Source Code Transformations Strategies to Load-balance Grid Applications*

Romaric David, Stéphane Genaud, Arnaud Giersch, Benjamin Schwarz, and Éric Violard

LSIIT-ICPS, Université Louis Pasteur, Bd S. Brant, F-67400 Illkirch (France)
Tel: +33 3 90 24 45 42 / Fax: +33 3 90 24 45 47
{david,genaud,giersch,schwarz,violard}@icps.u-strasbg.fr
http://grid.u-strasbg.fr/

Abstract. We present load-balancing strategies to improve the performances of parallel MPI applications running in a Grid environment. We analyze the data distribution constraints found in two scientific codes and propose adapted code transformations to load-balance computations. Experimental results confirm that such source code transformations can improve Grid application performances.

1 Introduction

With the growing popularity of middleware dedicated at making so-called *Grids* of processing and storage resources, network based computing will soon offer to users a dramatic increase in the available aggregate processing power. However, parallel applications have traditionally been designed for parallel computers and their executions on a Grid show poor performances. There are two major reasons for the lack of performance: first, the processors available are heterogeneous and hence the work assigned to processors is often unbalanced, and secondly the network links are orders of magnitude slower on the Grid than in a parallel computer.

Much work has been carried out to take into account such heterogeneous environments for distributed computing. However, few research works concern load-balancing strategies specifically guided by the application to be run. The well-known AppLeS project [1] uses information drawn from a specific application to schedule the execution of the application processes, but they do not modify the application source code to improve its execution. Thus, our project is original in the sense that we study how *source code transformations* may impact on the performances obtained when running applications on Grids, and eventually systematically operate these transformations so as to produce programs permanently adapted to heterogeneous environments. To validate our ideas, we work on some real scientific MPI applications.

* This work is supported by the French Ministry of Research through the ACI-GRID program.

M. Parashar (Ed.): GRID 2002, LNCS 2536, pp. 82–87, 2002.

This paper presents preliminary results concerning load-balancing techniques we designed to improve performances of two scientific applications. The two codes differ by their constraints on data distribution. The first one is unconstrained, it is possible to send any chunk of data to any process. The second one contains data dependencies, which implies a constrained data distribution. The next two sections describe each of the test applications. We discuss their constraints and expose the code transformation we applied to load-balance computations. We finish with some experimental results. The last section comments on the code transformations operated and discusses future work.

2 Load-Balancing for Unconstrained Data Distribution

2.1 Motivating Example: A Geophysical Application

We consider for our first experiment a geophysical code in the seismic tomography field. The parallelization of the application, presented in [2], assumes that all available processors are homogeneous (the implicit target for execution is a parallel computer). Consequently, an `MPI_Scatter` instruction was used, in which the root process distributes equal shares of data to each process.

We examine two methods of load-balancing this kind of application. The first one, a classical master/slave scheme, is *dynamic*. Slave processes ask a master process for a small chunk of the whole data set to compute. After computing it, the slaves ask for a new chunk and repeat this scheme until the master sends a termination message. The second one is *static*: the root process distributes the whole data set in a single communication round, except that it sends unequal shares of data whose sizes are statically computed on the basis of the processors and network performances.

We have implemented and tested both program transformations, but we focus in this study on the conditions the application must meet to implement the second load-balancing technique, and which performance results can be obtained with it.

2.2 Static Load-Balancing of Computations and Communications

The static load-balancing technique applies for SPMD programs made of rounds of simultaneous communications between all processes, followed by local computations ended by a global synchronization (which may be another communication round). Moreover, we must state further assumptions the programmer must verify before the load-balancing may take place. First, the data items in the input data set are *independent*, and secondly the number of data-items is *the only factor in time complexity.* Hence, giving any equal-size part of the domain to a given process will result in the same computation time.

Our objective is to minimize the elapsed time between two synchronization points. We consider the full overlap, single-port model of [3] in which a processor node can simultaneously receive data, send data to at most one destination, and perform some computation. As in this model the root process sends data to processes in turn, a receiving process actually begins its communication after all previous processes have been served. The root process starts its computation after all the processes have received their data. This leads to what we call a *"stair effect"*.

We explain in [4] how to compute, given the processors and network links speeds, the number of data items that should be allocated to each processor, so as to reach a load-balanced execution. Once this static load-balance computed, the `MPI_Scatter` instruction of the original program is replaced with an `MPI_Scatterv`.

2.3 Experimental Results

Our experiment consists in the computation of 827,000 data units on 16 processors. Processors are heterogeneous and located at two geographically distant sites. All machines run Globus with MPICH-G2 [5]. We made a series of benchmarks to measure processor and network performances.

The first experiment (fig. 1(a)) evaluates performances of the original program in which each process receives an equal amount of data. Non-surprisingly the processes end times largely differ, thus exhibiting an important imbalance. The second experiment (fig. 1(b)) shows the master/slave version behavior which appears well-balanced after we have finely tuned the size of data chunk sent to slaves[1].

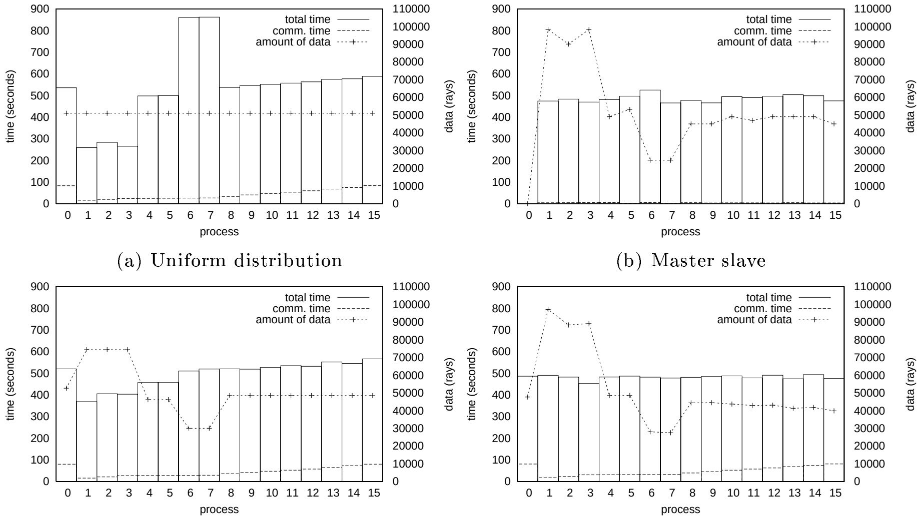
Next, we experiment the load-balance of the scatter operation. To assess important parameters, we have first tried to load-balance using the relative processors speed ratings only (fig. 1(c)). Omitting the network parameters leaves important imbalances and gives unsatisfactory results as compared to the master/slave implementation. The last experiment (fig. 1(d)) computes the load-balance using all parameters. We obtain here the best balance, which confirms the importance of all parameters, especially to take into account the "stair effect" that clearly appears on figures 1(a,c,d). In the master/slave implementation the total communication time is short as compared to the computation time. We conclude that in the scatter implementations only a small part of the measured total time is spent in true communications of data.

3 Load-Balancing with Constrained Data Distribution

3.1 Motivating Example: An Application in Plasma Physics

Our example is an application devoted to the numerical simulation of problems in Plasma Physics and particle beams propagation. The application implements the PFC resolution method discretizing the Vlasov equation. The parallel code [6] works with any number of processors and uses a classical data decomposition

[1] Note that there are only 15 computation processes in this implementation since the master process only handles data distribution.



(a) Uniform distribution (b) Master slave

(c) Load-balancing (without network) (d) Load-balancing (with network)

Fig. 1. Experimental results (the root process is on processor 0)

technique. Matrices are split into blocks of equal size distributed by row on the processors. The code consists in a succession of computation-communication phases where the communication steps are global block matrix transpositions.

3.2 Load-Balancing through Process Emulation

In such applications where data structure must be preserved, but computations can still be spread over any number of processes, a naive solution to achieve load balancing is to make the operating system run more than one process per processor on fastest processors. Such a procedure implies a lot of system overhead, due to inter-process communications and time-sharing mechanisms. An idea to avoid this, is to rewrite the code in such a way that only one real process will compute all data given to the processes we intended to run on the given processor. In other words we *emulate* several processes by a single one.

The original program must meet the following conditions : *(i) the code works with any number of processors*, *(ii) the workload is evenly spread over the processors* and *(iii)* the workload *only depends on the amount of data* to be processed.

The emulation of several processes by one process consists in serializing the computations of the computation phases and changing the communication scheme. The MPI calls have to be modified in order to map emulated processes onto real processes and perform memory copy instead of communications when sender and receiver are on the same processor.

Moreover, blocking point-to-point communications such as `MPI_Send` have to be replaced by their non-blocking versions (e.g. `MPI_Isend`) in order to avoid deadlock situations. Calls to a collective communication have to be performed only once on a processor, no matter the number of emulated processes it holds. Some collective communications such as `MPI_Barrier` do not require any other changes whereas some others do. For example `MPI_Scatter` instances should be replaced by `MPI_Scatterv` to distribute chunks of data proportionally to the number of emulated processes on each processor. A more detailed description of the transformation process can be found in [4].

3.3 Experimental Results

For our experiments, we use our code in Plasma Physics and a subset of our test Grid made of four heterogeneous processors. We approximate the number of emulated process from speed ratings obtained from benchmarks[2]. The results show significant gains in performance.

In order to validate our code transformation, we measured the wall clock time of: (a) the initial application with one process per processor, (b) the initial application with a basic load balancing using the system to run several processes on a single processor and (c) the transformed application with a load-balancing using emulated processes. For (b) and (c), we distributed 11 processes on the four processors. Experiments were conducted for 32^4, 48^4, and 64^4 data points. Results are reported in table 1. The modified algorithm is always better than the original one, may there be or not system load-balancing.

Table 1. Elapsed time (seconds)

Size	(a) No load balancing	(b) System load balancing	(c) Emulated processes
32^4	342	525	301
48^4	2005	2223	1874
64^4	5380	4752	4404

4 Related Work

The study carried out in [8] compares dynamic versus static load-balancing strategies in an image rendering ray-tracing application. They conclude that no one scheduling strategy is best for all load conditions, and recommend to investigate further the possibilities of switching from static to dynamic load-balancing. Our experiments confirm that static load-balancing requires precise information about parameters to be efficient whereas the master/slave model naturally adapts to heterogeneous conditions. Therefore, the variance of conditions could be taken into account to request static or dynamic load-balance during execution. Providing a dedicated library to implement load-balancing is also

[2] Work is under progress to use theoretical results from [7] so as to find the optimal number of emulated processes.

proposed by George [9] with the `DParLib`, who addresses the problem of dynamic load-balancing via array distribution for the class of iterative algorithms with a SPMD implementation. The statistics about load-balance produced dynamically during the execution can be used to redistribute arrays from one iteration to the other. However, the library does not take into account possible communication-computation overlaps that we need in our second test application.

5 Conclusion

It appears from this study, that the source code must be finely analyzed to choose which load-balancing solution matches best the problem. In the first example, we have put forward influent parameters for static load-balancing. In the second example we have shown that a possible transformation strategy to overcome the constraints on data-distribution and keep a good communication-computation overlap can be the process emulation technique.

Future work should be done in several directions. We first need to further investigate the communication schemes used in real applications and how they perform in a Grid environment. A classification of the communication types may be used by a software tool to select appropriate transformation strategies. We also believe the programmer should interact with the tool to guide program transformations as he can bring useful information about the application.

References

[1] Berman, F., Wolski, R., Figueira, S., Schopf, J., Shao, G.: Application-level scheduling on distributed heterogeneous networks. In: Proceedings of SuperComputing '96. (1996)

[2] Grunberg, M., Genaud, S., Mongenet, C.: Parallel seismic ray-tracing in a global earth mesh. In: Proceedings of PDPTA'02. (2002) 1151–1157

[3] Beaumont, O., Carter, L., Ferrante, J., Legrand, A., Robert, Y.: Bandwidth-centric allocation of independent tasks on heterogeneous platforms. Technical Report 4210, INRIA, Rhône-Alpes (2001)

[4] David, R., Genaud, S., Giersch, A., Schwarz, B., Violard, E.: Source code transformations strategies to load-balance grid applications. Technical Report 02-09, ICPS-LSIIT, University Louis Pasteur, Pôle API, Bd. S. Brant,F-67400 Illkirch (2002)

[5] Foster, I., Karonis, N.: A grid-enabled MPI: Message passing in heterogeneous distributed computing systems. Supercomputing (1998)

[6] Violard, E., Filbet, F.: Parallelization of a Vlasov solver by communication overlapping. In: Proceedings of PDPTA'02. (2002)

[7] Boulet, P., Dongarra, J., Robert, Y., Vivien, F.: Static tiling for heterogeneous computing platforms. Parallel Computing **25** (1999) 547–568

[8] Shao, G., Wolski, R., Berman, F.: Performance effects of scheduling strategies for master/slave distributed applications. Technical Report CS98-598, UCSD CSE Dept., University of California, San Diego (1998)

[9] George, W.: Dynamic load-balancing for data-parallel MPI programs. In: Message Passing Interface developers and users conference. (1999) 95–100