

receive from neighbors the copies of arr in $arrNb_0, \ldots, arrNb_{nbNbrs-1}$

for all neighbors j **do**

$Flow_j^{new} = Flow_j^{old}$

for $(k = 0; k < nLChgd; k++)$ **do**

$Flow_j^{new} = Flow_j^{new} - eWght_j \cdot (arr[p_k] - arrNb_j[p_k]) \cdot \delta_k$

end for

end for

return $Flow^{new}$

Function 19 CompFlowCapChgd

broadcast $\delta = Cap^{new} - Cap^{old}$

receive $\delta_0, \ldots, \delta_{nCChgd-1}$ from the processors whose capacity changed, $p_0, \ldots, p_{nCChgd-1}$

normalize the capacities

$\alpha = 0$

for $(k = 0; k < nCChgd; k++)$ **do**

$\alpha = \alpha + arr[p_k]\,\delta_k$

end for

for $(i = 0; i < p; i++)$ **do**

$arr[i] = arr[i] - \alpha$

end for

send α to neighbors

receive the α values from neighbors in $\alpha_0, \ldots, \alpha_{nbNbrs-1}$

allreduce $(load, TotLoad, SUM)$

for all neighbors j **do**

$Flow_j^{new} = Flow_j^{old} - TotLoad \cdot eWght_j \cdot (\alpha - \alpha_j)$

end for

return $Flow^{new}$

Function 20 CompFlowTopoChgd

$arr^{new} = CompNewBalOpArr()$

for $(i = 0; i < p; i++)$ **do**

$arr^{new}[i] = arr^{new}[i]/Cap$

end for

send arr^{new} to neighbors

receive from neighbors the local copies of arr^{new}, in $arrNb_0, \ldots, arrNb_{nbNbrs-1}$

allgather $(arrLoad, Load)$

for all *neighbors* j **do**

for $(i = 0; i < p; i++)$ **do**

$Flow_j^{new} = Flow_j^{new} + eWght_j \cdot (arr^{new}[i] - arrNb_j[i]) \cdot arrLoad[i]$

end for

end for

return $Flow^{new}$

Function 21 CompFlowNetChgd

for $(i = 0; i < p; i++)$ **do**

get the new values for $eWght_i, i = 0, \ldots, nbNbrs - 1$

end for

$arr^{new} = CompNewBalOpArr()$

for $(i = 0; i < p; i++)$ **do**

$arr^{new}[i] = arr^{new}[i]/Cap$

end for

send arr^{new} to neighbors

receive from neighbors the local copies of arr^{new}, in $arrNb_0, \ldots, arrNb_{nbNbrs-1}$

allgather $(arrLoad, Load)$

for all $neighbors\ j$ **do**

for $(i = 0; i < p; i++)$ **do**

$Flow_j^{new} = Flow_j^{new} + eWght_j \cdot (arr^{new}[i] - arrNb_j[i]) \cdot arrLoad[i]$

end for

end for

return $Flow^{new}$

Function 22 CompNewBalOpArr

for $(i = 0; i < p; i++)$ **do**

get the new values for $eWght_i, i = 0, \ldots, nbNbrs - 1$

end for

for $(i = 0; i < p; i++)$ **do**

$arr[i] = 0$

end for

$arr[rank] = 1$

for $(k = 0; k < p; k++)$ **do**

if $(k == p - 2)$ **then**

for $(i = 0; i < p; i++)$ **do**

$arrS[i] = arr[i]$

end for

end if

send arr to neighbors

receive from neighbors the local copies of arr, in $arrNb_0, \ldots, arrNb_{nbNbrs-1}$

for $(i = 0; i < p; i++)$ **do**

$\alpha = arr[i]/Cap$

for all neighbors j **do**

$arr[i] = arr[i] + eWght_j \cdot (\alpha - arrNb_j[i]/CapNb_j)$

end for

end for

allreduce $(arr[myrank], T, SUM)$

$arr[myrank] = arr[myrank] - (T/(k+1))$

end for

for $(i = 0; i < p; i++)$ **do**

$arr[i] = -(k+1)/T \cdot arrS[i]$

end for

return $arrS$

9.6 Experiments

We performed experiments with the AdA algorithm and compared it with the algorithms PCG, LPA, MPA. We used the following communication topologies: path, ring, binary tree, star and clique. We generated a number of messages of the following types:

- the load changed, coded by 0
- the capacities changed, coded by 1
- the communication topology changed, coded by 2

A number of five different series of messages (of the types described above) were generated. Each series contained 20 messages. A number of 5 arrays of loads, 5 arrays of capacities and 5 communication topologies were used. The test program examined at each step the queue of messages, extracted the current message and performed load balancing with each of the four algorithms. If the extracted message was of the type "load changed", the next vector of loads was considered in the suite of load vectors provided at start. The same idea was used for the vectors of capacities and for the topologies. The following test message arrays were used:

```
int  arrMsg[5][NMSG]={{2,0,1,0,2,1,2,0,1,0,1,0,0,1,0,1,1,0,1,0},
                      {2,0,1,2,0,1,2,0,1,0,2,0,1,2,0,1,2,0,1,0},
                      {2,0,1,2,2,1,2,0,1,0,2,0,1,2,0,1,2,0,2,0},
                      {2,0,1,0,2,1,2,2,1,2,1,2,2,0,2,1,2,2,2,0},
                      {2,0,2,2,2,1,2,0,1,2,2,0,2,2,2,1,2,2,2,0}};
```

Each algorithm AdA, LPA, PCG, MPA was executed each time a message was generated (`NMSG` times). Figures 9.1 to 9.5 show the execution times of these algorithms on the considered topologies.

Figures 9.1 to 9.5 show that the algorithm AdA performs better in all the tested cases. This is due to the fact that this algorithm is able to save computation time by taking advantage, when possible, of a previous computation of the balancing flow. Therefore, an incremental algorithm of the type of AdA may be more suitable than other approaches in a dynamic distributed environment.

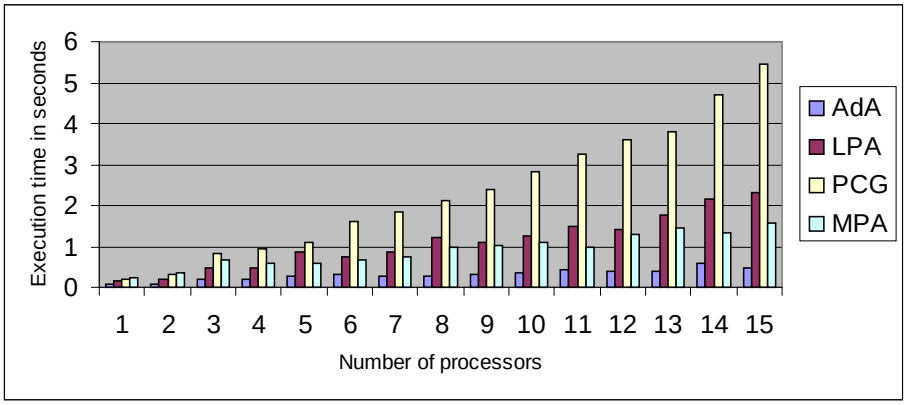


Figure 9.1: Execution times of the algorithms AdA, LPA, PCG, MPA corresponding to the first series of messages.

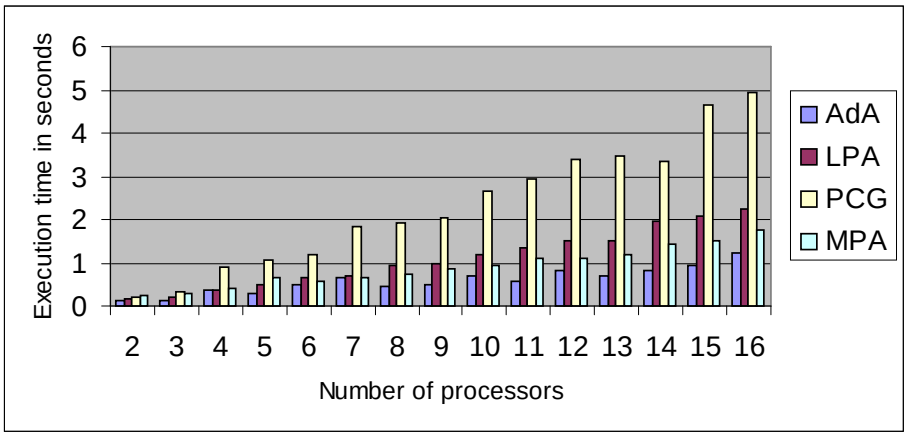


Figure 9.2: Execution times of the algorithms AdA LPA PCG, MPA corresponding to the second series of messages.

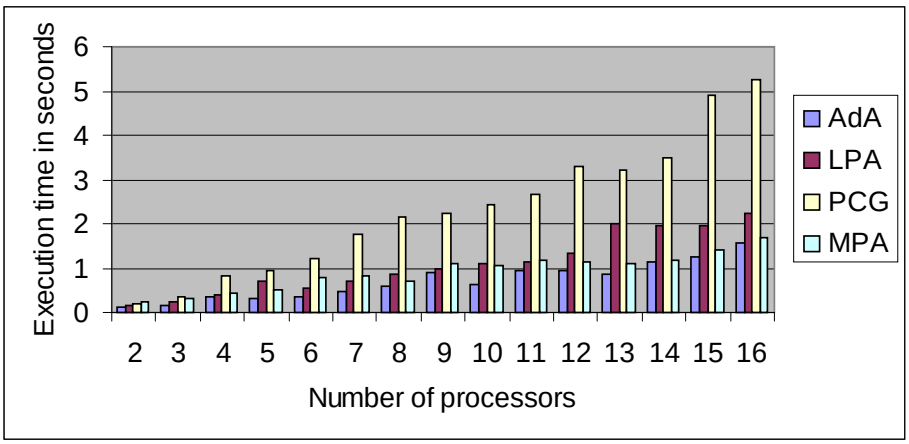


Figure 9.3: Execution times of the algorithms AdA, LPA, PCG, MPA corresponding to the third series of messages.

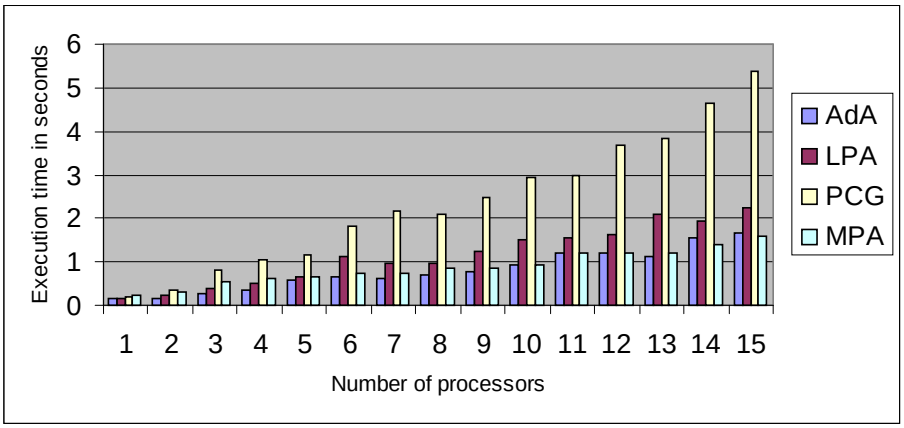


Figure 9.4: Execution times of the algorithms AdA, LPA, PCG, MPA corresponding to the fourth series of messages.

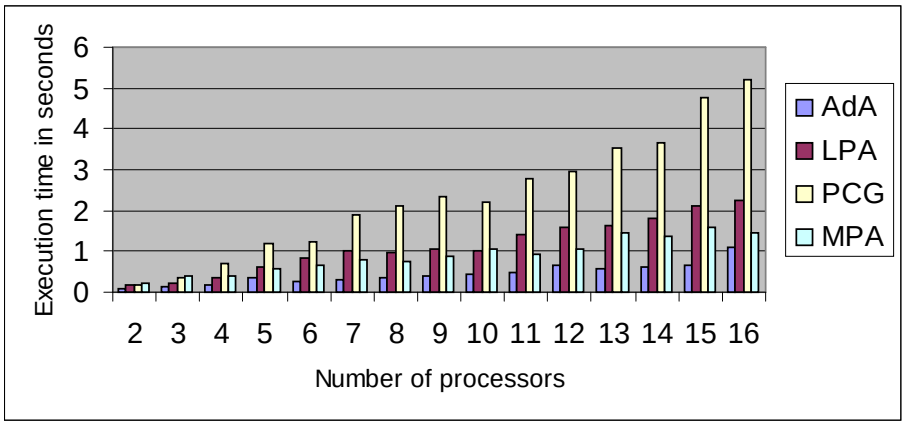


Figure 9.5: Execution times of the algorithms AdA, LPA, PCG, MPA corresponding to the fifth series of messages.

Chapter 10

Testbed

The number of scientific and industrial parallel applications has significantly grown in the last decade. However, many software is still characterized by a lack of generalization and standardization and in many cases by a lack of adequate documentation. This reduces the possibility of reusing already available codes and restricts the interdisciplinary interference between scientists. On the other hand, designing parallel solutions for scientific and computational engineering problems requires significant expertise and effort from the part of developers, as these tend to involve more and more advanced interdisciplinary knowledge. Developing specialized software modules, as for example a dynamic load balancing module, requires validation for a broad set of test-cases. In many cases the validation is done with respect to some specific applications and/or hardware architectures and the absence of clearly defined reference testbeds may constitute a limiting factor for the developers. On the other hand, developing a reusable simulation framework that meets the general characteristics of a real application may contribute to a substantial reduction of the development costs.

10.1 Simulation testbed

In order to be able to test our dynamic load balancing techniques, we simulated the behaviour of a generic parallel application that performs adaptive computations. We designed and implemented HeRMeS, a tool that allows to perform such simulations in distributed heterogeneous environments. HeRMeS incorporates the dynamic load balancing methods discussed in the previous chapters and

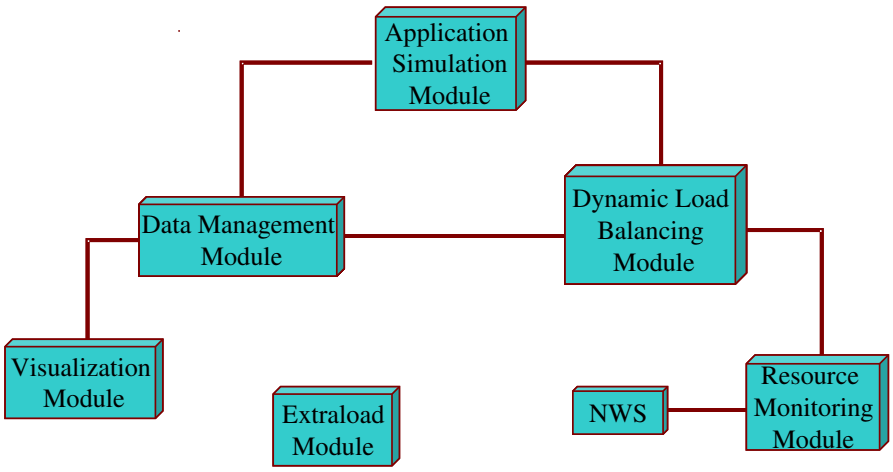


Figure 10.1: General scheme of HeRMeS.

is sensitive to the modification of the network or processor related parameters (such as the latency, bandwidth, processor capacity, memory). HeRMeS provides a basic framework for the development of a general class of applications that match the assumed scheme. It let the possibility to the user to add his own solver code, without being directly concerned with the dynamic load balancing implementation aspects.

The following software modules were designed and implemented: the *Application simulation module*, the *Data management module*, the *Dynamic load balancing module*, the *Resource monitoring module* and the *Visualization module*. Additionally, an independent module that simulates the activity of external users/processes was considered: the *Resources extra-load module*. The general scheme of the application is as shown in figure 10.1. We assumed a SPMD programming model and a conceptual scheme as illustrated in the figure 10.2. For the implementation we used the ANSI C programming language and LAM/MPI, as communication library. In the following we describe the functionality of each of the above mentioned modules.

- **The Data management module**

  This module is in charge with the management of the data handled by the application. It is assumed that between the objects encapsulating this data there exist computational dependencies. The underlying mesh is initially partitioned using the MeTiS [35] tool and afterwards is distributed onto the machines. It is assumed that a one-to-one mapping exists between

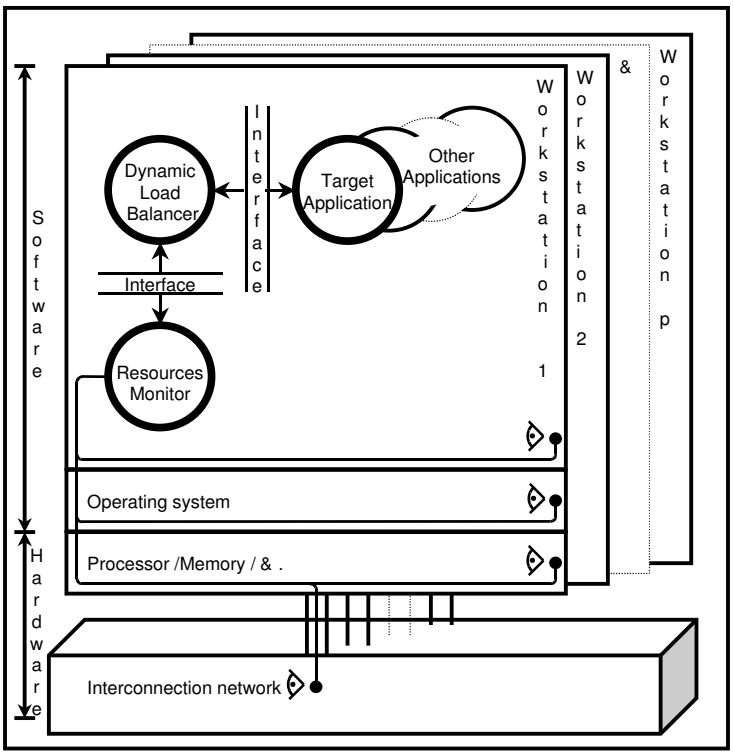


Figure 10.2: Conceptual scheme

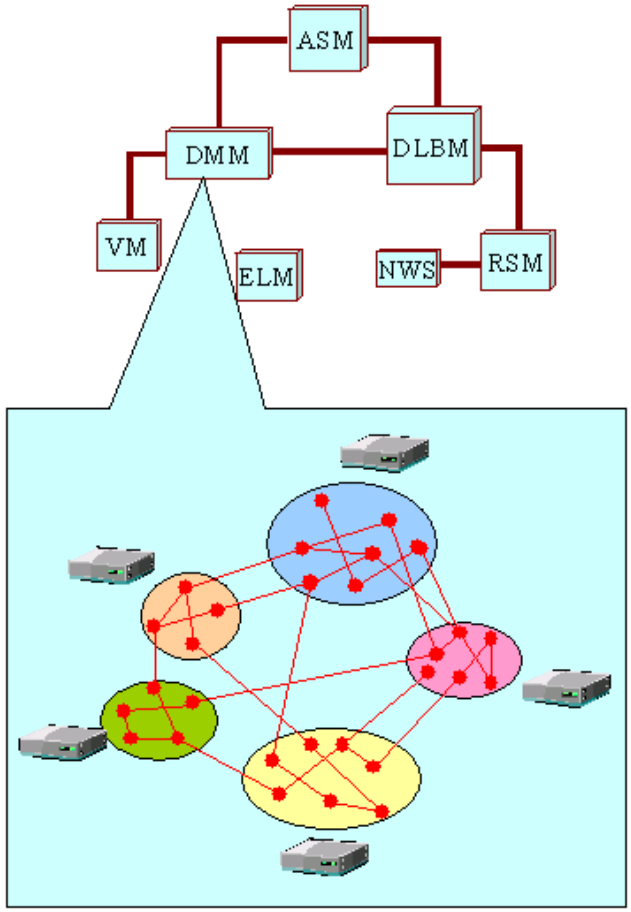


Figure 10.3: Data management module

partitions and machines.

This module contains several primitives that allow to manipulate data structures that encapsulate information regarding the mesh. Specifically, these primitives allow for

- reading input data and writing diverse information from/into files.
- distributing data to machines.
- filling the local structures of the processors with information related to the nodes of the assigned subdomains.
- building the subdomain graph corresponding to the current partitioning.

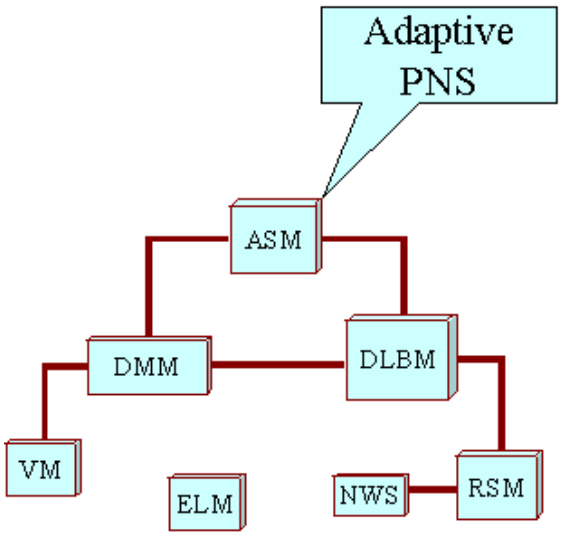


Figure 10.4: Application simulation module

- migrating data between the subdomains/processors (according to the informations received from the load balancing module).
- modifying the structure of the mesh, following the orders of the application simulation module (where to refine, which percentage).
- notifying the load balancing module when an adaption operation was completed.
- providing specific information regarding the subdomain graph to other modules, upon request.

- **The Application Simulation Module**

  The main task assumed by this module is to simulate the behaviour of an adaptive parallel numerical solver. In many cases such a solver consists in a computational kernel that solves a linear system of equations using iterative methods. The general assumed scheme is that indicated in the algorithm 2 and as illustrated in the figure 10.5. Here, one can identify a local computation part, a local communication part and a global communication part.

  The module repeats a number of times the following operations:

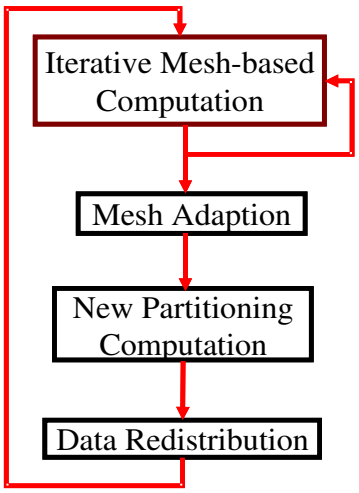


Figure 10.5: Generic scheme of a parallel numerical solver

- Each processor simulates the local computation part of a parallel numerical solver. Specifically, it performs a number of primitive operations according to a user supplied function that represents a measure of the complexity of the work assigned to it. This is usually a function of the volume of data residing on that processor and the processor's speed.
- Each processor exchanges a number of messages with its neighbors in the communication topology induced by the current data distribution. The size of the messages exchanged between two processors is proportional to the length of the shared boundary.
- A number of global operations are performed. In a real application these are usually necessary for computing a certain norm in order to be able to estimate the solution error.

Additionally, this module contains a number of primitives that allow it to fulfill the following tasks:

- It requests information about the network and the processor performance from the Resources Monitoring Module. Each processor maintains a structure that is dynamically updated and that contains the actual speed of the processor, its neighbors and the values of the network parameters.
- It requests from the Data Management Module the lengths of the boundaries shared by a processor with its neighbors in the communication topology induced by the application.

- It notifies the Data Management Module when a refinement operation must take place; it informs the Data Management Module on what regions (and in what percentage) this operation should be performed.

- **The Dynamic Load Balancing Module**

  This module implements a set of heterogeneous balancing methods that are able to take into account updated values of the workloads, processing capacities and of the network parameters when computing a new load distribution. All the methods discussed in the previous chapters were included. It contains a number of primitives that:

  - query the Resources Monitoring Module on system's parameters values (capacity, latency, bandwidth),
  - request information regarding the structure of the subdomain graph from the Data Management Module,
  - choose and apply methods for dynamic load balancing,
  - determine the amount of data to migrate,
  - select nodes for migration according to a gain function that takes into consideration the physical characteristics of the underlying communication network,
  - notify the data management module when a the load balancing process has been completed,
  - asks for the Data Management Module to operate modifications on the structure of the mesh by effectively migrating the selected data items.

  The process of the selection of nodes for migration is usually done in the homogeneous case so as the edge-cut is minimized. In this way, one hopes to minimize the communication time of the application. However, as Hendrickson et al. argued [29], this metric is inadequate. In the heterogeneous case one should provide a more adequate migration criteria for moving data between processors, capable to take into account the heterogeneities. In the homogeneous case the nodes are usually selected on the basis of a gain function. In the following we show how one may adapt this criteria for the use in heterogeneous environments [84].

Let $H = (G, l, c, w)$ be a heterogeneous model and $S_1, \ldots S_p$ the subdomains assigned to processors. The gain $g(v, q)$ of a vertex v in the subdomain S_p can be calculated for every other subdomain, S_q,$q \neq p$, and expresses how much the cost of a given partition would be improved were v to migrate to S_q. Given a vertex $v \in S_p$, let $e_q(v)$ denote the set of edges from v to vertices in S_q, $e_q(v) = \{\{v, z\} \in E : z \in S_q\}$. In the homogeneous case one has that $g(v, q) = |e_q(v)| - |e_p(v)|$. In the heterogeneous case,the communication costs shoould be also taken into account. The gain function to be used in this case is

$$g(v, q) = \sum_{i \in V(G)} |e_i(v)|(\widetilde{w}_{pi} - \widetilde{w}_{qi}),$$

where $\widetilde{w}$ is $p \times p$ matrix such that

$$\widetilde{w}_{ij} = \begin{cases} w_k, & \text{if } \{i, j\} = e_k \in E(G) \\ 0, & \text{otherwise.} \end{cases}$$

For a vertex $v \in S_p$ can be defined a preference function that gives the index of the sudomain/processor where to migrate v. This function can be defined as $f(v) = q$, where q is an index that maximizes the gain, i.e. for which $g(v, q) = \max_{t \neq p} g(v, t)$.

- **The Resources Monitoring Module**

  This module is responsible with the system resources status survey. It includes a benchmark code that was used for measuring the absolute computational capacities of the processors. We opted for a simple and easy to use benchmark that was developed in our department. This module has the responsibility to provide upon request to the Dynamic Load Balancing Module current values or forecasts of network or processor parameters such as available capacity, latency, bandwidth, etc. It exposes several interface functions that call NWS specific primitives. NWS is capable to furnish the fraction of CPU available for new processes, the fraction of CPU available to a process that is already running, network latency, network bandwidth, free memory, and the amount of space unused on a disk. NWS was configured manually for the target system before conducting experiments. Typically, a sensor and a forecaster are started on each machine, as is illustrated in figure 10.7. The application task residing on each machine first requests from the data management module the neighboring machines in the communication graph induced by the data distribution w.r.t the application and then sends

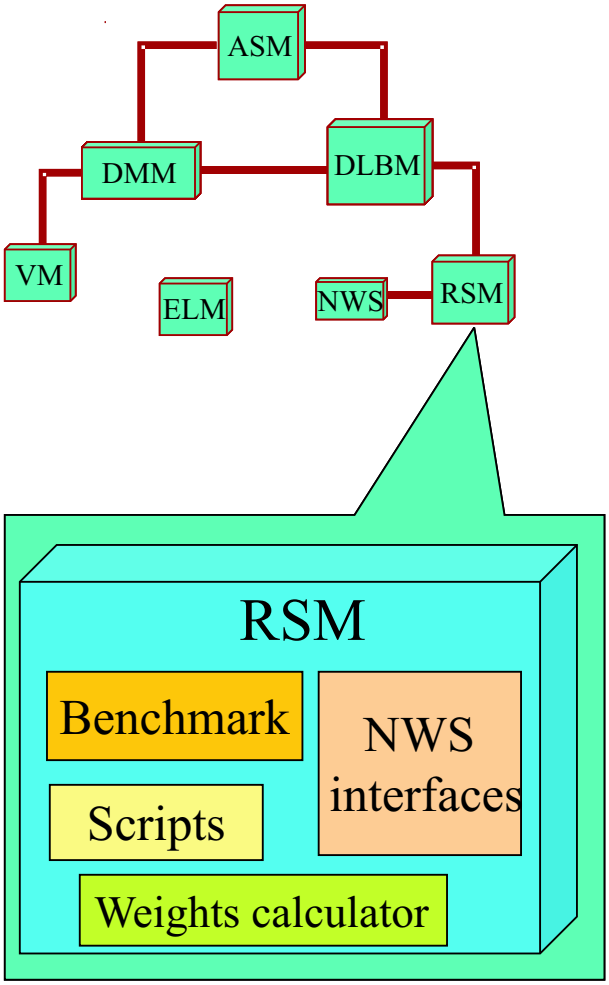


Figure 10.6: Resources monitoring module

queries to NWS to extract values for the system parameters along the corresponding links. These are used for setting up values for the synthetic parameters in the assumed theoretical heterogeneous model, which are then communicated to the Dynamic Load Balancing Module.

- **The Visualization module**

  This module contains simple primitives that allow to visualize a heterogeneous partitioning of a mesh. We first implemented it in Mathematica 4.0 and afterwards was ported to Java. It is able to visualize only 2-dimensional meshes for which the coordinates of the nodes are known.

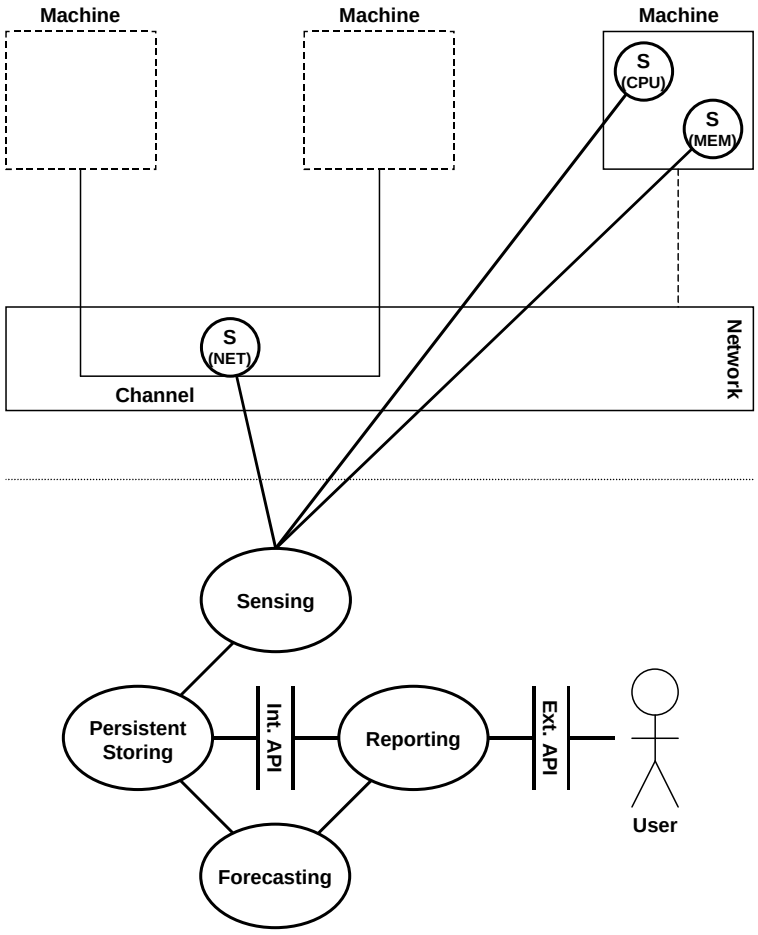


Figure 10.7: NWS general scheme

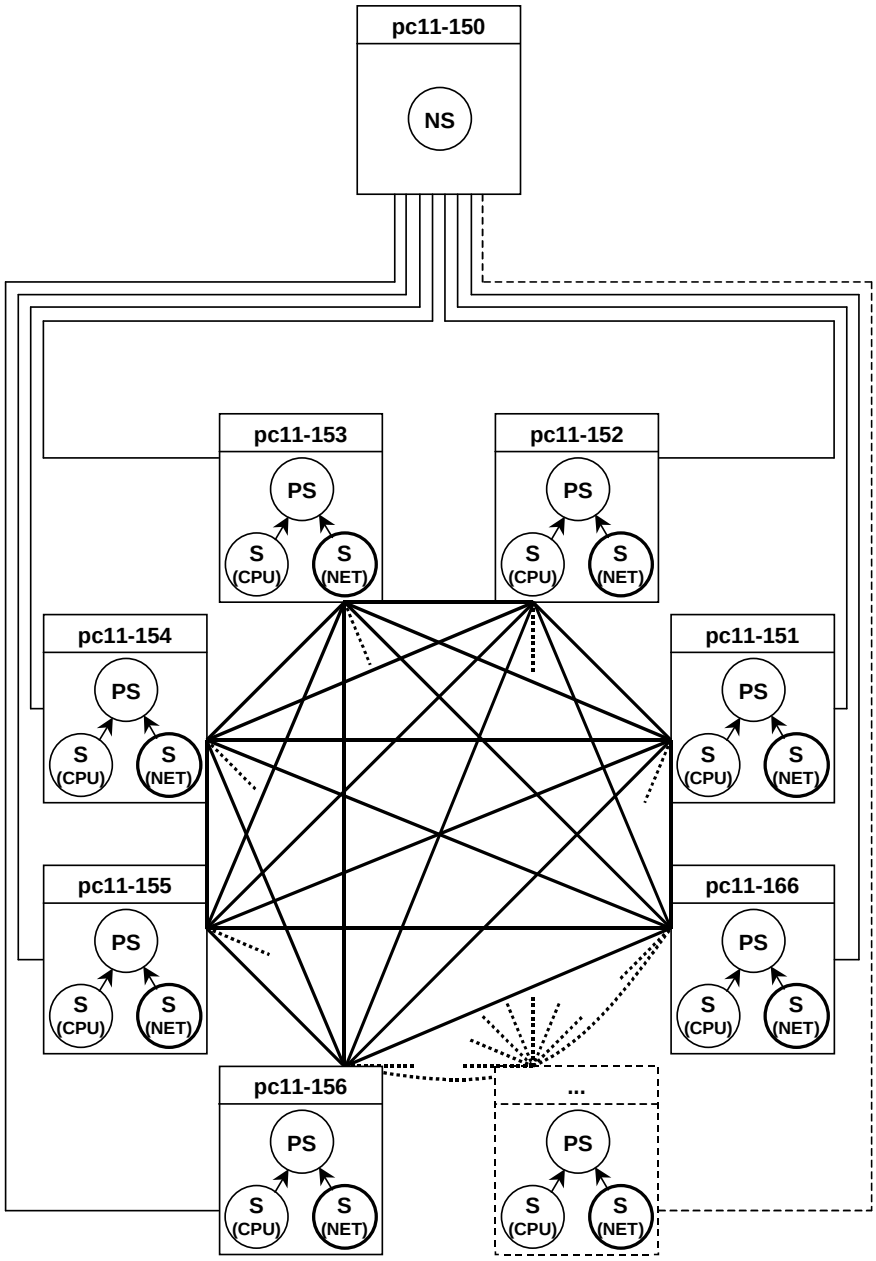


Figure 10.8: Deployment of NWS

In figure 10.9 a partitioning of the graph $grid20x19$, corresponding to a situation when all processors have the same performance, is illustrated. In figure 10.10, a partition issued from the above partition as a result of changing the capacity of the red processor is shown. The effect of the migration, as a result of the modifications of some processors, can be better in figure 10.11.

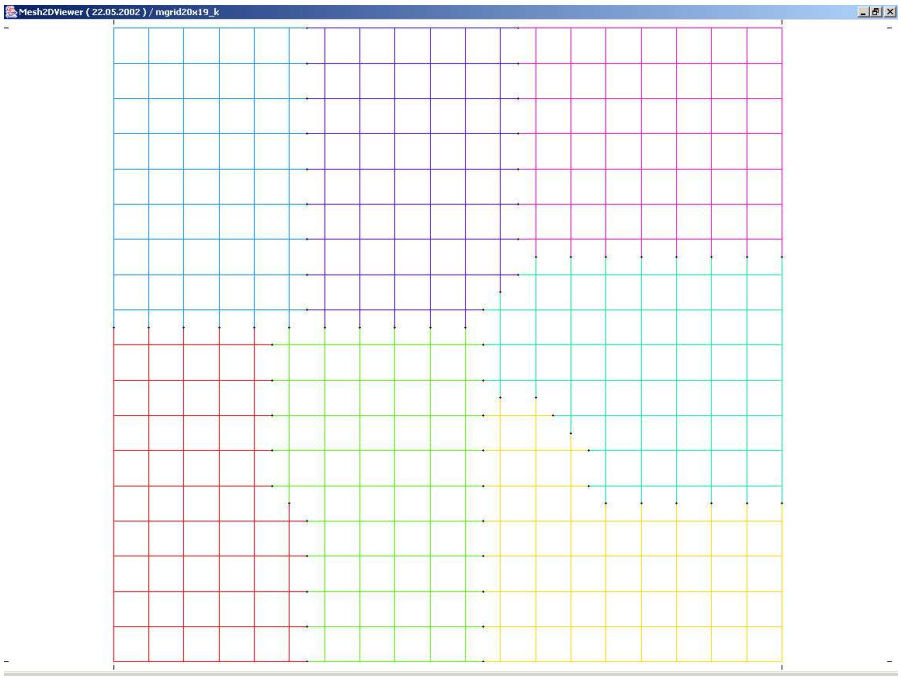


Figure 10.9: Homogeneous partitioning

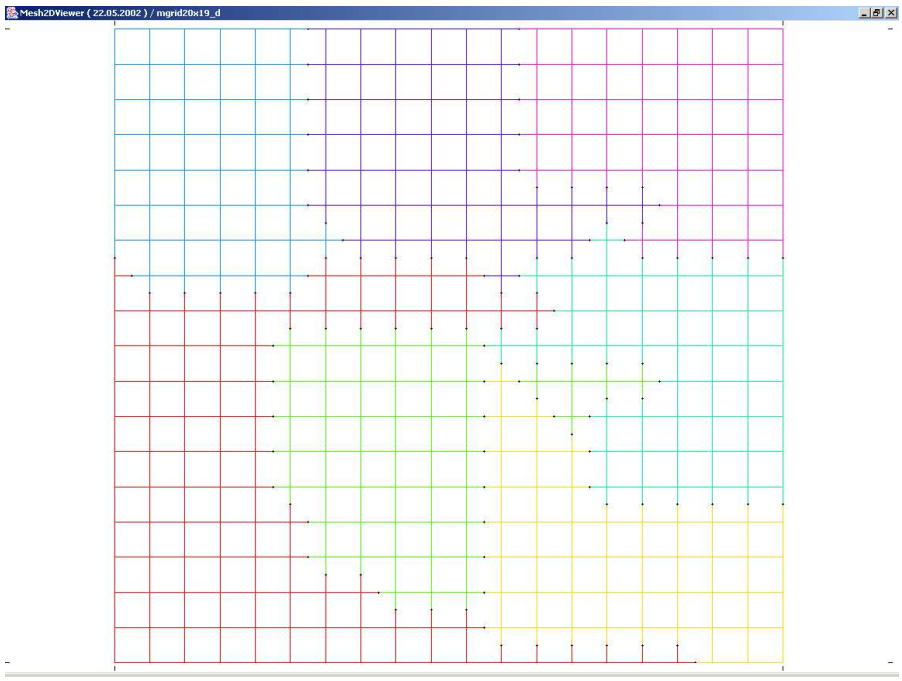


Figure 10.10: Heterogeneous partitioning when the red processor has double capacity than of the others

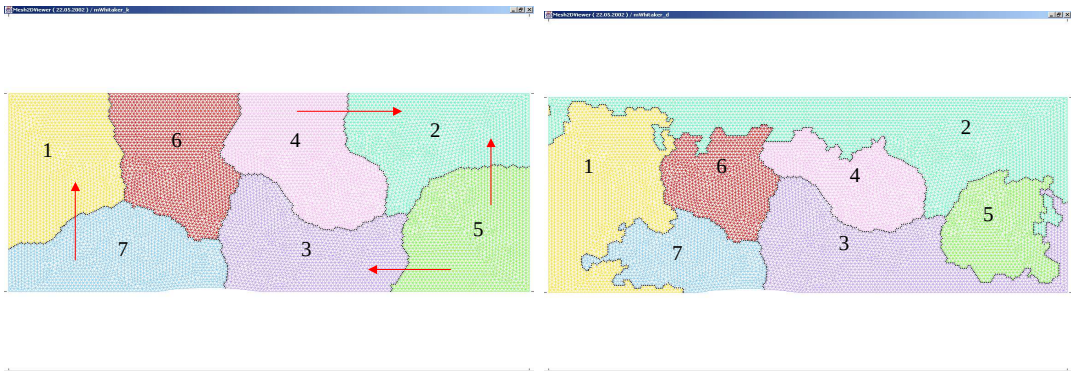


Figure 10.11: Visualization module. On the the left hand side, a homogeneous partitioning of the graph Whitaker is shown. On the right hand side, a partitioning resulted after redistribution is shown, for the case when the processors 1, 2 and 3 have double available processing capacities compared to the others.

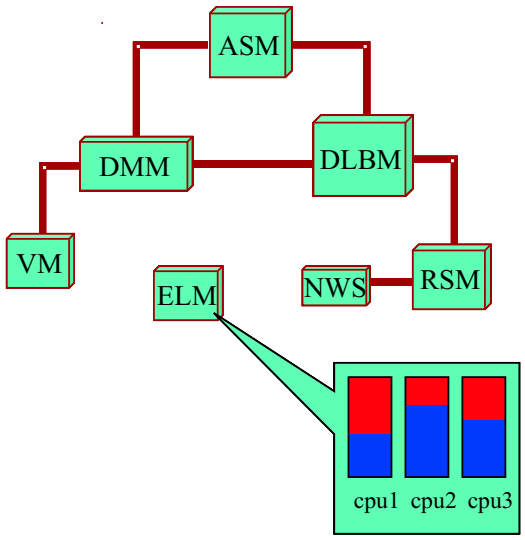


Figure 10.12: Extra-load simulation module

- **The Extra Load Simulation Module**

  This module allows to artificially generate loads on processors and congestion on network. It's role is to simulate the activity of external parallel applications in the system following a scenario defined by the user. The main idea that have been used is that the activity of a parallel task can be decomposed into a suite of interleaving phases of computation, communication or idleness. The behaviour of the external tasks is specified through configuration files. Each configuration file specifies a suite triples $(Data, Work, Wait)$ where $Data$ is a collection of pairs (p, c) indicating that an amount of data of size c should be sent to the processor p, $Work$ specifies the amount of work to be done by the task, $Wait$ specifies the interval of time it must stay idle.

Table 10.1: Execution times for different communication topologies

Topology	GDA	PCG	LPA	MPA
Path	19.82	0.45	0.32	0.28
Ring	6.69	0.59	0.39	0.30
Star	8.59	0.15	0.20	0.46
SD_whitaker	2.40	0.32	0.24	0.22
SD_airfoil	3.55	0.43	0.23	0.18
SD_grid20x19	2.05	0.16	0.20	0.15
SD_shock	3.97	0.39	0.23	0.20

10.2 Experiments

We used a computational grid that we built with commodity hardware available in our institute. This consists in a collection of 16 DELL PC's. In a first step, we focused our attention mainly on testing the inter-operability between the dynamic load balancing module and the resource monitoring module. A test suite was generated by using different load distribution and communication topologies provided by the data management module. The resource monitoring module provided online values for network and processor parameters. Typically, each processor knows what is its current load, who are its logical neighbors and its absolute capacity. For finding out forecasts for the available CPU fraction, latency and bandwidth on the links relying it with its neighbors, it sends requests to the resource monitoring module. The algorithms proposed for keeping the fairness at runtime were tested with subdomain graphs obtained from real test graphs used in CFD. The proposed dynamic load balancing methods were used with a multilevel scheme inspired by those provided by the ParMETIS package. During the preliminary experiments performed, the weights w_i appearing in the synthetic model were set to the square of the inverse of the forecasted bandwidth along the corresponding link.

Tests were performed with several dynamic load balancing algorithms. Table 10.1 reports results in terms of execution times expressed in seconds for the algorithms GDA1, PCG, LPA and MPA for a highly imbalanced load distribution. The tested topologies were path, ring, star and subdomain graphs of some well known unstructured meshes (whitaker, airfoil, grid20x19 and shock). We further performed tests with HeRMeS on this cluster. We simulated an application that matches the scheme indicated in the algorithm 2. We considered several types of meshes and we assumed that their

Table 10.2: Meshes used during the tests

Graph	nb. of vertices	nb. of edges
grid20x19	380	1442
stufe	1036	3736
L_9	17983	71192
Whitaker	9800	57978
airfoil	4253	24578
3elt	4720	27444

vertices had weights associated. These weights represent a measure of the complexity of the work associated with the corresponding vertices. During the experiments we considered a fixed number of 7 adaptions. In each adaption phase, the weights of a subset of vertices, among the vertices of all subdomains, were multiplied by a factor of 2. This had as effect the increase of the complexities of the work assigned to processors and the deterioration of an eventually already existent fair distribution. The size of subdomains may dramatically increase after the adaptions phases. As an example, the start distribution that was considered in the case of the mesh L_9 is as shown in the table 10.3. When no method for load balancing was used, the load distribution after the last adaption phase was as shown in the table 10.4(characterized by an imbalance factor of 1.94, i.e. $\max_i\{l_i/\overline{l}_i\} = 1.94$).

The work complexities were considered proportionally to the number of nodes in a subdomain multiplied by a scaling factor M. We considered the following test cases:

- Adaptive, with homogeneous capacities.

  A situation that was considered was when the capacities of the machines had similar values i.e. 0.667. In this case the main source of load imbalance was the successive application of adaptive procedures. It was considered that the load of a processor is proportional to the number of the weights of vertices in the assigned subdomain. The work performed by a processor consisted in repeating a fixed set of basic operations for a number of times that is the product between a scaling factor and the total weight of a subdomain. In this case we set the value of the scaling factor to 50.

  In the table 10.5, the execution times for the simulation with various dynamic load balancing techniques are illustrated. The first column contains execution times of the simulation in the case when no dynamic load balancing method is applied. The second column corresponds to a

Table 10.3: The initial loads assigned to processors

$load_1 = 1202$	$load_5 = 1208$	$load_9 = 1193$	$load_{13} = 1203$
$load_2 = 1206$	$load_6 = 1192$	$load_{10} = 1205$	$load_{14} = 1199$
$load_3 = 1196$	$load_7 = 1199$	$load_{11} = 1201$	$load_{15} = 1191$
$load_4 = 1206$	$load_8 = 1190$	$load_{12} = 1192$	

Table 10.4: The loads assigned to processors after the last adaption phase

$load_1 = 25844$	$load_5 = 121096$	$load_9 = 31545$	$load_{13} = 88568$
$load_2 = 84635$	$load_6 = 105085$	$load_{10} = 91754$	$load_{14} = 50348$
$load_3 = 62928$	$load_7 = 141652$	$load_{11} = 4888$	$load_{15} = 1574$
$load_4 = 121206$	$load_8 = 54541$	$load_{12} = 104326$	

simulation that uses the generalized diffusion method (based on a generalized diffusion matrix given by the equation (5.6.1)). The third, the fourth and the fifth correspond to a simulation that uses the algorithms PCG, LPA and MPA, respectively. The last column corresponds to a simulation that uses a method from the ParMETIS library. In this case one can remark that the differences between the methods are not very important. However, the ParMETIS method may perform better in some cases than the other algorithms. This is partly due to the fact that these algorithms need to maintain and update information about the capacities of the processors and the network parameters.

- Adaptive, with heterogeneous capacities.

  In the table 10.7 the execution times for the simulation with various dynamic load balancing techniques are illustrated with the capacities varying between adaption phases as illustrated

Table 10.5: Execution times of an adaptive simulation with various dynamic load balancing methods

#vertices × #edges	No DLB	GDA1	PCG	LPA	MPA	ParMETIS
380 × 1442	373.00 s	157.16 s	132.13 s	132.03 s	145.21 s	129.98 s
1036 × 3736	1146.95 s	475.62 s	419.45 s	415.59 s	420.93 s	389.73 s
4253 × 24578	3913.53 s	1247.62 s	1159.51 s	1093.52 s	1179.26 s	1093.30 s
4720 × 27444	4977.12 s	1462.09 s	1224.85 s	1220.71 s	1316.33 s	1307.82 s
9800 × 57978	9785.57 s	3340.40 s	3097.48 s	3272.81 s	3288.05 s	3481.00 s
17983 × 71192	17851.39 s	5017.15 s	5657.72 s	4820.13 s	4809.60 s	4693.74 s

Table 10.6: Variation of the processing capacities between adaption phases

Processor	adpt 1	adpt 2	adpt 3	adpt 4	adpt 5	adpt 6	adpt 7
P_1	0.0667	0.1153	0.1176	0.1000	0.1111	0.1111	0.1290
P_2	0.0667	0.0384	0.1176	0.0500	0.1111	0.1111	0.1290
P_3	0.0667	0.1153	0.0588	0.1000	0.0666	0.0555	0.1290
P_4	0.0667	0.1153	0.0980	0.1000	0.0888	0.0555	0.0322
P_5	0.0667	0.0769	0.0784	0.1000	0.0888	0.1111	0.0967
P_6	0.0667	0.0384	0.0392	0.0500	0.0888	0.0555	0.0322
P_7	0.0667	0.0769	0.0784	0.0500	0.0222	0.0555	0.0322
P_8	0.0667	0.0384	0.0392	0.0500	0.0666	0.0555	0.0645
P_9	0.0667	0.0769	0.0588	0.1000	0.0444	0.0555	0.0645
P_{10}	0.0667	0.0384	0.0980	0.0500	0.0666	0.0555	0.0322
P_{11}	0.0667	0.1153	0.0588	0.0500	0.0666	0.0555	0.0967
P_{12}	0.0667	0.0384	0.0588	0.0500	0.0888	0.0555	0.0645
P_{13}	0.0667	0.0384	0.0392	0.0500	0.0444	0.0555	0.0322
P_{14}	0.0667	0.0384	0.0392	0.0500	0.0222	0.0555	0.0322
P_{15}	0.0667	0.0384	0.0196	0.0500	0.0222	0.0555	0.0322

Table 10.7: Execution times for an adaptive simulation with various dynamic load balancing methods when the capacities vary between adaptions

#vertices × #edges	No DLB	GDA1	PCG	LPA	MPA	ParMETIS
380 × 1442	149.19 s	381.68 s	77.03 s	68.01 s	70.47 s	90.28 s
1036 × 3736	956.27 s	223.76 s	174.57 s	161.22 s	175.00 s	302.86 s
4253 × 24578	2365.73 s	521.78 s	494.11 s	482.68 s	490.84 s	535.11 s
4720 × 27444	2571.26 s	586.67 s	527.38 s	517.08 s	533.05 s	630.33 s
9800 × 57978	6293.51 s	1159.93 s	1049.05 s	1037.39 s	1103.45 s	2745.69 s
17983 × 71192	7759.36 s	2103.14 s	1888.27 s	1872.55 s	1883.49 s	3586.92 s

in the table 10.6. The meaning is the same as in the above case, a column corresponds to the simulation with a an algorithm. A line corresponds to the simulation with different dynamic load balancing methods for a given graph/mesh. Here, one can see that generally our methods result in better execution times of the simulation compared to the ParMETIS method. This is due to that these methods distribute the loads "fairly" (i.e. proportionally to the capacities) while the ParMETIS method distributes them "equally". The application of GDA1 in the case of the graph grid requires special attention, because this shows that the application of a "costly" dynamic load balancing method may yield worse execution times than in the case when no method is used.

10.3 Conclusions

The tests showed that in a heterogeneous dynamic context as that we assumed here, the use of a dynamic load balancing method can contribute to the improvement of the execution time of a parallel application. In both of the cases considered above, the use of dynamic load balancing methods generally resulted in a significant improvement of the execution time of the application. However, the choice of a dynamic load balancing method must be done carefully, in order to get the maximum benefit from its application. As it is shown in the table 10.7, for the first graph, the application of GDA1 may be very costly, as the execution time of the application is worse than in the case when no method is used. In the case of a homogeneous repartitioning, the methods PCG, LPA, MPA performed similarly to the considered ParMETIS method. In some cases the ParMETIS method outperformed the other methods. This may happen because our schemes spend some additional time for maintaining/updating information about the current system's state. However, when the capacities vary between the adaption phases, the use of any of algorithms GDA1, PCG, LPA or MPA resulted in better execution times of the application, as they redistribute the loads fairly, i.e. proportional to the capacities, while the invoked ParMETIS method redistribute them "equally", being not able to take in account the heterogeneities at runtime.

Chapter 11

Conclusions and future work

It is widely accepted that the distributed computing environments constitute a major option for the future development of high performance computing. However, although they offer numerous advantages, it is difficult in practice to take the maximum benefit of their theoretical computing power. A major problem to be solved is how to ensure an efficient utilization of their resources. In such systems, the parameters characterizing the computational capacities and the communication may vary in unpredictible ways. On the other hand, the application itself may undergo dynamic changes, with a direct impact on the total execution time. This is the case of the applications that perform adaptive numerical computations. During the computation, the complexity of the work associated with a subdomain assigned to a processor may increase significantly, producing an important load imbalance. As opposed to the homogeneous case, a dynamic load balancing method should also explicitly take into account the system's characteristics.

In the present thesis, various methods for the fair dynamic load redistribution in heterogeneous computational environments are described and investigated. We started with the generalization of a variant of the diffusion algorithm proposed by Boillat. Complexity estimations were formulated. The balancing flow generated by such methods was shown to have an important property, i.e. it is a scaled projection of the other balancing flows. The immediate consequence is the fact that this flow is minimal with respect to a weighted Euclidian norm. We proposed the use of a parameter

that is close to the optimal values given in the homogeneous case. Compared to the usually suggested optimal diffusion, the variant we suggest does not need the computation of the eigenvalues of the generalized Laplacian matrix. The same algorithm, GDA, was proved to be theoretically faster than the hydro-dynamic algorithm. Faster polynomial schemes like the second order schemes (SOS) generally perform faster when compared to the generalized diffusion, but they suffer of the same inconvenient: they rely upon parameters that are hard to choose properly or are expensive to compute. For using such schemes in dynamic contexts, we suggested the use of a parameter dependent on an upper bound of the convergence factor that is easier to compute. The heterogeneous implicit schemes showed to generate the same balancing flow as the generalized diffusion and as the polynomial schemes. Another approach, more direct, (PCG) was investigated by solving an optimization problem by an adapted conjugate gradient approach that has been successfully used also in the homogeneous case. The balancing flow generated by diffusion-like schemes is computed iteratively, typically by accumulating at each step fractions of load differences. We gave a quantitative characterization of the flow computed by these methods. Starting from this, two other algorithms were given, one relying on the Laplacian polynomial (LPA), the other on the minimum generating polynomial of the generalized Laplacian of the communication graph (MPA). The experiments performed showed that the three algorithms PCG, LPA, MPA perform significantly faster than GDA or SOS, but none of them performs best in all the situations. The structure of the communication topology strongly influences their execution times. This suggests the use of a hybrid algorithm, including a heuristics capable of choosing the adequate algorithm based on the characteristics of the communication graph.

We further showed that an incremental algorithm, capable of computing a new balancing flow by taking into account the previously computed one, may perform better than one that starts the computation from scratch.

All these methods were incorporated into a tool, called HeRMeS that allowed to perform tests with a simulated adaptive numerical application. The tests with HeRMeS showed that applying dynamic load balancing methods for an application of the type we considered may effectively result in a reduction of the execution time. Secondly, the tests showed that the method to be used must be chosen carefully, as an inadequate algorithm may increase the execution time of the application.

Thirdly, as seen in the previous chapter, when a homogeneous method is used in a heterogeneous context, the execution time of the application might be much higher than in the case when adequate heterogeneous methods are used. This emphasizes the importance of the study of dynamic load balancing techniques. Fast methods that minimize their overhead are necessary for that.

An interesting direction of research is the examination of the possibility to apply these methods (or variants of them) to other kinds of applications. Another interesting topic is the application of these methods in larger heterogeneous computing systems (grids).

Bibliography

[1] S. T. Barnard and H. D. Simon. A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency: Practice and Experience*, 6(2):101 –117, April 1994.

[2] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. SIAM, Philadelphia, PA, 1994.

[3] Olivier Beaumont, Vincent Boudet, Arnaud Legrand, Fabrice Rastello, and Yves Robert. Heterogeneity considered harmful to algorithm designers. In *Cluster'2000*, pages 403–404. IEEE Computer Society Press, 2000.

[4] M. Berger and S. Bokhari. A partitioning strategy for nonuniform problems on multiprocessors. *IEEE Transactions on Computers*, (C36:5):570–580, 1987.

[5] D.P. Bertsekas and J . N. Tsitsiklis. *Parallel and distributed computation: numerical methods.* Prentice-Hall, Inc., 1989.

[6] J. E. Boillat. Load balancing and Poisson equation in a graph. *Concurrency: Practice and Experience*, 2(4):289–313, 1990.

[7] J. E. Boillat and P. G. Kropf. A fast distributed mapping algorithm. In *Proceedings of the joint international conference on Vector and parallel processing*, pages 405–416. Springer-Verlag New York, Inc., 1990.

[8] J. Chen and V. E. Taylor. Mesh partitioning for distributed systems: Exploring optimal number of partitions with local and remote communication. In *SIAM Conference on Parallel Processing*, 1999.

[9] J. Chen and V. E. Taylor. Parapart: Parallel mesh partitioning tool for distributed systems. *Concurrency - Practice and Experience*, 12(1):111–123, 2000.

[10] N. Chrischoides. New approach to parallel mesh generation and partitioning problem. *Computational Science, Mathematics and Software*, pages 319–344, Dec, 2001.

[11] F. R. K. Chung. *Spectral Graph Theory.* CBMS Lecture Notes. AMS Publication, 1996.

[12] F. R. K. Chung and P. Tetali. Isoperimetric inequalities for cartesian products of graphs. *Probability, Combinatorics and Computing*, 7:141–148, 1998.

[13] G. Cybenko. Dynamic load balancing for distributed memory multi-processors. *Journal of Parallel and Distributed Computing*, 7:279–301, 1989.

[14] T. Decker, M.Fischer, R. Lüling, and S.Tschöke. A distributed load balancing algorithm for heterogeneous parallel computing systems. In *Int. Conf. on Parallel and Distributed Processing Techniques and Applications*, pages 933–940, 1998.

[15] R. Diekmann, A. Frommer, and B. Monien. Efficient schemes for nearest neighbor load balancing. In G. Bilardi et al., editor, *Proc. European Symp. on Algorithms (ESA'98)*, volume 1461 of *Lecture Notes in Computer Science*, pages 429–440. Springer, 1998.

[16] R. Diekmann, S. Muthukrishnan, and M. V. Nayakkankuppam. Engineering diffusive load balancing algorithms using experiments. In G. Bilardi, A. Ferreira, R. Lueling, and J. Rolim, editors, *Solving Irregulary Structured Problems in Parallel (IRREGULAR '97)*, volume 1253 of *Lecture Notes in Computer Science*, pages 111–122. Springer, 1997.

[17] R. Elsässer, A. Frommer, B. Monien, and R. Preis. Optimal and alternating-direction loadbalancing schemes. In P. Amestoy et al. (ed.), editor, *Euro-Par 99, Parallel Processing*, volume 1685 of *LNCS*, pages 280–290. Springer, 1999.

[18] R. Elsässer, B. Monien, and R. Preis. Diffusive load balancing schemes on heterogeneous networks. In G. Bilard et al. (eds.), editor, *12th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, volume 1461, pages 30–38, 2000.

[19] R. Elsässer, B. Monien, and R. Preis. Diffusion schemes for load balancing on heterogeneous networks. *Theory of Computing Systems*, 35:305–320, 2002.

[20] C. Farhat. A simple and efficient automatic domain decomposer. *Computer and Structures*, 28(5):579–602, 1988.

[21] M. Fiedler. Bounds for eigenvalues of doubly stochastic matrices. *Linear Algebra Appl.*, 5(299-310), 1972.

[22] M. Fiedler. A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory. *Czechoslovak Mathematical Journal*, 25(100), 1975.

[23] I. Foster. *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering.* Addison-Wesley Publishing Company, Inc., 1995.

[24] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *Intl J. Supercomputer Applications*, 11(2):115–128, 1997.

[25] A. George and J. Liu. *Computer Solution of Large Sparse Positive Definite Systems.* Prentice-Hall,Englewood Cliffs, New Jersey, 1981.

[26] G. H. Golub and C. F. Van Loan. *Matrix Computations. 3rd ed.* Johns Hopkins Press, Baltimore, MD., 1996.

[27] Andrew S. Grimshaw, Wm. A. Wulf, and CORPORATE The Legion Team. The legion vision of a worldwide virtual computer. *Communications of the ACM*, 40(1):39–45, 1997.

[28] A. Heirich and S. Taylor. A parabolic load balancing method. In *International Conference on Parallel Processing*, volume III, pages 192–202, 1995.

[29] B. Hendrickson. Graph partitioning and parallel solvers: Has the emperor no clothes? In *Proc. Irregular'98, Lecture Notes in Computer Science*, volume 1457, pages 218–225, 1998.

[30] B. Hendrickson and K. Devine. Dynamic load balancing in computational mechanics. To appear in Comp. Meth. Applied Mechanics & Engineering.

[31] B. Hendrickson and T. Kolda. Graph partitioning models for parallel computing. *Parallel Computing*, 26:1519–1534, 2000.

[32] B. Hendrickson, R. Leland, and R. Van Driessche. Enhancing data locality by using terminal propagation. In *Proc. 29th Hawaii Intl. Conf. System Science*, 1996.

[33] A. Heririch and S. Taylor. A parabolic load balancing method. In *Proc. 24th International Conference on Parallel Processing*, 1995.

[34] G. Horton. A multi-level diffusion method for dynamic load balancing. *Parallel Computing*, 19:209–218, 1993.

[35] http://www-users.cs.umn.edu/ karypis/metis/metis.

[36] http://www-users.cs.umn.edu/ karypis/metis/parmetis.

[37] http://www.ccrl nece.technopark.gmd.de/DRAMA/Workshop/doi/sld042.htm.

[38] http://www.cs.njit.edu/sohn/sharp/index.html.

[39] http://www.gre.ac.uk/ c.walshaw/jostle/.

[40] http://www.lam mpi.org/.

[41] http://www.nersc.gov/ dhbailey/cs267/.

[42] http://www.uni paderborn.de/fachbereich/AG/monien/RESEARCH/PART/party.html.

[43] Y. F. Hu and R. J. Blake. An improved diffusion algorithm for dynamic load balancing. *Parallel Computing*, 25:417–444, 1999.

[44] Y. F. Hu and R. J. Blake. Load balancing for unstructured mesh applications. *Parallel and Distributed Computing Practice*, 2(3), September 1999.

[45] Y. F. Hu, R. J. Blake, and D. R. Emerson. An optimal dynamic load balancing algorithm. *Concurrency: Practice and Experience*, 10:467–483, 1998.

[46] C. C. Hui and S. T. Chanson. A hydro-dynamic approach to heterogeneous dynamic load balancing in a network of computers. In *Proceedings of the International Conference on Parallel Processing (ICPP)*, volume III, pages 140–147, Bloomingdale, USA,, August 1996. IEEE.

[47] C. C. Hui and S. T. Chanson. Theoretical analysis of the heterogeneous dynamic load balancing problem using a hydro-dynamic approach. *Journal of Parallel and Distributed Computing*, 43:139–146, September 1997.

[48] C. C. Hui and S. T. Chanson. Hydrodynamic load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 10(11):1118–1137, November 1999.

[49] J. Jájá. *An Introduction to Parallel Algorithms.* Addison-Wesley, 1992.

[50] Stuart J.Berkowitz. On computing the determinant in small parallel time using a small number of processors. *Information Processing Letters*, 18:147–150, 1984.

[51] M. T. Jones and P. E. Plassman. Parallel algorithms for the adaptive refinement and partitioning of unstructured meshes. In IEEE Computer Society Press, editor, *Proceedings of the Scalable High Performance Computing Conference*, pages 478–485, 1994.

[52] E. Kaltofen and B. D. Saunders. On Wiedemann's method of solving sparse linear systems. In H. F. Mattson, T. Mora, and T. R. N. Rao, editors, *Proc. AAECC-9*, volume 539 of *LNCS*, pages 29–38. Springer, 1991.

[53] G. Karypis and V. Kumar. Parallel multilevel k-way partitioning scheme for irregular graphs. Technical Report 96-036, Department of Computer Science and Engineering, University of Minnesota, 1996.

[54] A. Kelmans, I. Pak, and A. Postnikov. Tree and forest volumes of graphs. 2000-03, DIMACS Technical Report, January 2000.

[55] B.W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Systems Technical Journal*, 29:291–308, 1970.

[56] S. Kumar, S. K. Das, and R. Biswas. Graph partitioning for parallel applications in heterogeneous grid environments. In *Proceedings of IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, Fort Lauderdale, FL, April 15-19 2002. IEEE Press.

[57] B. Litow, S. H. Hosseini, and K. Vairavan. Performance characteristics of a load balancing algorithm. *Journal of Parallel and Distributed Computing*, 31(2):159–165, 1995.

[58] L. Lovasz. Random walks on graphs: A survey. *Combinatorics, Paul Erdos is Eighty*, 2:1–46, 1993.

[59] J. L. Massey. Shift-register synthesis and bch decoding. *IEEE Transactions on Information Theory*, 15:122–127, 1969.

[60] C. D. Meyer. *Matrix Analysis and Applied Linear Algebra*. SIAM, 2000.

[61] B. Mohar. Some applications of Laplace eigenvalues of graphs. *Graph Symmetry: Algebraic Methods and Applications*, 497:225–275, 1997.

[62] M. Muthukrishnan, B. Ghosh, and M. Schultz. First- and second-order diffusive methods for rapid, coarse, distributed load balancing. In *Theory of Computing Systems*, volume 31, pages 331–354, 1998.

[63] B. Nour-Omid, A. Raefsky, and G. Lyzenga. Solving finite element equations on concurrent computers. *Parallel Computations and Their Impact on Mechanics*, ASME, 1987.

[64] L. Oliker and R. Biswas. Efficient load balancing and data remapping for adaptive grid calculations. In *Proc. of the 9th ACM Symposium on Parallel Algorithms and Architectures*, pages 22–25, Newport, Rhode Island, June 1997.

[65] Matei Ripeanu, Adriana Iamnitchi, and Ian Foster. Performance predictions for a numerical relativity package in grid environments. *The International Journal on High Performance Computing Applications*, 15(4):375–386, Winter 2001.

[66] T. Rotaru and H.-H. Nägeli. The generalized diffusion algorithm. Techn. Rep. RT-2000/06-1, Institut d'Informatique, Université de Neuchâtel, June 2000.

[67] T. Rotaru and H.-H. Nägeli. Heterogeneous dynamic load balancing. In Grigoras et al., editor, *NATO Advanced Workshop on Advanced Environments, Tools and Applications for Cluster*

Computing, volume 2326 of *Lecture Notes in Computer Science*, pages 136–144, Mangalia, September 2001. Springer.

[68] T. Rotaru and H.-H. Nägeli. Heterogeneous dynamic load balancing with a scheme based on the Laplacian polynomial. In Dongarra et al. Wyrzzykowski, editor, *Parallel Processing and Applied Mathematics*, volume 2328 of *Lecture Notes in Computer Science*, pages 107–114, Naleczow, September 2001. Springer.

[69] T. Rotaru and H.-H. Nägeli. Minimal flow generated by heterogeneous diffusion schemes. In *Interntional Conference On Parallel and Distributed Computing and Systems*, pages 136–141, Anaheim, USA, August 21-24 2001. ACTA Press.

[70] T. Rotaru and H.-H. Nägeli. A fast algorithm for fair dynamic load redistribution in heterogeneous environments. In *International Workshop on Parallel Matrix Algorithms and Applications*, Neuchatel, November 2002.

[71] K. Schloegel, G. Karypis, and V. Kumar. Multilevel diffusion schemes for repartitioning of adaptive meshes. Technical Report 97-013, Department of Computer Science and Engineering, University of Minnesota, 1997.

[72] K. Schloegel, G. Karypis, and V. Kumar. Parallel multilevel diffusion schemes for repartitioning of adaptive meshes. Technical Report 97-014, Department of Computer Science and Engineering, University of Minnesota, 1997.

[73] K. Schloegel, G. Karypis, and V. Kumar. Wavefront diffusion and LMSR: Algorithms for dynamic repartitioning of adaptive meshes. Technical Report 98-034, Department of Computer Science and Engineering, University of Minnesota, 1998.

[74] K. Schloegel, G. Karypis, V. Kumar, R. Biswas, and L .Oliker. A performance study of diffusive vs. remapped load-balancing schemes. Technical Report 98-018, Department of Computer Science and Engineering, University of Minnesota, 1998.

[75] H.D. Simon. Partitioning of unstructured problems for parallel processing. *Computing Systems in Engineering*, 2:135–148, 1991.

[76] A. Sinclair. Improved bounds for mixing rates of markov chains and multicommodity flow. *Combinatorics, Probability and Computing*, 1:351–370, 1992.

[77] A. Sinclair and M.R. Jerrum. Approximate counting, uniform generation and rapidly mixing markov chains. *Information and Computation*, 82:93–133, 1989.

[78] S. Sinha and M. Parashar. Adaptive runtime partitioning of AMR applications on heterogeneous clusters. In *Proceedings of the 3rd IEEE International Conference on Cluster Computing*, page 435 442, Newport Beach, CA, 2001. IEEE Computer Society Press.

[79] M. Soltys. Berkowitz's algorithm and clow sequences. *The Electronic Journal of Linear Algebra*, 9(2):42–54, April 2002.

[80] W. B. Tan and P. Strazdins. The analysis and optimization of collective communications on a beowulf cluster. In *Proceedings of ICPADS'02: 2002 International Conference on Parallel and Distributed Systems*. IEEE Press, Taipei, December 2002.

[81] R. van Driessche and D. Roose. Dynamic load balancing with a spectral bisection algorithm for the constraint graph partitioning problem. In *Lecture Notes in Computer Science*, volume 919, pages 392–397, 1995.

[82] R. S. Varga. *Matrix Iterative Analysis*. Series in Automatic Computation. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1962.

[83] A. Vidwans, Y. Kallinderis, and V. Venkatakrishnan. Parallel dynamic load-balancing algorithm for 3-dimensional adaptive unstructured grids. *AIAA Journal*, 32(3):497–505, 1994.

[84] C. Walshaw and M. Cross. Multilevel mesh partitioning for heterogeneous communication networks. Technical Report 00/IM/57, University of Greenwich, London SE10 9LS, UK, March 2000.

[85] C. Walshaw, M. Cross, and M. Everett. Dynamic load-balancing for parallel adaptive unstructured meshes. In M. Heath et al, editor, *Parallel Processing for Scientific Computing*, Philadelphia, 1997. SIAM.

[86] C. Walshaw, M. Cross, and M. Everett. Parallel dynamic graph partitioning for adaptive unstructured meshes. *Journal of Parallel and Distributed Computing*, 47(2):102–108, 1997.

[87] J. Watts. *Dynamic Load Balancing and Granuarity Control on Heterogeneous and Hybrid Architectures.* PhD thesis, California Institute of Technology, Pasadena, California, 1998.

[88] J. Watts, M. Rieffel, and S. Taylor. Dynamic management of heterogeneous resources. In *High Performance Computing: Grand Challenges in Computer Simulation*, pages 151–6, April 1998.

[89] J. Watts and S. Taylor. A practical approach to dynamic load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 9:235–248, 1998.

[90] D. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Transactions on Information Theory*, 32(1):54–62, January 1986.

[91] M.H. Willebeeck-LeMair and A.P. Reeves. Strategies for dynamic load balancing on highly parallel computers. *IEEE Transactions on Parallel and Distributed Systems*, 4(9):1305–1336, 1993.

[92] R. Wolski, N. Spring, and J. Hayes. The network weather service: A distributed resource performance forecasting service for metacomputing. Technical Report CS98-599, UCSD, 1998.

[93] C. Xu and F. Lau. *Load Balancing in Parallel Computers Theory and Practice.* The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, 1997.