Atmosphere and Ocean Circulation Simulation on Massively Parallel Computers

Lex Wolters*
High Performance Computing Division,
Department of Computer Science, Leiden University,
P.O. Box 9512, 2300 RA Leiden, The Netherlands

Abstract

In this paper we present some results on the implementation of atmosphere and ocean circulation models on massively parallel computers. These circulation models are enormous computationally demanding tasks and play an indispensable role in climate modelling. Therefore they have an important impact on society.

After a brief introduction on the physics behind the circulation models based on geophysical fluid dynamics, the differences between the simulation of atmosphere circulation and ocean circulation are outlined. Next we discuss the numerical techniques applied in the models to solve the partial differential equations. Performance results of different implementations of a simple, theoretical model on parallel architectures are presented, as well as preliminary results for the implementation of a very computational intensive task in a numerical weather forecasting model. Finally, ideas are outlined on the development of a distributed library for the type of calculations in the models concerned.

1 General circulation models

Atmospheric and ocean circulation models describe the circulation of heat, air/water and chemicals in the atmosphere and world oceans or smaller basins. In coupled models also the exchange between oceans and the atmosphere is taken into account. These circulations have a major impact on climate and climate change. The strong relation to the behaviour of climate means that a computer model designed to predict climate changes, must include the ocean circulation and mixing, as well as the atmospheric circulation and physics of clouds, radiation and precipitation. The atmospheric component is relatively well advanced, because of the economical importance of numerical weather prediction, which started in the 1950s. The first numerical ocean models were developed in the 1970s, so this research area is relatively new, and as a result lays behind the atmospheric component. Ocean Circulation Modelling and Climate Modelling are considered as two of the 'Grand Challenges' in computational sciences, as is outlined in a report by the Executive Office of the President of the USA [1], and in more detail in a report by the Committee on Physical, Mathematical and Engineering Sciences of the National Science Foundation [2]. In Europe the importance of these research areas has been recognized by the Commission of the European Communities in [3].

*Support was provided by the Esprit EC Agency under Grant No. APPARC 6634 BRA III.

2 Description of the models

2.1 Primitive equations

This section is based on [4] and [5]. We present only a global overview of some theoretical aspects of circulation modelling, so the reader can get an idea about the kind of physics behind these models. For more details the reader is referred to [4, 5, 6].

The basis of the general circulation models is stated by the theory of geophysical fluid dynamics. The Navier-Stokes equations are the starting point to create a model. However, since one wants to concentrate on the essential physics and to make the models not too complicated, the Navier-Stokes equations are approximated in several ways. By applying the thin-shell approximation, the hydrostatic assumption, the Boussinesq approximation, and considering scale analysis, one derives the so-called primitive equations. The exact form of these equations differs depending on the applied approximations. For a global ocean model they are given by (see e.g., [5]):

two momentum equations:

$$\frac{du}{dt} - \frac{uv\tan\phi}{a} - fv = -\frac{1}{\rho a\cos\phi}\frac{\partial p}{\partial\lambda} + F_\lambda \ , \tag{1}$$

$$\frac{dv}{dt} + \frac{u^2\tan\phi}{a} + fu = -\frac{1}{\rho a}\frac{\partial p}{\partial\phi} + F_\phi \ , \tag{2}$$

the hydrostatic equation:

$$\frac{\partial p}{\partial z} = -\rho g \ , \tag{3}$$

the continuity equation:

$$\frac{1}{a\cos\phi}\left[\frac{\partial u}{\partial\lambda} + \frac{\partial(v\cos\phi)}{\partial\phi}\right] + \frac{\partial w}{\partial z} = 0 \ , \tag{4}$$

an equation of state:

$$\rho = \rho\left(T, p, \ldots\right) \ , \tag{5}$$

and a thermodynamic equation:

$$\frac{dT}{dt} = f\left(T, \ldots\right) \ , \tag{6}$$

with in all equations

$$\frac{d}{dt} \equiv \frac{\partial}{\partial t} + \frac{u}{a\cos\phi}\frac{\partial}{\partial\lambda} + \frac{v}{a}\frac{\partial}{\partial\phi} + w\frac{\partial}{\partial z} \ .$$

In these equations a is the earth radius, Ω the angular speed, λ the longitude, ϕ the latitude, z the height, $f = 2\Omega\sin\phi$ the Coriolis parameter, and F_λ, F_ϕ the friction forces. The unknowns variables are the velocity components u, v, w, the pressure p, the density ρ, and the temperature T. For atmospheric models usually a moisture equation is added, while for ocean models a salinity equation is common, which results in another unknown variable for each type of model: the moisture μ and the salinity S, respectively. If desired this set of equations can be extended with other equations and their corresponding unknowns. Together with proper boundary conditions and initial values the equations 1–6 form the minimal set used in physical oceanography or meteorology for global circulation models.

2.2 Solving partial differential equations

The primitive equations are a set of partial differential equations (PDEs). To solve PDEs numerically one can choose between three methods:

I. Finite difference methods (FDM). These methods are very important for ocean models, and are applied in atmospheric models too.

II. Spectral methods (SM). They are often implemented in global atmospheric models. The complex geometry of the world ocean prevents a selection of suitable basis functions for these methods. So they are less appropriated for ocean modelling.

III. Finite element methods (FEM). The big advantage of these methods is that the resolution can be increased at regions of interest. However, the implementation of these kind of models is time-consuming and non trivial. First attempts of formulating atmospheric and ocean models in terms of finite element methods appear gradually, and it is expected that these methods will become more important in the near future.

At the moment we mainly concentrate on FDM, because they are applied both in meteorology and in physical oceanography. In FDM space and time are divided in discrete space and time intervals (Δx and Δt, respectively). In this way one obtains a 3-dimensional grid in space, and then the method consists of updating the variables associated with the gridpoints, where the derivatives are approximated by finite differences.

2.3 Subgrid processes

The equations resulting from FDM describe a model for grid scale motion and larger. However, nature contains also processes evolving faster than the time step Δt or on a smaller scale than the space interval Δx. These processes are called subgrid processes and they are too important be neglected. One obvious way to remove this processes is to increase the resolution of the model. Unfortunately in general this is impossible for computational reasons. As a result one has to accept the so-called parameterization assumption, which assumes that the effect of such processes can be expressed entirely in terms of the large scale variables. Two additional problems related to this assumption are:

1. There exists no simple truncation at these scales. This means that in contrast with numerical analysis, where in the limit of $\Delta x \to 0$, convergence to the exact solution can be proven, this does not hold for the parameterization of subgrid processes. In the limit $\Delta x \to 0$ the parameterization can change the convergence rate or even prevents convergence.

2. The separation of scales is not clear. There exists no clear division between the processes that are resolved by the grid and those that are not.

These problems make the correct inclusion of subgrid processes an important and essential aspect of the numerical modelling of general atmospheric and ocean circulation models.

2.4 Differences in atmosphere and ocean modelling

So far we did not discuss the differences between atmosphere and ocean modelling. The main differences can be summarized by the following items:

- The atmosphere includes more physical processes (e.g., cloud physics).
- The range of spatial scales of processes in the oceans is larger than those in the atmosphere: $O(10) - O(1000)$ km for the ocean compared to $O(100) - O(1000)$ km in the atmosphere.
- The total time scales of the processes taken place vary in the ocean from weeks to centuries, in the atmosphere from hours to years.

However, both areas are computationally intensive. As an illustration we present some numbers of the work by Andrich et al. [7]. They estimated the computational costs of a study of the dynamics of the climate. The general oceanic circulation model in this study was made up of $400 \times 400 \times 16$ gridpoints. For the general atmospheric circulation they used a model with $64 \times 50 \times 11$ gridpoints. The time step was 10 minutes for the atmospheric model, while for the oceanographic model 1 hour was sufficient. These models took approximately 40 hours of CPU per *year*'s simulation on one processor of a Cray-2 for the ocean and 10 hours for the atmosphere. The memory requirements were 150 Mwords and 4 Mwords, respectively. Despite the fact that one deals here with relatively coarse resolution models, it is clear how tremendous the computational problems already are.

2.5 Massively parallel computing

So the scales in space and time of the circulation models require an immense computer power. With the currently available supercomputers it is not possible to run ocean models and/or climate models with the required resolutions in a reasonable time. Also, producing faster chips will finally reach the physical limits. Therefore, one has to switch to massively parallel computing. First steps in this directions for numerical weather forecasting can be found in [8].

Finite difference methods provide in principle an ideal way to use the massively parallel computers efficiently. This idea is not new: already in 1922 L.F. Richardson described a way to make a numerical weather forecast by means of 64,000 human computers [9]. An efficient implementation of finite difference methods can be achieved by the technique of data decomposition. However, the effects of a choice between explicit (nearest neighbour communications) versus implicit (faster convergence) solutions and iterative methods versus direct methods are not clear.

3 Parallel machines

In this section we will sketch briefly the two parallel machines used in our investigation.

3.1 MasPar MP–1 system (SIMD)

The MasPar MP–1 system has a SIMD architecture and consists of the following parts [10]:

– The Processor Element (PE) array, which includes up to 16,384 PEs. Each PE is a register-based RISC processor with 16 or 64 Kbyte data memory. They can perform both 32 and 64-bits floating point arithmetic. The peak performance of a fully configured MP–1 system is 26,000 Mips, and 550 Mflops (64-bits) or 1,200 Mflops (32-bits).

- The Array Control Unit (ACU), which contains a 12 Mips RISC processor for operations on scalar data. Its main task is instruction decode and broadcast to and within the PE array.
- A Unix Subsystem or Front End acting as an interface to MP–1, and as a host for the MasPar Programming Environment (MPPE), compilers (i.e., Fortran90), and other tools.
- A high-speed I/O Subsystem.

3.2 Transputer network

Our MIMD platform is based on the transputer [11]. This is a complete computer on one chip with a processing unit, 4Kb static RAM, two timers, four DMA bidirectional links, a memory interface for up to 4 Gbyte external memory, a performance of 1.5 Mflops (for the T800) and 5–10 Mips, a communication mechanism based on the rendez-vous principle, and internal (on chip, virtual) and external (off chip, physical) channels.

4 Shallow water model

To investigate the suitability of parallel computing for atmospheric and ocean circulation models, we implemented a simple, but representative model on a transputer network. The FORTRAN code for this model is used as a benchmark program in numerical weather prediction. It is written by P. Swarztrauber at the National Center for Atmospheric Research (NCAR), Boulder, USA. The code is very simple and primitive, and based on a finite difference model for the shallow water equations [12]. The model is typical for computations found in atmospheric and ocean models.

4.1 Description of the model

The shallow water model is a simplification of the real world, in which the atmosphere or ocean is represented as a one layer of gas or fluid with a constant and uniform density. The height or thickness of the layer is given by $h(x, y, t)$, where the surface of the earth or sea bottom is equal to $h_B(x, y)$. Furthermore the layer is assumed to be flat instead of being spherical. For this model one can derived (see e.g., [6]) the so-called shallow water equations given by:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} - fv = -g\frac{\partial h}{\partial x} \;, \tag{7}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + fu = -g\frac{\partial h}{\partial y} \;, \tag{8}$$

$$\frac{\partial h}{\partial t} + \frac{\partial}{\partial x}\left\{(h - h_B)u\right\} + \frac{\partial}{\partial y}\left\{(h - h_B)v\right\} = 0 \;, \tag{9}$$

with as Coriolis parameter $f = 2\Omega$, the thickness of the layer given by $h - h_B$, and the other variables as in section 2.1. If the characteristic value for the thickness is equal to D and the characteristic length scale for the horizontal motion is equal to L, the model is valid if $\delta = \frac{D}{L} \ll 1$.

The benchmark developed by P. Swarztrauber is a further simplification. Firstly, the Coriolis term is neglected. Secondly, the domain is a simple rectangle with $a \leq x \leq b$ and $c \leq y \leq d$. Finally, the benchmark model has periodic boundary conditions. This means that the equations 7–9 can be rewritten as:

$$\frac{\partial u}{\partial t} - ZV + \frac{\partial H}{\partial x} = 0 \ , \tag{10}$$

$$\frac{\partial v}{\partial t} + ZU + \frac{\partial H}{\partial y} = 0 \ , \tag{11}$$

$$\frac{\partial P}{\partial t} + \frac{\partial U}{\partial x} + \frac{\partial V}{\partial y} = 0 \ , \tag{12}$$

where $P = p/g$, $U = Pu$, $V = Pv$, $H = P + \frac{1}{2}(u^2 + v^2)$, $Z = \zeta/p$, and $\zeta = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}$. It should be clear that this final model is just a theoretical model concentrating more on the types of calculations than on the physics.

4.2 Finite difference

For a finite difference description one starts to divide the rectangular domain ($a \leq x \leq b$, $c \leq y \leq d$) into gridpoints by defining $x_i = i\Delta x + a$ with $i = 0, \frac{1}{2}, 1, \cdots, M+1$, and $y_j = j\Delta y + b$ with $j = 0, \frac{1}{2}, 1, \cdots, N+1$, where $\Delta x = (b-a)/(M+1)$ and $\Delta y = (d-c)/(N+1)$. The half-valued numbers are necessary, since the grid is a staggered grid, which means that not all variables are associated with each gridpoint [13]. The reason for staggering the grid is to remove the computational mode in the finite difference formulation. As an example the equation $U = Pu$ is transformed to $U_{i+\frac{1}{2},j} = \frac{1}{2}(P_{i+1,j} + P_{i,j})u_{i+\frac{1}{2},j}$.

The time difference scheme applied in this model is a leap-frog scheme plus an additional time filter.

To spread the work over the transputers we applied the technique of data decomposition. In this way each transputer is running the same program on different data, where the total data is divided equally over all transputers.

4.3 Structure of the program

The global structure of the program is shown in figure 1. We have implemented this program on the transputer network. The parts where the periodic continuations are carried out and where the copy routine is called, require communications between the transputers. The others parts contain only local, on chip calculations.

Our implementation of this program exists of 10 processes per transputer: the actual DYN process, a router, four multiplexers, and four demultiplexers. See figure 2 for a schematic display of these processes on a transputer. The (de)multiplexers are necessary to share the physical link between the communication channels of different processes on neighbouring transputers. The router process is responsible for global communication between the transputers via the (de)multiplexers, in particular for the data distribution and data collection through the network. Finally, the DYN process performs the actual calculations, and is able to communicate with the neighbouring DYN processes via the (de)multiplexers in case of nearest neighbour communications. For all other types of communication DYN uses the router process.

```
Program Shallow ;
Initialization ;
While ( nstep <= max_steps ) Do
      Compute U, V, Z, & H from u, v & P ;
      Periodic continuation ( U, V, Z, H ) ;

      Compute unew, vnew & Pnew from
              uold, vold, Pold, U, V, Z & H ;
      Periodic continuation ( unew, vnew, Pnew ) ;

      Update old values from old, current and new values ;
      Copy new values to current values
End While.
```

Figure 1: Global structure of the parallel shallow water benchmark program.

4.4 Results

In table 1 we show some results of the implementation of a 64×64 grid on a network of 4×4 transputers. The calculations are performed with 64-bit precision. The timing are only based on the pure calculations. The collection of the calculated results is not implemented in our model. From table 1 we see that the obtained speed-ups vary from 7.5 to 14.4 depending on the scheduling of the processes and the organisation of the communication steps. The implementation methods II and III contain multiplex and demultiplex processes running with high priority. The high priority of the (de)multiplex processes (a process on a transputer can have either a high or a low priority) guarantees that the communication steps are performed as fast as possible, since as soon as the two necessary processes (a multiplex process on one transputer and a demultiplex process on the second one) are ready to send or receive a message, respectively, this will happen. The other processes with low priority cannot interrupt these high priority processes. The effect of neglecting this issue can be seen in table 1 from the disappointing result for method I.

In our implementation of the benchmark program the communication steps between transputer A and transputer B exist always of a several pairs of send and receive commands: a

Table 1: Some results of the Shallow Water benchmark (64×64 grid) on a network of 16 transputers. For the explanation of the different methods, see text.

Method	1 T800	16 T800	Speed up
I. all processes same priority	1200 s	159 s	7.5
II. mux and demux high priority	1195 s	93 s	12.4
III. as II, but with 'block' send and receive	1195 s	83 s	14.4

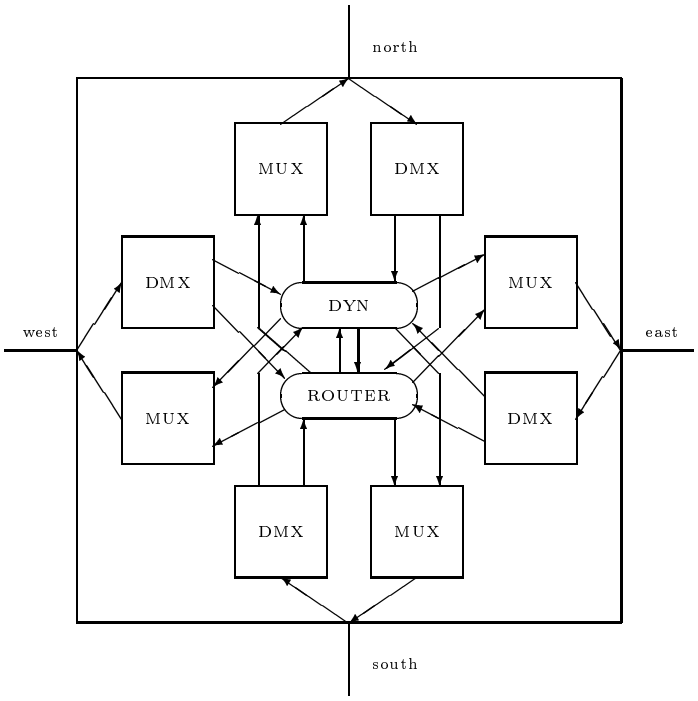

Figure 2: The ten processes on one network transputer.

send and a receive on A and B, respectively, should be followed by a receive and a send on A and B, respectively. The sequence of these pairs of sends and receives differ in the methods II and III presented in table 1. In method II a send on transputer A to transputer B is followed immediately by a receive on transputer A from transputer B, followed by a second pair of send/receive to/from transputer C, and so on till the complete communication step is finished. However, in method III we perform within one communication step first all the necessary sends or receives on one transputer, followed by all receives or sends, or vice versa. So one creates a block of sends and a block of receives. This organisation of the communication results in an improvement of the efficiency by 10%, going up from 80% to 90%.

We have also implemented other methods, which are not described in this paper, and each of them resulted in a different speed-up.

As a conclusion it follows that a very efficient implementation of this benchmark program on a transputer network can be achieved, but that this depends on several non-trivial factors.

5 HIRLAM

In this section we present some results on the implementation of a real production code on parallel systems. This code is a numerical weather forecast program called HIRLAM.

We started a collaboration between Leiden University and the Royal Netherlands Meteorological Institute (KNMI) to see if their weather forecast model for the near future can be implemented efficiently on massively parallel computer systems. In this section we will present some first results on the implementation of a very computational intensive component in the HIRLAM model on a transputer network and on a MasPar system.

```
Program HIRLAM ;
Read data ;
Initialization ;
For each time step
   DYN; { dynamical part }
   PHYS; { physics part }
   6 hourly input of new boundary data
End For ;
Output results.
```

Figure 3: Global structure of HIRLAM.

5.1 Some properties of HIRLAM

HIRLAM [14] stands for High Resolution Limited Area Model and is the result of a development project aiming at a state-of-the-art analysis and forecast system for numerical weather forecasts up to +48 hours. The countries participating in this project are Denmark, Finland, Iceland, Ireland, the Netherlands, Norway, and Sweden. HIRLAM will be or is already used for routine forecasting at several of the participating weather services. It is a production code running on several machines/architectures.

As an example of the computational demands we present some numbers of the model at KNMI. This model uses a $92 \times 81 \times 16$ grid. The applied time step is 6 minutes, and the goal is to produce a 36 hours forecast. At the moment it takes 21 s per time step on a Convex C210, which results in 2 hours for the complete forecast. Taken into account that KNMI has to produce each 6 hours a new forecast, this leaves only 4 hours for analyzing the observations (the input of the forecast model) and making a actual forecast from the output of the numerical model. It should be remarked that the analysis part of the observations is at least as computational demanding as the forecast, but is not taken into account in our investigations until now.

5.2 The discrete HIRLAM model

The discrete HIRLAM model is based on finite difference methods. The horizontal grid is a so-called staggered Arakawa C-grid [13]. As space difference scheme a second order centered space differencing is applied, while for the time difference scheme a second order leap-frog time differencing is used. In this last scheme the technique of explicit splitting is implemented, while a semi-implicit correction is optional together with an Asselin time-filter. The boundaries are treated by a form of boundary relaxation.

In figure 3 we present the global structure of HIRLAM. The routine DYN performs one time step in solving the primitive equations and takes about 30% of the total execution time. The other routine PHYS deals with all subgrid processes and consumes nearly 50% of the total time. The variables at one gridpoint in DYN mainly depends on the variables at the other gridpoints in the horizontal plane. For the variables in PHYS the dependency is mainly in vertical direction. Since PHYS is more time consuming than DYN the most natural way to decompose the data is in the horizontal plane. This will minimize the required communications in the most time consuming routine PHYS. On the other hand the decomposition

results in more communications in DYN, which makes this routine more interesting from a computational point of view. Therefore we will concentrate on DYN in this paper.

As said before DYN performs one time step in solving the primitive equations. It is a 64-bits precision routine written in Fortran77. For a $50 \times 50 \times 16$ grid it has to perform ≈ 12 Million floating point operations.

5.3 Preliminary performance results

In this section we present some preliminary results for the implementation of the DYN routine on the transputer network and the MasPar system.

We start with the results for the transputer network, which is the same network as described in section 3.2. Analyzing the DYN routine one can observe that all calculated variables are a function of a set input variables. As a result during initialisation one can distribute the values of the input variables on the borders of each transputer to some extra buffers of the neighbouring transputers and perform the necessary calculations on these buffers too. In this way the number of communications decreases, while the number of computations increases. The balance between these numbers has to be found.

Table 2 shows the first results obtained by adding the extra buffers. If we take into account only the pure calculations we obtain a speed-up of 7.5 on 16 transputers compared to 1 transputer. At first sight this seems to be a disappointing result. However, due to memory limitations of the transputer it was only possible to use a relative small grid size ($12 \times 12 \times 16$ gridpoints). This means that each transputer has a local grid of $3 \times 3 \times 16$ points. Since each transputer needs two extra buffers the maximum theoretical speed-up is given by $\frac{3\times3}{3\times3+2\times3} \times 16 = 9.6$. The reason that this number is not achieved, is caused by the fact that in our implementation still some communication step are involved, because some variables need two extra boundaries for each border, which would be a little bit overdone for the small grid size.

The second result in table 2 shows that the distribution of data over the network during initialisation and the collection of data (the output results) counter nearly all the gain obtained during the pure calculations. Fortunately, in practice it will not be necessary to collect the results each time step. In a typical case one needs the calculated results once each ten time steps. This results in a total speed-up of 5.0 (see table 2).

As a last point concerning the implementation on transputers, we want to emphasize the fact that the presented results are preliminary. Our research on this topic is still going on at the moment. We have implemented a version with all communications included, so without

Table 2: Preliminary performance results for the implementation of the DYN routine with a $12 \times 12 \times 16$ grid on a 4×4 transputer network. See text for the explanation of the different speed-ups.

	Speed up (16 T800s)
pure calculations	7.5
1 step + data distribution/collection	1.25
10 steps + data distribution/collection	5.0

the extra buffers. This makes it possible to find the balance (if it exists) between communications and extra computations. Furthermore, the grid sizes and number of processors have been varied. Finally, a more detailed performance analysis has been made. All these options show that an efficient implementation depends on many non-trivial issues.

The DYN routine has been implemented on a MasPar system too. The MasPar system which we used, consists of 4K processors with 4Kbyte memory each. As a front-end a DEC station 5000/200 was installed. The MasPar Fortran compiler and operating system versions were 1.2.2 and 1.2.35, respectively. Utilized software tools were the MasPar Programming Environment (MPPE) for execution, debugging and profiling, and the Vast-II compiler for transformation from Fortran77 to Fortran90 source code. The last tool is important, since only parts of the code written in Fortran90 vector syntax are executed in parallel on the PE-array. The other parts run sequentially on the front-end.

We will now discuss some results. Firstly, the actual performance achieved is ≈ 35 Mflops for a $50 \times 50 \times 16$ grid. Since only 50×50 processors are effectively utilized (the variables in the vertical direction are all processed by the same processor element), the theoretical maximum performance for a 4K processor machine in 64-bits precision is given by $137.5 \times \frac{50 \times 50}{4096} = 84$ Mflops. Taken into account this maximum theoretical performance, the obtained 35 Mflops means an efficiency of about 42%, which is typical for routines extracted from real applications running on MasPar systems (average between 40%–60%).

Secondly, one could ask what can be gained by connecting the MasPar system to the front-end, a DEC station, which is already a fast workstation. The obtained speed-up is $15 - 16$ using the MasPar compiler. Compiling the program with the DEC Fortran77 compiler on the front-end instead of the MasPar compiler reduces the speed-up to $9 - 10$.

Finally, an important topic is the location of the data. On the MasPar system the data arrays are placed either on the front-end or on the back-end. The location is specified on a compiler directive given by the user, or automatically determined by the compiler. In the latter case the compiler looks for Fortran90 syntax. If the data array is used in Fortran90 vector syntax, the location of the array will be on the back-end. If no Fortran90 vector syntax is applied for the array, it is placed on the front-end. In case an array is placed on the front-end in one routine and on the back-end in another routine, it has to be copied between front-end and back-end, when the first routine calls the second one. And the array will be copied back when the second routine returns to the first one. This is called sloshing and can reduce the efficiency dramatically: the speed-up obtained by the parallel computations can be vanished completely or even turned into a speed-down.

Currently we are porting the complete HIRLAM code to the MasPar system. In the near future other compute platforms will be investigated for the implementation of the HIRLAM model, but also for porting the GFDL Modular Ocean Model (MOM) [15]. MOM is a result of a collaborative effort at the National Oceanographic and Atmospheric Administration's Geophysical Fluid Dynamics Laboratory in Princeton, New Jersey, USA. MOM is a primitive equation general ocean circulation model intended to be a flexible tool for exploring ocean and coupled air-sea applications over a wide range of space and time scales.

6 Distributed libraries

The research described above should not only result in efficient implementations of atmosphere and ocean models on different parallel architectures, but moreover in the creation of so-called distributed libraries. The last decade a dramatic increase of vector and parallel architectures could be observed. To simplify the creation of efficient algorithms on these different architectures computational primitives have been developed like the basic linear algebra subroutines (BLAS): BLAS1, BLAS2, and BLAS3 [16]. Recently the BLAS 3 standard was fully exploited in the definition of the LAPACK library [17].

The arrival of parallel distributed memory architectures increased the complexity of defining numerical libraries which can be efficiently exploited by these architectures. In fact it is well-known that just a few complete application codes are ported to these new architectures, because of the added complexity of developing distributed memory programs. To overcome the burden of developing application programs for these architectures one could make use of restructuring compilers [18], shared virtual memory support for distributed memory machines [19], annotations for explicit data distribution [20, 21], and problem solving environments to provide a higher level of abstraction for expressing problem instances [22]. Several research projects deal with one or more of these topics. Beside these tools to develop new application packages one needs libraries to ensure easy portability of the codes. Also some of the above mentioned tools, e.g. restructuring compilers and problem solving environments, are depended upon these libraries.

It is our belief that for specific application areas larger grain computational tasks need to be defined and characterized in order to provide the means for achieving efficient implementations on parallel distributed memory architectures. Therefore, the main topic of our research is to identify the actual granularity needed for libraries so that an efficient implementation across a wide variety of architectures can be ensured. This involves a detailed study into the application areas as well as an extended experimentation of the efficiency of the developed library routines on the various architectures available and the dependence of each of these library routines on the context from which they are called. An example of the latter is formed by the occurrence of a call to a library routine which manipulates the same datastructure as a previous called library routine, but requires a different data distribution scheme. In this case it has to be investigated whether the granularity of the library is high enough to compensate for the overhead of a redistribution of the data or whether another library routine should be called to ensure overall efficiency of the code. By starting from an application area directly, our research effort can be seen as the counterpart of an effort which is underway in the parallel numerical algorithm community to group numerical algebra routines together [23, 24], so that efficient implementations can be devised. Both ways form starting points for programming environments and later on for problem solving environments.

References

[1] Executive Office of the President: Office of Science and Technology, *The High-Performance Computing Initiative*, September 1989.

[2] Committee on Physical, Mathematical, and Engineering Sciences, *Grand Challenges: High Performance Computing and Communications*, National Science Foundation, Washington, 1991.

[3] Commission of the European Communities, *Report of the EEC Working Group on High-Performance Computing*, Geneva, 1990.

[4] Mark A. Cane, *Introduction to Ocean Modeling*, in [25].

[5] A.J. Semtner Jr., *Finite-Difference Formulation of a World Ocena Model*, in [25].

[6] Joseph Pedlosky, *Geophysical Fluid Dynamics, second edition*, Springer-Verlag, New York, 1987.

[7] Patrick Andrich, Gurvan Madec, and Didier L'Hostis: *Performance Evaluation for an Ocean General Circulation Model: Vectorization and Multitasking*, in proceedings 1988 International conference on supercomputing, ACM, St. Malo, France, July 1988, pp 295–302.

[8] G.-R. Hoffmann and D.K. Maretis (editors), *The Dawn of Massively Parallel Processing in Meteorology*, Proceedings of the 3rd Workshop on Use of Parallel Processors in Meteorology, Springer-Verlag, Berlin, 1990.

[9] L.F. Richardson, *Weather Prediction by Numerical Process*, Cambridge University Press, Cambridge, 1922.

[10] MasPar, *MasPar MP–1 Hardware Manuals*, July 1992.

[11] INMOS, *Transputer Reference Manual*, 1986.

[12] R. Sadourny, *The Dynamics of Finite-Difference Models of the Shallow-Water Equations*, Journal of the Atmospheric Sciences 32 (1975) 680-689.

[13] A. Arakawa, *Design of the UCLA General Circulation Model*, Department of Meteorology, University of California, Los Angeles, Technical Report No. 7, 1972.

[14] P. Kallberg (editor), *Documentation Manual of the Hirlam Level 1 Analysis-Forecast System*, June 1990.

[15] R. Pacanowski, K. Dixon, and A. Rosati, *The G.F.D.L. Modular Ocean Model Users Guide version 1.0*, GFDL Ocean Group Technical Report #2, Geophysical Fluid Dynamics Laboratory, Princeton, 1991.

[16] J. Dongarra, J. Bunch, C. Moler, and G.W. Stewart, *LINPACK User's Guide*, SIAM Publications, Philadelphia, 1979.

[17] J. Demmel, *LAPACK: a Portable Linear Algebra Library for High-Performance Computers*, Concurrency: Practice and Experience, vol. 3(6) (1991), 655–666.

[18] H. Zima, and B. Chapman, *Supercompilers for Parallel and Vector Computers*, Frontier series, ACM Press, New York, 1991.

[19] T. Priol, and Z. Lahjomri, *Trade-offs Between Shared Virtual Memory and Message-passing on an iPSC/2 Hypercube*, Technical Report 637, IRISA, 1992.

[20] E. Paalvast, A. Gemund, and H. Sips, *A Method for Parallel Program Generation with an Application to the Booster Language*, in proceedings of the 1990 ACM Conference on Supercomputing, ACM Press, 457–469.

[21] Proceedings of the *High Performance Fortran Forum*, Houston, Texas, January 27–28, 1992.

[22] E. Houstis, J. Rice, and T. Papatheodorou, *//Ellpack: An Expert System for the Parallel Processing of Partial Differential Equations*, Math. and Comp. in Simulation, 31 (1989), 497–507.

[23] A.T. Chronopoulos, and C.W. Gear, *s-Step iterative methods for symmetric linear systems*, J. on Comp. and Appl. Math., 25 (1989), 153–168.

[24] J. Dongarra, I. Duff, D. Sorensen, and H. van der Vorst, *Solving Linear Systems on Vector and Shared Memory Computers*, SIAM Publishers, 1991.

[25] James J. O'Brien (editor), *Advanced Physical Oceanographic Numerical Modelling*, Proceedings, NATO ASI Series, Reidel Dordrecht, 1986.