Downloaded from UvA-DARE, the institutional repository of the University of Amsterdam (UvA)
http://dare.uva.nl/document/107491

File ID	107491
Filename	Chapter 5 Parallelization and adaptive load balancing on the grid

SOURCE (OR PART OF THE FOLLOWING SOURCE):

Type	Dissertation
Title	A virtual reactor for simulation of plasma enhanced chemical vapor deposition
Author	V.V. Krzhizhanovskaya
Faculty	Faculty of Science
Year	2008
Pages	116
ISBN	9789090231662

FULL BIBLIOGRAPHIC DETAILS:
http://dare.uva.nl/record/273799

Copyright

It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other then for strictly personal, individual use.

UvA-DARE is a service provided by the library of the University of Amsterdam (http://dare.uva.nl)

Chapter 5. Parallelization and Adaptive Load Balancing on the Grid*

5.1. Introduction

Porting complex distributed applications to distributed resources like the Grid poses a grand challenge to the computer and computational sciences, mostly due to the dynamical and decentralized nature of the Grid. Involving parallel computational solvers further complicates the problem because of the severe heterogeneity of Grid resources characterized by a wide range of processors and network communications performance. Lately, the scientific community has been investing lots of efforts into development of Grid-aware problem solving environments (PSE) for complex applications [2,3]. In the next Chapter (Chapter 6), we will discuss the integration of the Virtual Reactor into such a PSE. In this Chapter we address the challenge to efficiently execute the PECVD simulations on the Grid.

Grid computing technologies opened up new opportunities to access virtually unlimited computational resources, and inspired many researchers to work on adaptation of parallel methods and to develop new mechanisms for distributed applications on the Grid. The PECVD Virtual Reactor discussed in this thesis serves as a test-case driving and validating the development of the Russian-Dutch computational Grid (RDG) for distributed high performance simulation [4].

The Virtual Reactor is an excellent example of the multiphysics applications spanning a wide range of spatial and temporal scales [5,6]. Simulation of three-dimensional flow with chemical reactions and plasma discharge in complex geometries is one of the most challenging and demanding problems in computational science and engineering, requiring both high-performance and high-throughput computing.

The Russian-Dutch Grid provides a real-life hardware background for this research as it contains sites with both homogeneous and heterogeneous computing and networking resources. To execute a Grid-enabled complex simulation, a proper functional decomposition of the constituting components is required. To assure that the components, especially computational modules, are distributed efficiently it is necessary to carry out performance evaluation of the individual modules on Grid resources and draw generic conclusions on their behavior: how scalable they are depending on input data and resources used, what is the possible achievable speedup, how infrastructure properties influence the application performance, etc.

A countless number of parallel applications have been developed for traditional (i.e. static homogeneous) parallel systems. Porting such applications from homogeneous

* Parts of this chapter were published in [1]

computing environments to dynamic heterogeneous computing and networking resources poses a challenge to keep up a high level of application efficiency. To assure efficient utilization of Grid resources, special methods for workload distribution control should be applied. Proper workload optimization methods should take into account two aspects: (1) the application characteristics (e.g. the amount of data transferred between the processes, amount of floating point operations and memory consumption) and (2) the resource characteristics (e.g. processors, network and memory capacities, as well as the level of heterogeneity of the dynamically assigned resources). The method should be computationally inexpensive not to induce a large overhead on application performance. In this chapter we present such a method and evaluate it using one of the core parallel solvers of the Virtual Reactor.

The issue of load balancing in a Grid environment is addressed by a number of research groups. Generally studies on load balancing consider distribution of processes to computational resources on the system/library level with no modifications in the application code [7,8]. Less often, load balancing code is included into the application source-code to improve performance in specific cases [9,10]. Some research projects concern load balancing techniques that use source code transformations to improve the execution of the application [11]. We employ an application-centric approach where the balancing decisions are taken by the application itself. The algorithm that estimates the available resources and suggests the optimal load balancing of a parallel job is generic and can be employed in any parallel application to be executed on heterogeneous resources.

A detailed description of global load optimization approaches for heterogeneous resources and adaptive mesh refinement applications is given in [12,13,14]. However, in [12] and [14] no network link heterogeneity was considered and only static resource estimation (initialization) was performed in [12] and [13]. These two issues are the major challenges of Grid computing: 1) the heterogeneity of the network links can be an order of magnitude higher that of the processing power; and 2) Grid resources are inherently dynamic. Developing our algorithm, we tried to address specifically these two issues. The approaches discussed in [12] and [14] are only valid for batch sequential applications (specifically for the queuing systems and computer cluster schedulers), whereas our effort is directed towards parallel programs utilizing heterogeneous resources.

5.2. Parallelization

In order to fully appreciate the distributed nature of the Grid environment on which the virtual reactor will run, we need a distributed, loosely coupled Single Processor Multiple Data-stream (SPMD) type of parallelization [15]. We investigate various levels of parallelization [16]:

5.2.1. Job-level parallelism

Because of the large number of parameters that influence the film properties and deposition rate (pressure, gas and substrate temperatures, flow velocity, reactor volume and configuration, mixture composition and numerous plasma characteristics), for thorough investigation of the system behavior many tasks with different initial conditions can be run

in parallel on different processors, providing exploration of a large parameter space. This way of job-level parallelization (also known as Capacity computing) is the most natural and efficient approach for many computational fluid dynamic problems that require significant efforts for parallelization of individual program algorithms and codes. To utilize the advantages of this approach, a task Manager system has was developed, looking after cluster processor idleness and loading them with new jobs. When one of the processors finished its job, the Manager initiates a process with new initial data set, running over the values of parameters studied (see Fig. 5.1).

Fig. 5.1 Scheme of job-level parallelization.

5.2.2. Task-level parallelism

Another level of parallelization, for an individual task, is also needed, as one of the aims of this project was the creation of a computational system for large-scale real-time three-dimensional simulation and visualization of complex-shaped industrial reactors.

A natural approach to solve transient computational fluid dynamics problems in parallel is based on decomposition of the computational domain into a number of sub-domains (equal to the number of processors) and on computation of each sub-domain by one of the processors, followed by the data exchange at each time step.

The efficiency of this approach essentially depends on the simplicity of the domain geometry, spatial uniformity of the processes and the communication / computation ratio. A typical spatial domain of a PECVD reactor is usually rather complex, so the trivial geometric decomposition makes sub-domains unbalanced in number of mesh points and total calculations to be done. A specific decomposition algorithm needs to be developed in order to alleviate this imbalance. It is based upon multi-block mesh decomposition with a beam distribution for parallelization. According to this concept the whole simulation domain is divided into a number of primitive blocks. For example, Fig. 5.2 (left) shows a two-dimensional complex domain consisting of nine blocks. Each block is divided into 4-face cells such that the whole domain has two families of beams: in “i” and “j” index directions. Each cell belongs to only one beam of each family and each beam includes a sequence of cells with adjacent faces such that one beam begins and finishes at the physical boundary. An advantage of this beam decomposition approach is that it supports the use of a uniform implicit scheme by applying sweep-type algorithms to the single beam.

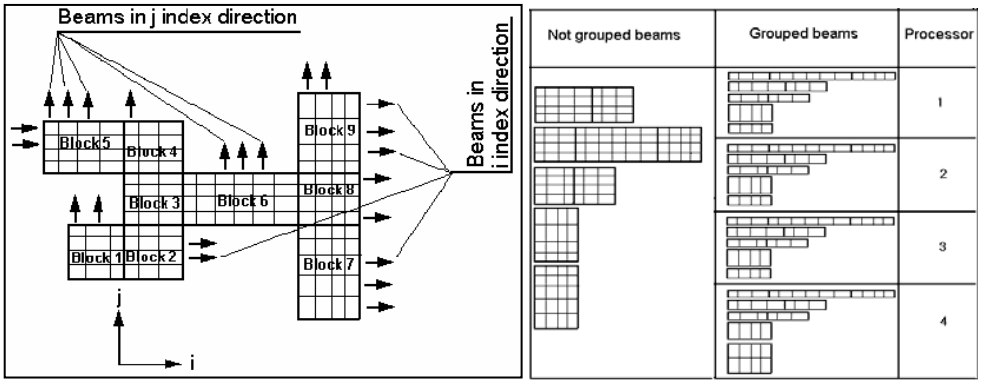

Fig. 5.2 Left: Decomposition algorithm for multi-block meshes and implicit numerical schemes. Right: Beam distribution for parallelization.

The parallel algorithm boils down to the following steps:

• Formation of two double-linked lists of beams in both index directions ("i" list and "j" list) for the given domain.

• Compilation of NP groups of beams inside each of two lists. (NP – number of processors), so that each group has approximately equal number of cells.

Fig. 5.2 (right) illustrates grouping of beams for the "i" list of our nine-block domain from for four processors. One can see that in this case loading of all processors is approximately equal. Similar grouping is made for the "j" list.

• Based on this beam grouping, the work instructions are formulated for each processor, listing the beams to be calculated by every processor.

• According to these instructions, in the beginning of each time step the master processor sends necessary information to slave processors. Each slave processor receives the data and computes the beams listed in its instructions.

• After performing its management duties, the master processor accomplishes its own calculation tasks and waits for results (decrements of conservative variables) from the slave processors. On receiving these results it merges all data arrays into one and repeats the previous step.

This algorithm for 1 time step is briefly illustrated in Fig. 5.3.

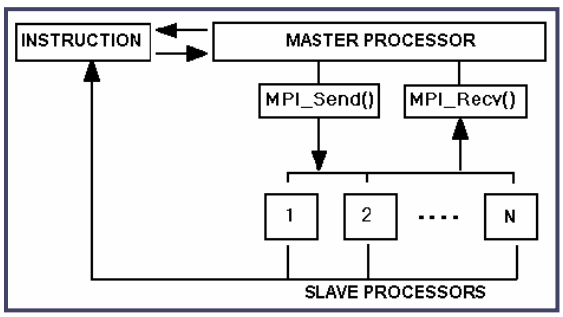

Fig. 5.3 Scheme of task-level parallelization algorithm.

In [16] some performance measurements on the task-level parallelization are shown, indicating the rapid saturation of the parallel efficiency due to the imposed communication overhead. The parallel efficiency was calculated as the ratio of t_{seq} to $n \cdot t_n$, where t_{seq} is the time consumed in a sequential mode and t_n is the computational time for a parallel algorithm to run on n processors. More details on the parallelization strategies and the associated performance results are provided in [16].

5.3. The simulation components of the Virtual Reactor

The Virtual Reactor includes the basic components for reactor geometry design; computational mesh generation; plasma, flow and chemistry simulation; editors of chemical processes and gas properties connected to the corresponding databases; pre- and postprocessors, visualization and archiving modules [5]. The aim of our research is to virtualize separate modules of the application and to run them efficiently as services and access them on the Grid.

The application components perform one (or a few) of the following functions: problem description, simulation, visualization and interaction. This is schematically shown in Fig. 5.4, where we emphasize the *simulation* components.

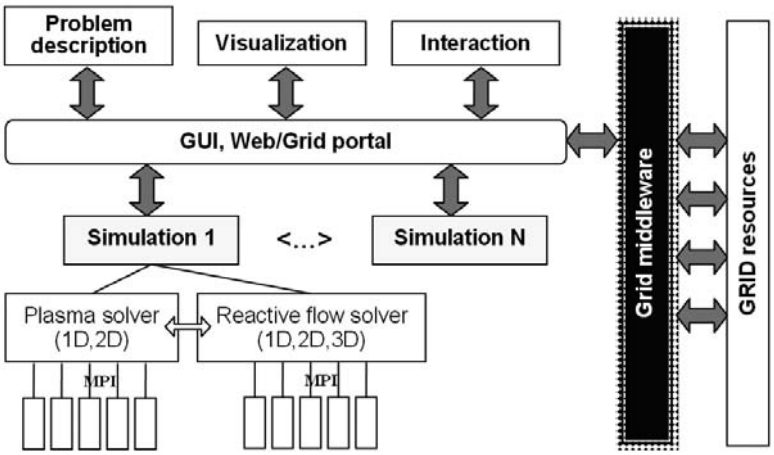

Fig. 5.4 A hierarchy of simulations. Plasma and Reactive flow solvers are the most time-consuming simulation components.

The core components are modules simulating plasma discharge, gas flow, chemical reactions and film deposition processes occurring in a PECVD reactor. The details on numerical methods and parallel algorithms employed in the solvers are described in previous chapters and in [16]. The most important features relevant to the Grid implementation are as follows: for stability reasons, implicit finite volume schemes were applied, thus forcing us to use a sweep-type algorithm for solving equations in every "beam" of computational cells in each spatial direction of the Cartesian mesh. A special parallel algorithm was developed with beams distributed among the processors.

Communications are organized exploiting a Master-Slave model; where at each simulated time step the Master prepares instructions for the Slaves, sends them the data to be processed, receives the results, and processes them before proceeding to the next step. The algorithm is implemented in an SPMD model, using the MPI message passing interface with MPI Barrier points for synchronization. Data exchange between the Master and the Slaves is repeated every time step, and simulation proceeds for thousands to millions of steps. In the testbed we use generic MPICH-P4 built binaries that can be executed on all the testbed machines using the Globus job submission service. To study the influence of various parameters on the simulated processes we run a number of simulations in parallel (shown in Fig. 5.4 as "Simulation 1" … "Simulation N" blocks) with the assistance of Nimrod-G [17].

To provide efficient execution of a parallel application on heterogeneous resources, one needs to clearly understand the application performance dependencies on homogeneous resources first. This gives an insight into the application scalability, induced fractional overhead, dependencies of the amount of the communications and calculations on the number of processors used, etc. The results of such tests can help estimating and predicting the behavior of the application on heterogeneous resources, thus simplifying the adaptation process.

5.4. The Russian-Dutch Grid testbed infrastructure

Generally the infrastructure of a site within a Grid testbed can be of one of the following types depending on the underlying resources:

I. traditional homogeneous computer cluster architecture: homogeneous worker nodes and uniform interconnection links;
II. homogeneous worker nodes with heterogeneous interconnections;
III. heterogeneous worker nodes with uniform interconnections;
IV. heterogeneous nodes with heterogeneous interconnections.

A complete Grid infrastructure is always of the Type IV, characterized by severe heterogeneity with a wide range of processor and network communication parameters. As we show later in this paper, the type of resources allocated to a parallel application significantly influences its performance, and different load balancing techniques shall be applied to different combinations of the resources. Currently the Russian-Dutch Grid testbed consists of six sites with different infrastructures:

Amsterdam-1 (contains 3 nodes, 4 processors) – Type IV;
Amsterdam-2 (32 nodes, 64 processors) – Type I;
St. Petersburg (4 nodes, 6 processors) – Type IV;
Novosibirsk (4 processors) – Type II;
Moscow-1 (13 nodes, 26 processors) – Type I;
Moscow-2 (12 nodes, 24 processors) – Type I.

The Russian-Dutch Grid testbed is built with the CrossGrid middleware [3] based on the LCG-2 distributions and sustains the interoperability with the CrossGrid testbed. More detailed information on the RDG testbed can be found in [4]. The RDG Virtual Organization (VO) is included into the CrossGrid VO, thus allowing the RDG certificate holders to access some of the CrossGrid resources and services. The CrossGrid testbed consists of 16 sites with the infrastructures of all 4 types.

5.5. Performance analysis on homogeneous resources

5.5.1. Benchmark approach

Benchmarking of a complex application is required to evaluate its performance and reveal the dependencies of its behavior on the underlying infrastructure. We use a structural approach to benchmark the Virtual Reactor as an example of a complex application. Within this approach, the overall functionality of the whole system is studied, followed by performance measurements of the individual components while they are not influenced by activities of the other components.

Benchmarking the components of a complex problem solving environment allows evaluating their performance depending on various parameters like types of input data and the resources used. This helps to predict the performance of a given component and use it for efficient resource allocation, thus improving the overall resource management within the whole application.

The earlier tests of the Virtual Reactor performed on the CrossGrid testbed showed that most of the interactive components of the Virtual Reactor do not put restrictions on the computer systems and network bandwidth and can be efficiently executed on distributed Grid resources [5]. Next, we focused on benchmarking of the *simulation* modules. Each simulation consists of two basic components: one for plasma simulation and another for reactive flow simulation (see Fig. 5.4). These two components exchange only a small amount of data every hundred or thousand time steps, therefore the network bandwidth is not a critical factor. Finally, we concentrate on benchmarking the individual parallel solvers, starting from a 2D PECVD solver which contains all the features of the 3D solver but takes less time to estimate the solver behavior on the Grid.

5.5.2. Benchmark setup

The goal of the benchmarking is to determine the scalability of the application, find out the limitations on the efficiency posed by the application architecture, resources and types of the simulations. Uncovering such details will allow us to optimize a resource management strategy for allocating the application components within the whole Virtual Reactor problem solving environment.

The solver operates a reactor geometry that is composed of a number of connected blocks. Different types of simulation can be performed within a single geometry: a chemically inactive flow and a flow with chemical and plasma processes. Physically the problem type is determined by the gas mixture composition, temperatures, pressures, and

the plasma discharge operation mode. From a computational point of view these types of simulations differ by the ratio of computations to communications: for simulating chemical processes the computational load is significantly higher.

We start from a light-weighted problem not simulating the chemical and plasma processes and with a simplified reactor geometry consisting of a single block that allows for easy tracking of parameter influences on the execution time. To measure the dependency of the solver performance upon the input data, multiparameter variation is applied. We measured the solver execution time, speedup and communication time depending on the combinations of input parameters: the computational mesh size, number of simulation time steps and number of processors.

The benchmark tests had to be automated because the parameter variation leads to a large number of job submissions. To solve this problem we have built an execution environment that supports a series of parameter-sweep job submissions in Globus. The environment is generic and can be used for any kind of performance benchmarks with user-defined metrics and parameters to be analyzed. Within this environment, the application to benchmark is described using some templates that are filled with particular application data (e.g. the Globus RSL template for job submission which also contains a list of input and output files). One of the functionalities of this execution environment is the support for parameter-sweep runs, analogous to what Nimrod-G or Condor-G provides. The advantage of our implementation is that we can specify the parameters (and their ranges) that shall be changed, as well as the characteristics to be measured and visualized automatically to analyze the influence of those parameters.

In these tests, a single-block topology was used. The block was subdivided into a (*ncell* x *ncell*) number of computational mesh cells, with ncell running from 40 to 100, thus forming 1600–10000 cells. We performed also some tests with real reactor geometries in order to check whether the reactor topology influences the parallel performance, since potentially it can introduce some load imbalance.

5.5.3. Influence of the number of time steps and reactor topology

Experiments with a different number of time steps showed that the execution time and other measured parameters are linearly proportional to the number of time steps, provided that this number is high enough and the standard output and hard disk operations are kept minimal (that means no excessive logging, nor any storing of the 2D fields or other additional files every time step). All the results presented below are measured for 100 time steps.

Along with the single-block geometry, we studied the performance of the solver with a complex multi-block PECVD reactor topology, which consists of an equivalent number of computational mesh cells. The results showed that all the measured characteristics of the solver behavior (execution time, speedup, computation and communication time) on the same resources do not differ for the single-block and multi-block topologies of equal number of cells within 1% accuracy. This assures us that the parallel algorithm used in the solver provides a good load balancing even in cases of complex topologies. Further we test the influence of the problem size (the number of mesh cells) with the single-block reactor

geometry, since it is easier to vary the mesh size arbitrarily with a single-block geometry than with a multi-block complex topology.

5.5.4. Speedup of the chemistry-disabled and chemistry-enabled simulations

The measurements were carried out on all the Grid sites within the RDG testbed. The parallel solver showed a noticeable speedup on the Moscow and Amsterdam sites of Type I (homogeneous cluster with uniform communication links). Fig. 5.5 and Fig. 5.6 demonstrate the total execution time and speedup of the parallel solver for different types of simulation: A chemistry-disabled "light-weighted" simulation and a chemistry-enabled "heavy" simulation respectively.

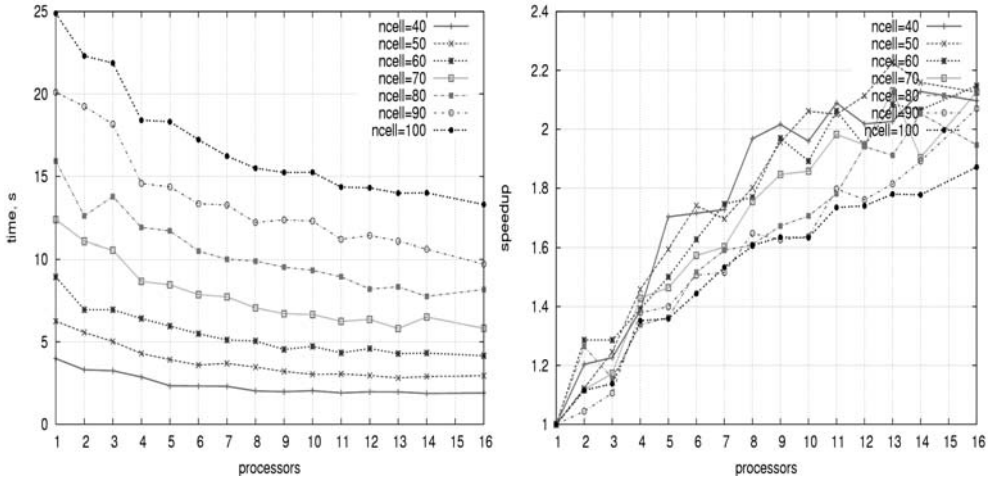

Fig. 5.5. Light-weighted (no chemistry) simulation: total execution time and speedup for different computational mesh sizes.

We observe different trends of the solver performance: for the light-weighted simulation, the speedup decreases with the increase of the mesh size (see the different curves in Fig. 5.5, right), while for the chemistry-enabled simulation, the speedup increases with increasing problem size (Fig. 5.6). The different absolute values of the speedup are mostly dependent on the resources: The results presented in Fig. 5.5 were obtained on the Moscow-1 site with slow inter-processor links, and Fig. 5.6 shows the results of the Amsterdam-2 site with fast communications. The gap between the *ncell*=90 and *ncell*=80, observed in Fig. 5.6 left, is most likely due to non-linear aspect of computations and communications, as well as cash saturation effects. We did not pursue deeper analysis of this result.

The same parallel solver tested on homogeneous Grid sites with a higher ratio of the inter-process communication bandwidth to the processor performance achieved much higher speedups. For instance on lisa.sara.nl with Infiniband interconnections it was 3 times higher for large problem size simulations. The type of MPI library also influences the parallel efficiency of a program: a specialized library optimized for the native communication technology (e.g. MPICH-GM for Myrinet communications on

das2.nikhef.nl) increases the speedup up to two times compared to the generic MPICH-P4 or MPICH-G2.

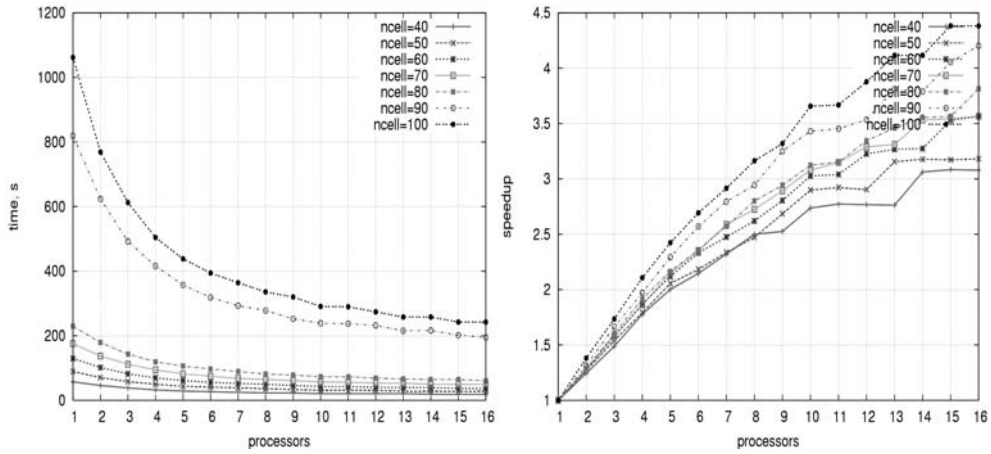

Fig. 5.6. Chemistry-enabled simulation: total execution time and speedup for different computational mesh sizes

5.5.5. Communication time trends

The time spent on inter-process communications within the solver is shown in Fig. 5.7 for different mesh sizes. The communication time was calculated as a sum of MPI Send/MPI Receive time on the master node over the total number of iterations.

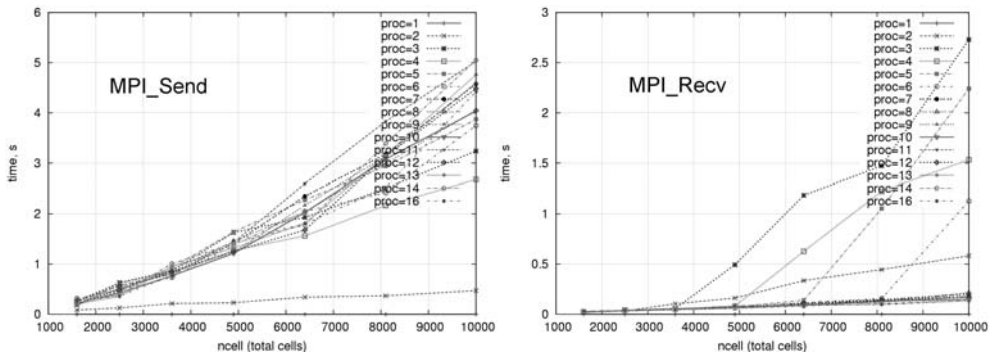

Fig. 5.7. Dependency of the communication time on the computational mesh size for different number of processors (light-weighted simulation)

We observe that communication time grows super-linearly with the increase in mesh size, although the amount of data transferred is linearly proportional to the number of mesh cells. The exact understanding of this behavior is not relevant and falls outside the scope of this thesis. The fact that the time spent for sending the data is not equal to the time of receiving the data in Fig. 5.7 can be explained by the type of measurements: both plots represent the data sent or received by the master node only, as a synchronizing process. As

one can see, the size of the received data (which represent the results of the calculations performed on the slave nodes) is significantly less than the data sent.

Some peculiarities in the communication time can be seen from Fig. 5.7, which are even more explicit in Fig. 5.8: (1) The communication time grows non-monotonically with the number of processors, but drops down a little on every processor with an even number; and (2) The time of MPI Receive calls is an order of magnitude higher for the larger meshes on the first few processors.

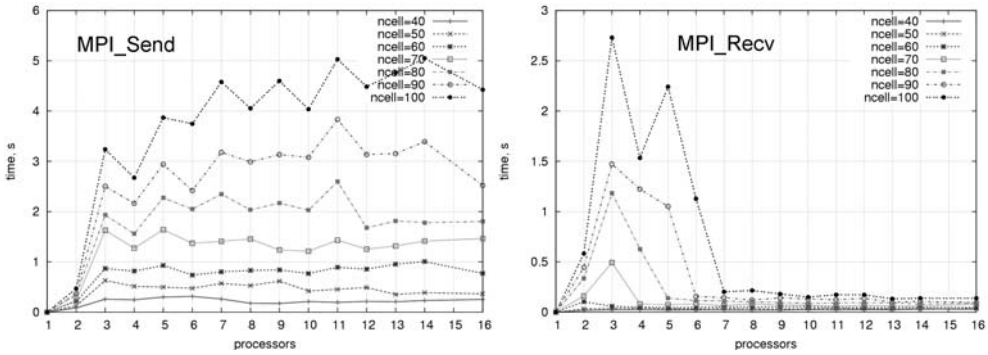

Fig. 5.8. Dependency of the communication time on the number of processors for different computational mesh sizes (light-weighted simulation)

5.5.6. Computation to communication ratio

In Fig. 5.9 the total execution time is presented along with the contributions of calculation and communication. For a smaller computational mesh, the communication time makes a relatively small contribution to the total execution time even for a large number of processors involved. For a larger mesh, the communication makes up to 30% of the execution time. This result confirms that the network bandwidth is not sufficient for this type of problem.

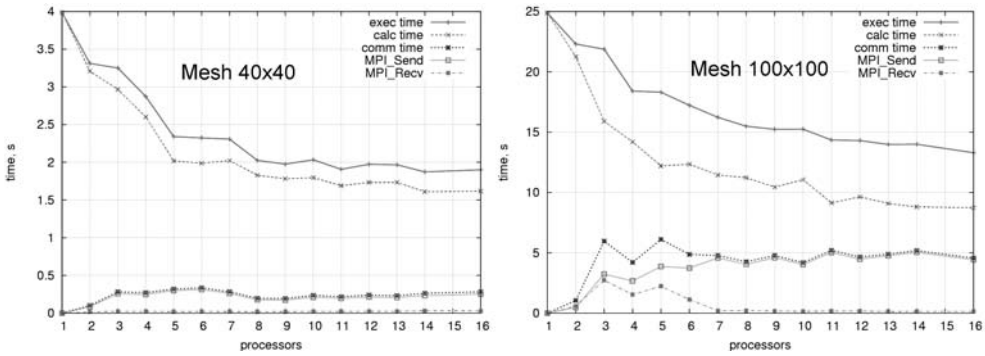

Fig. 5.9. Total execution time and contributions of the calculation and communication depending on the number of processors for different computational mesh sizes (light-weighted simulation)

Fig. 5.10 demonstrates the ratio of computation to communication time for different mesh sizes with different types of the simulation. The higher the ratio, the less communications are required, which obviously offers a better parallel efficiency and application scalability. The ratios in Fig. 5.10 explain the different speedup trends observed for chemistry-enabled and chemistry-disabled (light-weighted) simulations. From the presented graphs we can see that the behavior of this ratio does not depend on the mesh size for the chemistry-enabled simulations, while this behavior for the light-weighted simulations significantly differs for small and large mesh sizes. For a small mesh size, the ratio stays decently high, and for 6 processors and more it reaches the level of the chemistry-enabled simulations. For a larger mesh, the computation/communication ratio for the no-chemistry simulations is very low, thus diminishing the overall parallel efficiency.

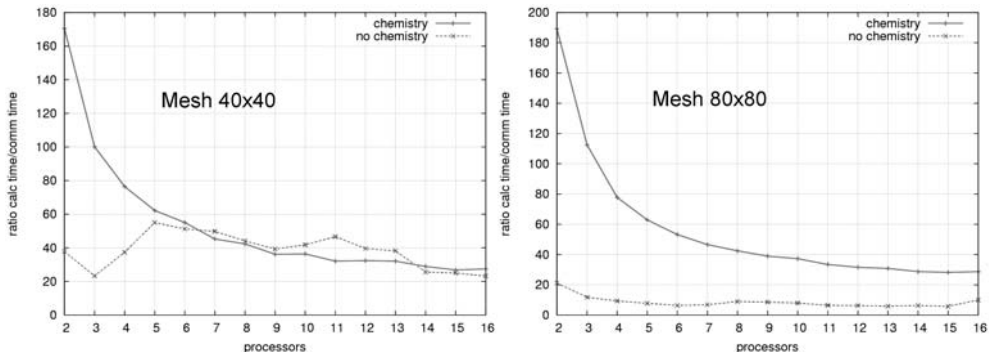

Fig. 5.10. The ratio of the computation to communication time for chemistry-enabled and light-weighted simulations

5.5.7. Discussion of the results for homogeneous resources

The results presented in this section show that the parallel speedup is lower for a larger problem size (with more computational mesh cells) for the simulations of problems without chemical processes. This fact indicates that the ratio of the inter-process communication bandwidth to the processor performance was not high enough for light-weighted problems with relatively small number of operations per computational cell. It means that for optimal usage of the computing power, a large number of processors for one parallel run shall only be used for relatively small computational meshes. Thus the communication technology puts a limit to the scalability of the solver for this problem type. On the other hand, the simulation of the flow *with* chemical processes shows higher speedup with larger meshes (see Fig. 3). Here the amount of computations brought by simulating the chemistry changes the behavior of the solver qualitatively. This leads us to the conclusion that different resource allocation strategies should be applied for different types of simulation and meshes used.

The results in Fig. 5.8 reflect the network and nodes features of the tested Grid site:

1. Since the site consists of dual nodes, the network channels work more efficiently for data transfers between the Master and a Slave processor if a connection was already established with another Slave processor on the same node. This can be explained by

implementation of the MPI library which saves network resources while opening and maintaining connections for concurrent processes on the same node.

2. The "peaks" of the MPI Receive time for the first few processors (see Fig. 5.8, right) are caused by the constraints on the size of the data that could be accommodated at once. The constraining factors can be the network bandwidth distribution, the processor cache size, the memory available on the node, or a combination of these factors.

5.6. Application performance on heterogeneous resources

The RDG Grid sites with heterogeneous processors and/or network links (Types II, III, IV) provided only a limited parallel speedup or even a slow-down of the original solver with a homogeneous parallel algorithm (data not shown). This was inevitable since, in addition to the low-bandwidth links, these sites are characterized by very diverse resources: the processor and network parameters differ by orders of magnitude for different nodes.

The parallel algorithm of the solver was originally developed for homogeneous computer clusters with equal processor power, memory and inter-processor communication bandwidth. In case of submitting equal portions of a parallel job to the nodes with different performance, all the fast processors have to wait at the barrier synchronization point till the slowest ones catch up, thus the effect of slow-down on heterogeneous resources is not surprising. The same problem occurs if the network connection from the Master processor to some of the Slave processors is much slower than to the others. As we have shown in the previous section, for communication-bound simulations (chemistry-disabled simulation with large computational meshes), the communication time on low-bandwidth networks is of the order of the calculation time, therefore the heterogeneity of the inter-processor communication links is a hindrance as considerable as the diversity of the processor power. One of the natural ways to adapt the solver to the *heterogeneous* Grid resources is to distribute the portions of job among the processors according to the processor performance and network connections, *taking into account the application characteristics*. To adapt the parallel solver, we developed a new method presented in the next Section.

5.7. A new methodology for resource-adaptive load balancing on heterogeneous resources

One of the factors that determine the performance of parallel applications on heterogeneous resources is the quality of the workload distribution, e.g. through functional decomposition or domain decomposition. Optimal load distribution is characterized by two things: (1) all processors have a workload proportional to their computational capacity; (2) communications between the processors are minimized. These goals are conflicting since the communication is minimized when all the workload is processed by a single processor and no communication takes place, and distributing the workload inevitably incurs communication overheads. Thus it is needed to find a trade-off and define a metric that characterizes the quality of workload distribution for a parallel problem. One of the existing methods to measure it is to introduce a cost function reflecting the application execution time. Minimization of this function corresponds to minimization of the application runtime.

The function should be simple and independent of the details of the code. The generic form of such a cost function is [18,19,20]:

$$H = H_{calc} + \beta H_{comm}, \tag{5.1}$$

where H_{calc} is minimized when the workload distribution among the processors is proportional to the processors capacity (or equal in case of homogeneous processors); H_{comm} is minimized when the communication time is minimal; and β is a parameter that can be varied in order to tune the balance between the calculation and communication terms. This parameter is dependent on the characteristics of both the application requirements and the resources capabilities.

The main generic parameters that define a parallel application performance are:

- An application parameter $f_c \sim N_{comm}/N_{calc}$ (N_{comm} and N_{calc} are the amounts of application communications and computations respectively);
- A resource parameter $\mu \sim t_{comm}/t_{calc}$ (t_{comm} is a typical time taken to communicate a single word between the processors, t_{calc} - typical time required to perform a generic floating point calculation).

The product of these two parameters $f_c\mu$ is often called the fractional communication overhead [18].

The goal of load balancing is to minimize the cost function (5.1). The parameter β in this expression is an aggregated value based on the application and resource specific parameters f_c and μ. Knowledge of these application and resource properties allows constructing an appropriate form of parameter β and performing suboptimal load distribution [20]. However in most real-life complex simulation problems, it is not possible to theoretically calculate the application specific parameter f_c with a reasonable precision. Even a detailed analysis of the algorithms and codes can fail in many practical cases when the code has multiple logical switches and completely different algorithms and computational schemes are used while solving a problem, depending on the initial conditions and computational parameters. Estimation of the resource-specific parameter μ also poses a challenge on heterogeneous Grid resources, since there is a multitude of processors with the ratio of communication to computation performance spanning a few orders of magnitude. Moreover, the Grid exhibits dynamic network and processor performance, therefore static domain decomposition fails to provide realistic estimations and consequently optimal load distribution. To ensure efficient load balancing of a parallel application on the Grid, it is necessary to estimate the β parameter experimentally. There are two possible approaches to that: (1) directly measure the lumped value of β for the application on the allocated resources and (2) separately benchmark the resources, estimate μ and then find out the application-specific parameter f_c that would provide an optimal workload distribution on a given set of resources. The first approach requires serious intrusion into the application code. This is certainly not desirable, especially when targeting to build a generic load balancing system which tries to abstract from the application specific issues. Thus we have chosen the second approach which is more generic and requires minimal modifications in the application code.

We have developed a meta-algorithm for adaptive load balancing on heterogeneous resources based on benchmarking the available resources capacity (defined as a set of individual resource parameters $\boldsymbol{\mu} = \{\mu_i\}$) and experimental estimation of the application parameter f_c. The algorithm ensures efficient load distribution, thus minimizing the application execution time. The cost function in our case is the experimentally measured execution time, which depends on the distribution of the workload between the participating processors. The target is to experimentally determine the value of f_c that provides the best workload distribution, i.e. minimal runtime of the application mapped to the resources characterized by parameter set $\boldsymbol{\mu}$.

The outline of the load balancing meta-algorithm is as follows:

1. Benchmark the resources dynamically assigned to a parallel application; measure the resource characteristics that constitute the set of resource parameters $\boldsymbol{\mu}$ (available processors power, memory and links bandwidth).
2. Estimate the range of possible values of the application parameter f_c. The minimal value is $f_c^{\min} = 0$, which corresponds to the case when no communications occur between the parallel processes of the application. The maximum value can be calculated based on the following reasoning: For the parallel processing to make sense, that is to ensure that running a parallel program on several processors is faster than sequential execution, the calculation time should obviously exceed communication time. For homogeneous resources this can be expressed as follows:
   $$\frac{T_{comm}}{T_{calc}} < 1 \quad \Leftrightarrow \quad \frac{N_{comm} t_{comm}}{N_{calc} t_{calc}} < 1 \quad \Leftrightarrow \quad f_c^{\max} = 1/\mu$$
   Likewise, for heterogeneous resources the upper limit can be found as:
   $$f_c^{\max} = \max(t^i{}_{calc}) / \min(t^i{}_{comm})$$
3. Run through the range of possible values of f_c with a discrete step. For each value of f_c calculate the corresponding load distribution based on the resource parameters $\boldsymbol{\mu}$ determined in step 1 (details on calculating the load distribution weights will follow this algorithm). With this distribution perform one or a few time steps/iterations, and measure the execution time. Proceed with the next value of f_c for the subsequent iterations, assuring that the simulation continues without delays, with a modified load distribution,
4. Analyze performance results for different values of f_c; find the optimal value f_c^* which provides the best performance of the application (i.e. minimal execution time).
5. Execute further calculations using the discovered f_c^*.
6. In case of dynamic resources where performance is influenced by other factors (which is generally the case on the Grid), a periodic re-estimation of resource parameters $\boldsymbol{\mu}$ and load re-distribution shall be performed.

7. If the application is dynamically changing (for instance due to adaptive meshes, moving interfaces or different combinations of physical processes modeled at different simulation stages) then f_c^* must be periodically re-estimated on the same set of resources.

Periodic re-estimations in steps 6 and 7 shall be performed frequently during the run-time of the application to correct the load imbalance with a reasonably short delay. The minimally required frequency of re-balancing can be estimated by calculating the relative imbalance introduced during the controlled period of time (the number of time steps/iterations).

The combination of $\boldsymbol{\mu}$ and f_c^* determines the distribution of the workload between the processors. To calculate the amount of workload per processor, we assign a weight-factor to each processor according to its processing power, memory and network connection. A similar approach was applied in [21] and in [10] for heterogeneous computer clusters, but the mechanism for adaptive calculation of the weights and application requirements was not developed there. Moreover, the tools developed for cluster systems cannot be used in Grid environments without modifications since static resource benchmarking is not suitable for dynamic Grid resources, where the weights shall be calculated every time the solver is started on a new set of dynamically assigned processors.

Let us assume that for the ith processor: p_i is the available processor performance (e.g. in Flop/s), m_i is the available memory (in MB) and n_i - available network bandwidth to the processor (in MB/s). An individual resource parameter μ_i then can be represented using the values of p_i, m_i, n_i. In a simple case when memory is considered only a constraining factor (and not driving the load balancing process) it is $\mu_i = p_i/n_i$. This resource parameter is widely used in scientific applications where the most important factor is the ratio of the computational power to the network bandwidth. In a more general case, two parameters shall be considered, μ_i and m_i. For the memory-driven applications, the ratio of the available memory to the network capacity of that processor m_i/n_i should play the major role in resource evaluation.

To reflect the processor capacity, we introduce a weighting factor w_i for each processor. It determines the final workload for a processor given by:

$W_i = w_i W$, where W is the total workload.

To determine the weighting factors we introduce parameters c_p, c_m and c_n that reflect computational, memory and communication requirements of the application. Then the weight of each processor is estimated using the following expression:

$$w_i = c_p p_i + c_m m_i + c_n n_i; \quad \sum_i w_i = 1. \tag{5.2}$$

This weighting factor w_i reflects a relative capacity of the resources according to the estimated infrastructure parameter $\mu_i = \mu(p_i, m_i, n_i)$ and the application parameter f_c. The infrastructure parameters μ_i can be determined by a set of benchmark runs before the actual calculations start (but after the resources have been assigned to the application).

Searching through f_c with fixed values of μ_i gives us the optimal value f_c^* which corresponds to the optimal mapping of the workload to the resources.

The parameters c_p, c_m and c_n depend not only on the application characteristics but also on the heterogeneity of the resources. Next we analyze how these parameters and weighting factors w_i are related to f_c and μ_i. Consider a situation where memory is only a constraining factor ($c_m = 0$). Then parameters c_p and c_n shall be proportional to the amount of application communications (computations) and the heterogeneity factors:

$$c_p \sim N_{calc}\varphi_{proc}; \quad c_n \sim N_{comm}\varphi_{net}. \tag{5.3}$$

Here φ_{proc} and φ_{net} are heterogeneity metrics of processors and network links. In case of equal network links the weighting should be done only according to the processors capacity, therefore the network heterogeneity parameter is nullified: φ_{net} =0. Analogously, for homogeneous processors φ_{proc} =0. The heterogeneity metrics of the network and computing resources can be defined as follows:

$$\varphi_{net} = \frac{\sum_{i=1}^{N}(n_i - n_{avg})^2}{N n_{avg}^{\ 2}}, \quad \varphi_{proc} = \frac{\sum_{i=1}^{N}(p_i - p_{avg})^2}{N p_{avg}^{\ 2}}. \tag{5.4}$$

Substituting expressions (3) for c_p and c_n in eq. (2), the weights can be re-written as:

$$w_i \sim N_{calc}\varphi_{proc}p_i + N_{comm}\varphi_{net}n_i \tag{5.5}$$

For the trivial cases:

$\varphi_{net} = 0$ (the network is homogeneous): $w_i \sim N_{calc}\varphi_{proc}p_i \sim p_i$

$\varphi_{proc} = 0$ (the processors are homogeneous): $w_i \sim N_{comm}\varphi_{net}n_i \sim n_i$

otherwise

$$w_i \sim N_{calc}\varphi_{proc}p_i + N_{comm}\varphi_{net}n_i \sim p_i + n_i f_c \frac{\varphi_{net}}{\varphi_{proc}}; \quad \sum_i w_i = 1 \tag{5.6}$$

Defining $\varphi = \varphi_{net}/\varphi_{proc}$ as an aggregate heterogeneity metric of resources and keeping in mind that $\mu_i = p_i/n_i$, we get: $w_i \sim p_i(1 + f_c\varphi/\mu_i)$

Considering $\vartheta_i = \frac{\mu_i}{\varphi}$ which combines the characteristics of the resource performance and heterogeneity we get:

$$w_i \sim p_i(1 + f_c/\vartheta_i) \tag{5.7}$$

Knowing the fractional overhead of the application and the heterogeneity level of the resources, we can optimize the workload distribution using this fast weighting technique. To evaluate the efficiency of the workload distribution we introduce the load balancing speedup Θ:

$$\Theta = \frac{T_{non-balanced}}{T_{balanced}} \cdot 100\% , \tag{5.8}$$

where $T_{non-balanced}$ is the execution time of the parallel application without the load balancing, and $T_{balanced}$ is the execution time using load balancing on the same set of resources. This metric is used to estimate the f_c^* that provides the best performance on given resources – the largest value of Θ in a given range of f_c^*. In a non-trivial case we expect to find a maximum of Θ and thus an optimal f_c^* for some workload distribution. Finite and non-zero value of f_c^* means that the application requirements fit best the resources in this particular workload distribution, which minimizes the total run-time of the application. The case of $f_c^* = 0$ while $\varphi \neq 0$ means that the application is totally computation dominated i.e. there is no communication between different processes, and the optimal workload distribution will be proportional only to the computational power of the processors. The case of $\varphi_{net} = 0$ means that we consider the resource infrastructure of heterogeneous processors connected by homogeneous network links and the value of f_c does not play a role – the distribution is again proportional only to the processing power.

In the discussion presented above while deriving eq. (5.4), we considered a simple case where memory requirements only put a Boolean constraint to the allocation of processes on the resources: either there is enough memory to run the application or not. But it can play a role in the load balancing process being one of the determining factors of application performance. This is the case for applications that are able to control memory requirements according to the available resources. In this case there will be additional parameters analogous to f_c and μ_i (or these functions will be more complex), but the idea and the load balancing mechanism remain the same.

5.8. Experimental results of the workload balancing algorithm

To illustrate the approach described in Section 5.7 we present the results obtained for different types of simulation (chemistry-disabled and enabled) of a reactor geometry with 10678 cells on the St. Petersburg Grid site. This site is heterogeneous in both the CPU power and the network connections of the processors (Type IV). There are two 1.8 GHz nodes (nwo1.csa.ru, nwo2.csa.ru) and two dual 450 MHz nodes (crow2.csa.ru, crow3.csa.ru), all having 512 MB RAM. One of the dual nodes (crow3.csa.ru) is placed in a separate network segment with 10 times lower bandwidth (10 Mbit/s against 100 Mbit/s in the main segment). The load balancing tests were performed with a moderate-size problem which does not pose restrictions on required memory, thus the memory influence parameter c_m was reduced to zero and the exploration was done for the application parameter f_c. The link bandwidth between the Master and Slave processors was estimated by measuring the time of MPI_Send transfers of a predefined data block (with the MPI buffer size equal

to 10^6 of MPI_DOUBLEs) during the solver execution, after the resources have been allocated. In these measurements the same logical network topology was used as employed in the solver. The CPU power and available memory were obtained by a function from the *perfsuite* library [22]. To validate the approach presented in the previous section we applied the workload balancing technique for a single simulation running on different sets of heterogeneous resources. The performance was estimated for different possible values of the parameter f_c (hence different weighting and workload distribution). For one simulation type we expect to obtain approximately the same value of the parameter f_c^* (that provides the best performance, see previous section) on different sets of resources. Fig. 5.11 illustrates the load-balancing speedup Θ achieved by applying the workload balancing technique for different values of the parameter f_c on several fixed sets of heterogeneous resources for a light-weighted (chemistry-disabled) simulation.

In Table 5.1 we summarize the combinations of processors dynamically allocated in 4 tests (different sets of resources) and the weights assigned to each processor for the values of f_c^* providing the best execution time, i.e. maximum balancing speedup (see Fig. 5.11, left).

Table 5.1 . Distribution of processors and balancing weights providing the best load balancing speedup for different sets of resources.

Sets of resources	**Weights assigned to each processor:**						**Heterogeneity metrics**		**Balancing speedup**
	nwo1 1.8GHz/ 100Mb/s	**crow2/1** 450MHz/ 100Mb/s	**crow3/1** 450MHz/ 10Mb/s	**crow2/2** 450MHz/ 100Mb/s	**nwo2** 1.8GHz/ 100Mb/s	**crow3/2** 450MHz/ 10Mb/s	φ_{proc}	φ_{net}	Θ
set I 3 processors	0.580	0.274	0.146	-	-	-	0.618	0.606	196 %
set II 4 processors	0.452	0.218	0.112	0.218	-	-	0.638	0.502	182 %
set III 5 processors	0.314	0.146	0.080	0.146	0.314	-	0.591	0.439	201 %
set IV 6 processors	0.278	0.160	0.062	0.160	0.278	0.062	0.618	0.606	207 %

Fig. 5.11 (left) shows that for a given simulation the best performance is delivered by weighting the resources with the value of $f_c \approx$ 0.3-0.4. Noticeably, this corresponds to the value obtained for this simulation during the preliminary analysis on homogeneous resources (compare to results for similar simulations in Section 5.5 Fig. 5.9). The results show that the algorithm gives an increase of balancing speedup Θ up to 207 percent compared to the initial non-balanced version of the code (with homogeneous workload distribution) on the tested resource sets. We can see that the distribution of the workload proportional only to the processor performance ($f_c = 0$) also gives a significant increase of the performance, but introduction of the dependency on application specific communication/computation ratio f_c and resource infrastructure parameters μ_i adds another 40 percent to the balancing speedup Θ.

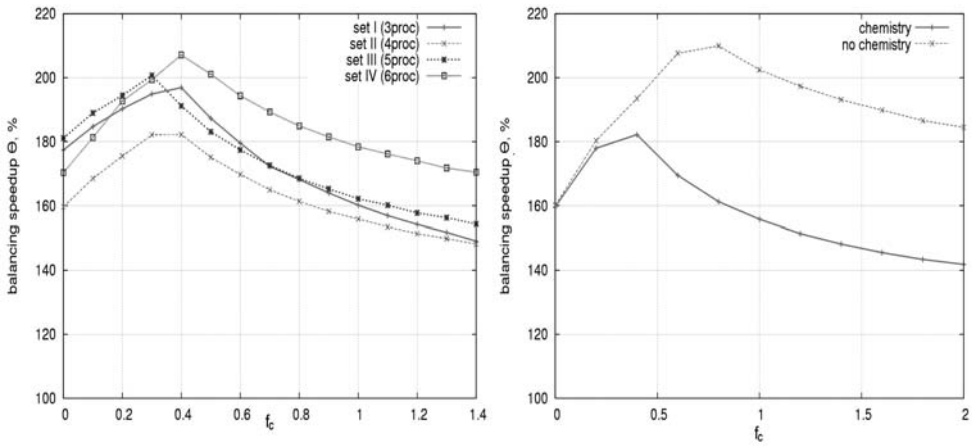

Fig. 5.11. Dependency of the balancing speedup Θ on the resource parameter f_c. Left: single simulation on different sets of resources. Right: different types of simulation on the same set of resources.

Fig. 5.11 (right) shows the dependency of the balancing speedup Θ for different types of simulation (chemistry enabled or disabled) on the same set of resources (set III from Table 5.1). The chemistry-disabled simulation has a higher communication/computation ratio (as was shown also in 5.5 Fig. 5.9). This is clearly seen in the experimental results where chemistry-disabled simulation obtains the highest balancing speedup Θ at higher f_c values. Moreover, the gain in the balancing speedup (maximal value of Θ) is higher for the simulation with a larger fraction of communications. These results illustrate that the introduced algorithm for resource adaptive workload balancing can bring a valuable increase in the performance for communication-intensive parallel programs running on heterogeneous resources.

5.9. Discussion and suggestions for generalized automated load balancing

The introduction of the load balancing technique allowed us to increase the efficiency of the parallel solver on heterogeneous resources. The proposed method of successive estimation of resource infrastructure parameters μ_i and further determination of the application specific f_c shows the possibility of automatic load balancing for applications with unknown internal structure (computations and communications).

Analysis of the results achieved with the workload balancing algorithm suggested that the following issues shall be addressed in order to optimize the balancing technique:

1. To measure the inter-process communication rate, we send a fixed amount of data from the Master to each Slave processor. However in some cases the response of the communication channels to the increasing amount of data is not linearly proportional as shown in Fig. 5.8. For the slower networks this tendency is even more pronounced.

This brings us to a conclusion that the amount of data sent to measure the links performance shall be close to the amount really transferred within the solver for every particular mesh size, geometry and solver type. Another option to estimate the inter-processor communication rate is to analyze the iteration data transfer time during the actual execution. However, this requires significant code modifications and might be undesirable.

2. To properly take into account the memory requirements of each particular instance of a parallel solver, similar reasoning shall be applied as for selecting c_p and c_n: the choice of the c_m coefficient, setting the significance (priority) of the memory factor influence on the application performance, must depend on the type of resources assigned, analogous to the c_p.
3. The specialty of the memory factor is that in addition to this resource-dependency it is strongly influenced by the application features. To take into account the memory requirements of a parallel solver, the weighting algorithm must be enriched by a function measuring the memory requirements per processor for each simulation on each set of resources. In case of sufficient memory on allocated processors, the load balancing can be performed taking into account all the factors (CPU, memory and network) where memory factor is a constraint. After this, another check of meeting the memory requirements on each processor must be performed. In the unfavorable case of insufficient memory on some of the processors, they must be disregarded from the parallel computation or replaced by other, better suited processors. This must be done preferably outside the application, on the level of parallel job scheduling and resource allocation. This brings us to the conclusion that ideally a combined technique shall be developed, where the application-centered load balancing approach is coupled with a system-level resource management.

5.10. Conclusions

In this chapter we address the issue of porting distributed problem solving environments to the Grid, using Virtual Reactor as an example of a complex application. One of the most challenging problems we encountered was porting parallel modules from homogeneous cluster environments to heterogeneous resources of the Grid, specifically the issue of keeping up a high parallel efficiency of the computational components. This problem arises for a wide class of parallel programs that employ homogeneous load distribution algorithms. To adapt these applications to heterogeneous Grid resources, we developed a theoretical approach and a generic workload balancing technique that takes into account specific parameters of the Grid resources dynamically assigned to a parallel job, as well as the application requirements. We validated the proposed algorithm by applying this technique to the Virtual Reactor parallel solvers running on the Russian-Dutch Grid testbed. It is worth noting that the load balancing speedup *does* go through a maximum at $f_c = f_c^*$ as shown in Fig. 5.11. This indicates that the load balancing strategy does find an optimum in the complex parameter space of the heterogeneous application/architecture combination. The clear maximum gives an unbiased guide towards automatic load balancing. The developed approach is well suited for either static or

dynamic load balancing, and can be combined with the Grid-level performance prediction models or application-level scheduling systems [23,24]. To further explore this new load balancing approach, we are currently working on the comparison of the theoretically derived optimization parameters for some specific topologies of parallel applications with those predicted by the heuristic algorithm presented in this chapter.

In order to optimize the resource management strategy of mapping the distributed components of the application problem solving environment, we benchmarked the individual components of the Virtual Reactor on a set of diverse Russian-Dutch Grid resources, and extensively studied the behavior of the parallel solvers with various problem types and input data on different resource infrastructures. The results clearly show that even within one solver different trends can exist in the application requirements and parallel efficiency depending on the problem type and computational parameters, therefore distinct resource management and optimization strategies shall be applied and automated procedures for load balancing are needed to successfully solve complex simulation problems on the Grid.

5.11. References

1. V.V. Korkhov; V.V. Krzhizhanovskaya and P.M.A. Sloot: A Grid Based Virtual Reactor: A case study of parallel performance and adaptive load balancing, Journal of Parallel and Distributed Computing, V. 68/5, May 2008, pp 596-608. http://dx.doi.org/10.1016/j.jpdc.2007.08.010
2. proj-openlab-datagrid-public.web.cern.ch, www.nbirn.net, www.fusiongrid.org, www.globus.org/alliance/projects.php, cmcs.ca.sandia.gov, www.us-vo.org
3. The CrossGrid EU Science project: http://www.eu-CrossGrid.org
4. High Performance Simulation on the Grid project: Dutch-Russian NWO-RFBR-047.016.007
5. V.V. Krzhizhanovskaya, P.M.A. Sloot, and Yu. E. Gorbachev. Grid-based Simulation of Industrial Thin-Film Production. Simulation: Transactions of the Society for Modeling and Simulation International, V. 81, No. 1, pp. 77-85, 2005
6. V.V. Krzhizhanovskaya, M.A. Zatevakhin, A.A. Ignatiev, Y.E. Gorbachev, W.J. Goedheer and P.M.A. Sloot. A 3D Virtual Reactor for Simulation of Silicon-Based Film Production. Proceedings of the ASME/JSME PVP Conference. ASME PVP-Vol. 491-2, pp. 59-68, PVP2004-3120 (2004)
7. A. Barak, G. Shai, and R. Wheeler The MOSIX Distributed Operating System, Load Balancing for UNIX, LNCS, V. 672, Springer-Verlag, 1993
8. K.A. Iskra, F. van der Linden, Z.W. Hendrikse, B.J. Overeinder, G.D. van Albada and P.M.A. Sloot: The implementation of Dynamite - an environment for migrating PVM tasks, Operating Systems Review, V. 34, N 3 pp. 40-55. Association for Computing Machinery, Special Interest Group on Operating Systems, July 2000.
9. G. Shao, R. Wolski and F. Berman. Master/Slave Computing on the Grid. Proceedings of Heterogeneous Computing Workshop, pp 3-16, IEEE Computer Society (2000)

10. S. Sinha, M. Parashar. Adaptive Runtime Partitioning of AMR Applications on Heterogeneous Clusters. In Proceedings of 3rd IEEE Intl. Conference on Cluster Computing, pp. 435-442, 2001
11. R. David et al. Source Code Transformations Strategies to Load-Balance Grid Applications. LNCS V. 2536, pp. 82-87, Springer-Verlag, 2002
12. C. Lu, S.-M. Lau. An Adaptive Load Balancing Algorithm for Heterogeneous Distributed Systems with Multiple Task Classes, International Conference on Distributed Computing Systems (ICDCS'96)
13. Z. Lan, V.E. Taylor, G. Bryan. Dynamic Load Balancing of SAMR Applications on Distributed Systems, Proc. of the ACM/IEEE conference on Supercomputing, 2001
14. Y. Zhang, K. Hakozaki, H. Kameda, K. Shimizu. A performance comparison of adaptive and static load balancing in heterogeneous distributed systems, Proc. of the 28th Annual Simulation Symposium, p. 332, 1995.
15. Fox, G.C., Williams, R.D., Messina, P., 1994, Parallel Computing Works!, Morgan Kaufmann Publishers, Inc.
16. V.V. Krzhizhanovskaya, M.A. Zatevakhin, A.A. Ignatiev, Y.E. Gorbachev, P.M.A. Sloot. Distributed Simulation of Silicon-Based Film Growth. Proceedings of the Fourth International Conference on Parallel Processing and Applied Mathematics (PPAM 2001), Lecture Notes in Computer Science, V. 2328, pp. 879-888. Springer-Verlag 2002
17. Nimrod-G: http://www.csse.monash.edu.au/~davida/nimrod/
18. G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D.Walker. Solving Problems on Concurrent Processors, volume 1, Prentice-Hall, 1988.
19. J.F. de Ronde; A. Schoneveld and P.M.A. Sloot: Load Balancing by Redundant Decomposition and Mapping, Future Generation Computer Systems, vol. 12, nr 5 pp. 391-407, April 1997
20. J.F. de Ronde. Mapping in High performance Computing. A case study on Finite Element Simulation, PhD thesis, University of Amsterdam, 1998
21. J.D. Teresco et al. Resource-Aware Scientific Computation on a Heterogeneous Cluster. Computing in Science & Engineering, V. 7, N 2, pp. 40-50, 2005
22. R. Kufrin. PerfSuite: An Accessible, Open Source Performance Analysis Environment for Linux. 6th International Conference on Linux Clusters. Chapel Hill, NC. (2005)
23. F. Berman, R. Wolski, H. Casanova, W. Cirne H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, N. Spring, A. Su, D. Zagorodnov. Adaptive Computing on the Grid Using AppLeS. IEEE Trans. on Parallel and Distributed Systems, vol. 14, no. 4(2003) 369—382
24. X.-H. Sun, M. Wu. Grid Harvest Service A System for Long-Term, Application-Level Task Scheduling. Proc. of 2003 IEEE International Parallel and Distributed Processing Symposium (IPDPS 2003)(2003)