Dynamic Load Balancing for Ocean Circulation Model with Adaptive Meshing

Eric Blayo, Laurent Debreu, Grégory Mounié, and Denis Trystram

LMC-IMAG, BP 53, 38041 Grenoble Cedex 9, France
tel: (33) (0)476514500, fax: (33) (0)476631263
{name}@imag.fr

Abstract. This paper reports the parallel implementation of adaptive mesh refinement within finite difference ocean circulation models. The implementation is based on the model of *Malleable Tasks* with inefficiency factor which allows a simple expression of the different levels of parallelism with a good efficiency. Our goal within this work was to validate this approach on an actual application. For that, we have implemented a load-balancing strategy based on the well-known level-by-level mapping. Preliminary experiments are discussed at the end of the paper.

Keywords: Load Balancing - Malleable Tasks - Adaptive Mesh Refinement - Ocean Modeling

1 Introduction

Numerical modeling of the ocean circulation started in the sixties and was continuously developed since that time. Today the motivation for ocean modeling is mainly twofold: The first goal is climate study (prediction of the evolution of the climate, at ranges from a few months to a few years), while the second motivation is operational oceanography, i.e. near real time forecast of the "oceanic weather", in a way similar to operational meteorology.

A major practical problem within ocean general circulation models (OGCMs) is their very large computational cost. These models are run on vector and/or parallel supercomputers (Cray C90 or T3E, Fujitsu VPP . . .), where a usual simulation requires several hundred or thousand hours of CPU-time.

In this context, it is clear that adaptive meshing could be of real interest for ocean modelers. It could reduce the computational cost of models by taking advantage of the spatial heterogeneity of oceanic flows and thus using a fine mesh only where and when necessary. Moreover, this would allow local zooms on limited areas of particular interest, e.g. for operational purposes. The finite element technique permits of course the use of such a non-uniform adaptive grid over the computational domain. However, this approach is mostly reserved, in the field of ocean modeling, to coastal or tidal models, and all major OGCMs use finite difference methods on structured grids. Starting from this consideration, Blayo and Debreu [2] are currently developing a Fortran 90 software package, which

P. Amestoy et al. (Eds.): Euro-Par'99, LNCS 1685, pp. 303–312, 1999.

will furnish any finite difference ocean model the capability of adaptive meshing and local zooming. The method is based on the adaptive mesh refinement (AMR) algorithm proposed by Berger and Oliger [1], which features a hierarchy of grids at different resolutions, with dynamic interactions. Since the mesh refinement is adaptive over the time step, the number of grids as well as their sizes and resolutions vary during the simulation. Thus, the computational load is also varying in time, and a dynamic load balancing is necessary to implement efficiently this AMR method on a parallel computer.

This paper will discuss the design, implementation and performance of a load balancing package and its integration into a code for large scale simulations of ocean circulation. It is organized as follows: in section 2, we describe some features of ocean models useful for the rest of the paper and present the AMR method. The model of Malleable Tasks with inefficiency factor is presented in section 3. Then, an adaptation of the level-by-level load balancing algorithm is detailed and analyzed in section 4. Some numerical experiments are reported in section 5, in order to illustrate the behavior of the algorithm on a parallel machine. Finally, some conclusions and perspectives are discussed.

2 Ocean Models and Adaptive Mesh Refinement

2.1 About Ocean Models

The basic relations describing the ocean circulation are equations for conservation of momentum (Navier-Stokes equations), mass (continuity equation), heat and salt, and an equation of state. The addition of the Boussinesq's approximation (variations of density are small) and the hydrostatic approximation leads to the so-called "primitive equations" (PE), which are solved by most OGCMs, using finite difference techniques. Simpler models of ocean circulation can also be derived by making additional assumptions.

Thus, an ocean model can be written in a symbolic way as: $\dfrac{\partial X}{\partial t} = F(X)$ where t is the time, X is the state variable and F is a non-linear operator. The time discretization scheme is generally explicit, which leads in most cases to discretized equations of the form $X(t+\delta t) = G(X(t), X(t-\delta t))$ where δt is the discretization time step. However, some models use also an implicit scheme for one of the equations (equation for barotropic – or depth-averaged – motion), which implies in that case to solve at each time step a linear system for $X(t+\delta t)$.

The parallelization of ocean models is performed by domain decomposition techniques : the geographical domain is divided into subdomains, each of them being affected to a processor (e.g. [11] [3] [12]). An important point for the purpose of this work is to emphasize that ocean models are *regular* applications, in the sense that the volume of computations can be estimated quite precisely as a function of the grid size and the number of processors (at least, it can be measured by adequate benchmarks).

2.2 Adaptive Mesh Refinement (AMR)

The basic principle of AMR methods consists in locally refining or coarsening the computation mesh, according to some mathematical or physical criteria (like error estimates or eddy activity). Such techniques are widely used with finite element codes, but rather rarely with finite differences because refinements lead to non-homogeneous grids and thus complicate the handling of the code. However, Berger and Oliger [1] proposed an AMR algorithm which avoids this drawback, by considering a locally multigrid approach. In their method, the refinement is not performed on a unique non-homogeneous grid, but on a hierarchy of grids, i.e. a set of homogeneous embedded grids of increasing resolutions, and interacting among themselves. Without going deeply into more details (the reader can refer for example to [1] or [2]), the principle of the algorithm is as follows. Consider a hierarchy of grids, like in Figure 1: it consists in a root (or level-0) grid covering the entire domain of computation with coarse resolution Δh_0 and coarse time step Δt_0, and a number of subgrids (or level-1 grids) with a finer resolution $\Delta h_1 = \Delta h_0/r$ and a finer time step $\Delta t_1 = \Delta t_0/r$ focused only on some subdomains (r is an integer called the refinement ratio). This structure is recursive, in the sense that any level-l grid can contain finer level-$(l+1)$ subgrids, with $\Delta h_{l+1} = \Delta h_l/r$ and $\Delta t_{l+1} = \Delta t_l/r$ (of course until a maximum level $l_{\max}$).

Time integration is performed recursively starting from the root grid. Any level-l grid is advanced forward one time step Δt_l. The solutions at time t and $t + \Delta t_l$ are used to provide initial and boundary conditions for the integration of the level-$(l + 1)$ subgrids. These subgrids can then be advanced forward r time steps Δt_{l+1}, to provide a more precise solution at time $t + r\Delta t_{l+1} = t + \Delta t_l$ on the regions covered by the subgrids. These solutions at level $(l+1)$ are then used to improve the solution at level l, via an update procedure.

The relevance of the grid hierarchy is checked regularly every N coarse time steps Δt_0. A criterion is evaluated at every grid point to determine whether the local accuracy of the solution seems sufficient or not. Subgrids can then be created, resized or removed.

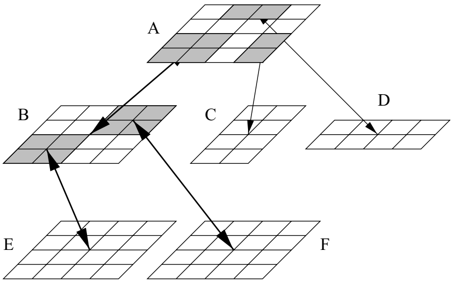

Fig. 1. Example of grid hierarchy

A major interest of this AMR method is that it can be used without modifying the model. The model can be seen like a black-box (corresponding to the Step routine in the previous algorithm) needing only configuration parameters (domain of computation, mesh size Δh, time step Δt, initial conditions, boundary conditions) to be run. This is one of the key ideas of the package that we are presently developing.

In the parallel version of our package, the model will still be used as a black-box like in the sequential case, but with an additional parameter: the number of processors used to run the model. Since the different grids at a given level l can be run simultaneously, they will be affected to different groups of processors. The problem is then to determine, for a given hierarchy, which is the best grids-to-processors correspondence for computational efficiency. These assignments must be determined at every regridding step, because of the evolution of the grid hierarchy during the simulation. The model of Malleable Tasks presented in the next section is an efficient tool for solving such problems.

3 Parallel Implementation

3.1 Malleable Tasks

Since the eighties, many works have been developed for parallelizing actual large scale applications like the oceanographic simulation considered in this paper. The solution of the scheduling problem (in its larger acception) is a central question for designing high-performance parallelization. It corresponds to finding a date and a processor location for the execution of each task of the parallel program. Among the various possible approaches, the most commonly used is to consider the tasks of the program at the finest level of granularity and apply some adequate clustering heuristics for reducing the relative communication overhead [6]. The main drawback of such an approach is that communications are taken into account explicitly (they are expressed assuming a model of the underlying architecture of the system which is very difficult to establish and often far from the actual asynchronous and non-deterministic behavior of the parallel execution). It is well-known that the introduction of explicit communications into the scheduling algorithms renders the problem harder than without communication [4]. Recently, a new computational model has been proposed [5]. *Malleable tasks* (denoted MT in short) are computational units which may be themselves executed in parallel. It allows to take into account implicitly the influence of communications.

3.2 Motivation for Using MT

There are several reasons for introducing MT as a way to design efficient parallel applications. Let us summarize them below:

- MT allow to hide the complex behavior of the execution of a subset of tasks by using a parameter (the inefficiency factor μ discussed later) to abstract

the overhead due to the management of the parallelism (communications, idle times due to internal precedence constraints, etc..).
- MT reflect the structure of some actual parallel applications because they unify the usual single processor tasks and tasks which may require more than one processor for their execution. The structural hierarchy is then included into the model and the user does not specify when to use one model or another.
- MT simplify the expression of the problems in the sense that the user does not have to consider explicitly the communications. Thus, the same code may be used on several parallel platforms (only the μ factors will change), leading to a better portability.

The idea behind this model is to introduce a parameter which will implicitly give the communication overhead. An average behavior has to be determined using the *inefficiency* factor that will be now discussed in more details.

3.3 Inefficiency Factor: Definition and Properties

The efficiency of the execution of a parallel application depends of many factors like the algorithm, the data size, the data partitioning between available processing units, communication volume, network latency, topology, throughput, etc.. Ideally, a parallel system with m processors could complete the execution of an application m times faster than a single processor. However, an actual m processors system does not achieve such speedup due to some overhead introduced by inter-processor communication (which is slow compared to basic processor computation) and synchronization between the tasks. The ratio between achieved speedup and theoretical speedup depends on the number m of processors and the size N of the application. It will be denoted by $\mu(m, N)$ and called *inefficiency factor*. We give below the definition using a geometric interpretation.

Definition 1. *The inefficiency factor is the expansion of the computational area while using m processors: $t_m = \mu(m, N)\frac{t_1}{m}$ where t_1 (resp. t_m) is the time required to complete a malleable task of size N on one processor (resp. on m processors).*

Generally, the inefficiency factor increases with the number of processors and decreases with the size of the parallel task (at least until a certain threshold). The general shape of μ function may be divided into three zones of consecutive intervals of number of processors: (I) corresponds to the start-up overhead due to the management of the parallelism, where the efficiency may be quite bad. (II) is the region where the addition of extra processors cost almost nothing, speed-ups here are generally linear. (III) corresponds to a degradation due to the lack of parallelism (for instance, for too small MT). Some properties of μ are presented below. They will be useful for bounding the time in the load-balancing heuristics.

Property 1. $\mu(q,.)$ is an increasing function of q

Proof. Adding a processor generally involves an extra cost for managing communications and synchronizations between this processor and the others. This comes directly from the Brent's Lemma [9]. □

Property 2. Given a specific m processor system and a particular application, $\mu(1,N) = 1$ and $\mu(q+1,N)\dfrac{t_1}{q+1} < \mu(q,N)\dfrac{t_1}{q}$ $\forall q, 1 \leq q \leq m-1$.

Proof. These relations are usual hypotheses in parallel processing. The first part comes from the sequential execution of a task. For the second relation, just remark that it is useless to solve the application with one more processor if t_{q+1} is greater than t_q. □

A direct corollary of this last property ensures that $\dfrac{\mu(q,.)}{q}$ is a decreasing function of q.

4 Load-Balancing Issues

4.1 Definition and Context

A parallel application is usually described as a set of *tasks* (which are computational units) plus their interactions which are represented as a graph, called the *precedence task graph* [9]. For the specific oceanographic application we are considering in this paper, the tasks correspond to solve the equations on a mesh at a given time step. The parameters of a task are the size of the mesh (n_1 and n_2 plus the height along the vertical axis, at each time step k). This graph is not known in advance, it evolves in time and a mesh refinement leads to introduce a new series of tasks in the graph. In this sense, the problem is dynamic.

According to most studies in parallel processing, we are interested in minimizing the total execution time. It is influenced by two contradictory criteria, namely the idle time (or load-balance) which decreases as the size of the grain decreases and the communication overhead which grows with the number of processors. The load-balancing problem corresponds formally to determine an application which associates to each (malleable) task a sub-set of the processors. The resulting scheduling problem (defined as determining the date of which each task will start its execution, synchronously on the sub-set of processors) should add some constraints; namely, a task is allocated only once and should not be preempted. One contribution of this work is to introduce the influence of the inefficiency factor into load-balancing policies.

The well-known *gang scheduling* policy corresponds to allocate the total resources of the parallel system for executing each task [7]. Gang will be used as a basis for comparison.

4.2 Level-by-Level Mapping

The well-known level-by-level load-balancing policy can also be adapted to MT with inefficiency. A complete presentation of the basic heuristic can be found in [8]. It was proposed for a classical model of tasks with no communication delays. The basic idea of the level-by-level mapping is to distribute the tasks level by level in the precedence task graph. The level i is defined as the sub-set of independent tasks (not related by the precedence constraints) at distance i from the root of the corresponding not-weighted task graph. In other words, this policy may be interpreted as a series of successive synchronized Macro-Gang schedulings with sets of independent malleable tasks.

5 Experiments

5.1 The Ocean Model

The ocean model used for this study is a simple quasi-geostrophic box model, since our intention is to validate the MT approach and to design a good scheduling heuristic before running the simulation on the operational models. This type of model has been widely used in the ocean modeling community, and is known as a simple prototype of eddy-active large scale circulation in the mid-latitudes. Blayo and Debreu [2] already implemented the AMR method in such a multi-layered model, and demonstrated its interest.

In the present study, we use a barotropic (i.e. one layer) version of this model (see for instance [10] for a detailed description of such a model). Its governing equation can be written as:

$$\frac{\partial \Delta\psi}{\partial t} + J(\psi, \Delta\psi) + \beta\frac{\partial\psi}{\partial x} = \mathrm{curl}\tau - r\Delta\psi + A\Delta^2\psi \qquad (1)$$

where ψ is the streamfunction, J is the Jacobian operator, β is the meridional gradient of the Coriolis parameter at the mid-latitude of the domain, τ is the horizontal wind stress at the surface of the ocean, r is a bottom friction coefficient, and A is a lateral viscosity coefficient.

5.2 Evaluation of Inefficiency

The inefficiency factors μ correspond to the penalty coefficients of the parallel time due to the overhead coming from the management of the parallelism. The parallel code was benchmarked in order to determine empirically an expression of the inefficiency factors for including them into the load-balancing policies. The μ is measured using single grid parallel computation. The variation of these curves in regard to the number of processors confirms the same global behaviour. Two aspects were studied:

- The variation of μ versus the grid size (cf Fig. 2) for a given number of proc. Note that for other number of processors, the general shape of the curves remains the same. The greater the number of processors, the greater the number of anomalies.

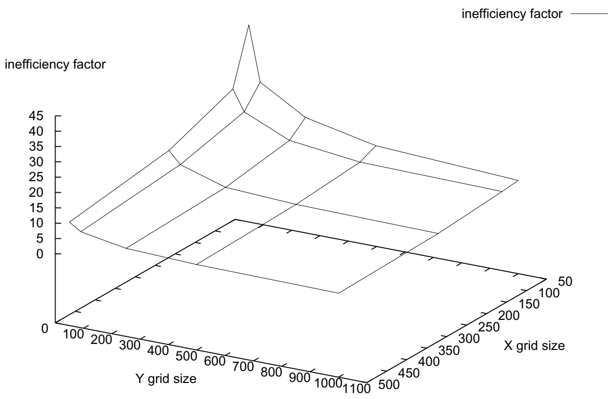

Fig. 2. Inefficiency factor as a function of the number of gridpoints in $x-$ and $y-$directions for 15 processors

- Figure 3 depicts the typical behaviour of μ versus the number of processor for a small and a large grid. This behaviour shows clearly some limitations of speedup due to inefficiency for small grids (speed down) and a linear speedup of larger problems.

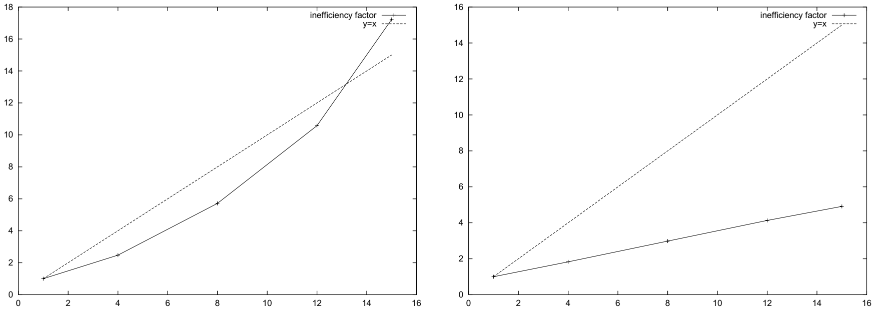

Fig. 3. Inefficiency factor as a function of the number of processors. Left panel: with anomaly (small grid size); right panel: without anomaly (large grid size). The reference curve $y = x$ is also plotted.

5.3 Experimental Results

We are currently developing numerical experiments on an IBM-SP2 parallel machine with 16 nodes connected on a fast switch.

Since the oceanographic model considered here is rather simple, the present study should be interpreted as a preliminary study for comparing the load-balancing policies and demonstrating the interest of using MT model for actual applications.

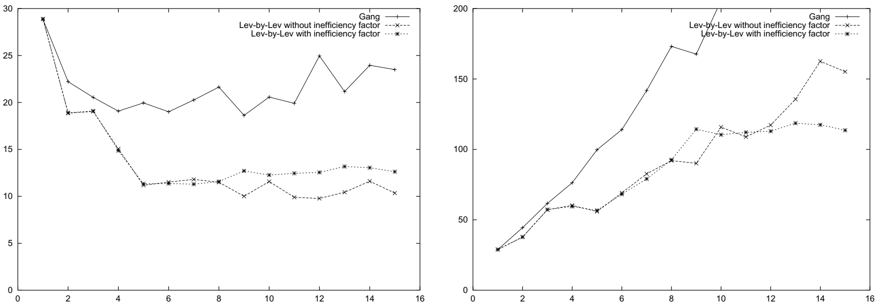

Fig. 4. Time and Work achieved by Gang, Level-by-Level with and without inefficiency factor scheduling algorithms versus the number of processors.

The tests reported in Fig. 4 compare speedup and efficiency of the three load-balancing policies: gang, level-by-level with and without inefficiency factor for multiple runs of a test case.

On a small number of processors the difference between algorithms is small. In fact, as the load imbalance produces large overhad, the gang scheduling perform better than Level by Level algorithms in some cases.

On a greater number of processor, the gang scheduling induces a large communication overhead.

The two Level by Level algorithms (with and without inefficiency factor) seem quite close in appearance on left panel of Fig. 4, namely in achieved makespan. However the Level by Level algorithm with inefficiency factor never used more than 9 processors while it achieves almost the same time. The amount of computation reported in Fig. 4 show clearly that inefficiency factor allows to bound it without large performance penalty.

The good behavior of the Level-by-Level scheduling algorithm with inefficiency factor allows us to expect good results for future works using a large number of processors, for example on a 128 processors Cray T3E.

Note, since the adaptive algorithm use the oceanographic model as black boxes, it is intrinsically synchronous; communications between MT only occur before and after computation. Thus the makespan is quite sensitive to the synchronization overhead between MT. This is especially true for the Gang scheduling and partially explains its poor efficiency. Gang scheduling is also highly sensitive to perturbations occurring in multi-user environment.

6 Conclusion and Future Works

We have presented in this work a new way to parallelize actual applications. It is based on the model of Malleable Tasks with inefficiency factor. We have discussed some fundamental properties of MT and showed how they are suitable for designing load-balancing heuristics. Some adaptations of well-known heuristics have been done, and preliminary experiments on a test problem coming from oceanography are currently carried out to show the feasibility and potential of MT model.

Other short term perspectives are to develop other load-balancing heuristics, and implement them within operational models of ocean circulation.

References

[1] Berger M. and J. Oliger, 1984: Adaptive mesh refinement for hyperbolic partial differential equations. *J. Comp. Phys.*, **53**, 484-512.

[2] Blayo E. and L. Debreu, 1998: Adaptive mesh refinement for finite difference ocean models: first experiments. *J. Phys. Oceanogr.*, **29**, 1239-1250.

[3] Bleck R., S. Dean, M. O'Keefe and A. Sawdey, 1995: A comparison of data-parallel and message passing versions of the Miami Isopycnic Coordinate Ocean Model (MICOM). *Paral. Comp.*, **21**, 1695-1720.

[4] Hoogeveen J., Lenstra J.-K., and Veltman B., 1994: Three, four, five, six, or the complexity of scheduling with communication delays. *Operations Research Letters*, **16**, 129–137

[5] Turek J., Wolf J. and Yu, P., 1992: Approximate algorithms for scheduling parallelizable tasks. *4th Annual ACM Symposium on Parallel Algorithms and Architectures*, 323-332

[6] Gerasoulis A. and Yang T., 1994: DSC : Scheduling Parallel Tasks on an Unbounded Number of Processors. *IEEE Transaction on Parallel and Distributed Systems*, **5**,951-967

[7] Scherson I.D., Subramanian S.D., Reis V. L. M. and Campos L. M., 1996: Scheduling computationally intensive data parallel programs. *Placement dynamique et répartition de charge : application aux systèmes parallèles et répartis (École Française de Parallélisme, Réseaux et Système)*, Inria, 107-129

[8] Kumar V., Grama A., Gupta A. and Karypis G., 1994: Introduction to Parallel Computing: Design and Analysis of Algorithms, *Benjamin/Cummings*

[9] Cosnard M. and Trystram D., 1993: Algorithmes et architectures parallèles. InterEditions, collection IIA.

[10] Pedlosky J., 1987: Geophysical fluid dynamics. Springer-Verlag, 710.

[11] Smith R.D., J.K. Dukowicz and R.C. Malone, 1992: Parallel ocean general circulation modeling. *Physica D*, **60**, 38-61.

[12] Wallcraft A.J. and D.R. Moore, 1997: The NRL layered ocean model. *Paral. Comp.*, **23**, 2227-2242.