# A PARALLEL IMPLEMENTATION OF THE HIRLAM MODEL

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

Gerard Cats
Royal Netherlands Meteorological Institute
P.O. Box 201, 3730 AE De Bilt, The Netherlands

**Abstract**

This paper is a status-report describing the effort to implement the HIRLAM numerical weather prediction model on a MasPar MP–1 system. Several results of the parallel implementation of the forecast model are presented. Differences with more traditional implementations on vector-computers are outlined. While work is still going on, this report mainly focuses on explicit Eulerian methods. Semi-implicit methods and Lagrangian methods will be treated in the next implementation.

## 1 Introduction

It is well known that atmospheric circulation models, as part of numerical weather prediction systems, require enormous computer power. Coupled to ocean models, they form an even more computationally intensive task: climate models. As a natural consequence atmospheric modeling has always been an important application for high performance computer systems.

Since the 70's, numerical weather forecasting has been successfully utilizing vector-computers. During the last decade, one could observe the arrival of very powerful parallel computer systems. However, only a few complete application codes have been ported to these new parallel architectures, because of the additional complexity of parallelizing programs.

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

This report describes an effort to implement a complete application (and production) code on a massively parallel computer system. The application code is a numerical weather forecast model. It is part of the HIRLAM system, which is described in more detail in section 2. Section 3 provides the reader an overview of the hardware and software components of the MasPar MP–1, the parallel platform used in this investigation. The motivation to port the model to a parallel system is given in section 4. The rest of that section describes several implementation issues. Results of the current parallel implementation are presented in section 5. Finally, section 6 contains the conclusions.

# 2 HIRLAM

HIRLAM is a state-of-the-art analysis and forecast system for numerical weather forecasts up to +48 hours. The name HIRLAM stands for HIgh Resolution Limited Area Model. The HIRLAM system has been developed by the HIRLAM-project group, a cooperative project of Denmark, Finland, Iceland, Ireland, The Netherlands, Norway and Sweden. Currently it is in production for making actual forecasts in Denmark, Finland, and Sweden. In the near future HIRLAM will also be used for routine forecasting at several of the other participating weather services. At the moment it runs on different compute platforms, most of them having a vector-architecture, like Cray and Convex.

The complete system consists of three parts: the analysis system dealing with the observations, the initialization scheme, and the actual forecast model. In this paper we only concentrate on the forecast model. The initialization step is modest in computer requirements; furthermore, parallelization can proceed along the same lines as for the forecast model, therefore we will not address this step further. However, the analysis system differs completely from the forecast model and has therefore not been investigated until now.

For a detailed description of the numerical methods and their implementation in the HIRLAM model the reader is referred to the HIRLAM documentation [2]. We limit ourselves here only to a brief overview.

The forecast model contains two main components. Firstly, one has a 'dynamics-part' (called DYN), which solves the primitive equations during one time-step. And secondly, the 'physics-part' (PHYS) responsible for the parameterization of several physical processes. The model provides the user with different options to chose between alternative numerical methods. The most important options are: gridpoint versus spectral techniques, explicit versus semi-implicit time schemes, and Eulerian versus semi-Lagrangian methods. As a starting point we choose for the easiest options with respect to the parallelization of the model, which are gridpoint, explicit, and Eulerian. The other options will be implemented in the next parallel version.

Keeping this choice of options in mind, the discrete HIRLAM model is based

on finite difference methods. The horizontal grid structure is a staggered Arakawa C-grid, while its vertical structure is based on a general pressure and terrain following vertical coordinate $\eta(p,p_s)$. A centered space differencing is applied, while the time differencing is based on a leap-frog scheme with explicit splitting and an Asselin time-filter. Lateral boundaries are treated with a boundary relaxation scheme with a 6-hourly data input interval.

# 3 MasPar MP–1

In this section we present some characteristics of the massively parallel computer system we have used in this investigation: a MasPar MP–1 system. For a completed description see [1]. A MasPar MP–1 system is a SIMD architecture. In the next subsections hardware, communication, and software topics, respectively, will be discussed.

## 3.1 Hardware

In figure 1 a schematic overview of the MasPar system is shown. The most interesting component of the hardware is the Processor Element (PE) array. This component can contain from 1,024 (1K) up to 16,384 (16K) PEs. Each PE is a load/store arithmetic processor with 16 Kbytes or 64 Kbytes data memory. Furthermore, a PE has forty 32-bit private registers, thirty-two special purpose registers, and can calculate with 1, 8, 16, 32, and 64 bit integers. The floating point precision is 32 or 64 bits. A full 16K system has a peak performance of 26,000 MIPS and 550 Mflops (64-bits) or 1,200 Mflops (32-bits).

A second component in the MasPar system is the Array Control Unit (ACU). This is the single control unit for all PEs. The ACU is a register-based load/store processor with thirty-two 32-bit registers, 128 Kbytes data memory and 1 Mbytes instruction memory. The limit of its virtual instruction address space is 4 Gbytes. The ACU is responsible for the instruction decode and broadcast of instructions and data. It also includes a 12 MIPS scalar RISC processor for operations on scalar data.

The third component is the front-end (versus the two first components, which are called the DPU, which stands for Data Parallel Unit). In our case the front-end is a Dec workstation. This Dec 5000 station determines the operating system (Ultrix) and runs also other software like Dec-windows or X. It serves as an interface to the MP–1 and is host for tools and compilers.

The last hardware component is optional: the parallel disk array. This disk array contains 4, 8, or 16 disks with a total capacity up to 11.2 Gbyte. Its sustained I/O rate is equal to 9 Mbyte/s. It enables the user to have parallel access to large files.

Figure 1: The MP–1 system architecture.

## 3.2 Communications

In parallel distributed memory computer systems communications between the processors form a critical component. They are often the bottleneck in achieving higher performance. For the MasPar MP–1 system one can distinguish three types of communications.

Firstly, one has the communication between the Processor Elements (PEs). This type can be divided into two classes. One is the so-called Xnet communication. This is a special case of communication, namely it can perform nearest-neighbor communications. The Processor Element Array is arranged in a 2-dimensional mesh with toroidal wrap-around. With Xnet communications one can send to or receive data from the eight neighboring PEs, so in horizontal, vertical and diagonal directions. This can be extended to communication between two PEs that lie on a straight line in each of the eight directions. The maximum communication speed using Xnet is 23 Gbyte/s for a full configuration.

The second class of communication between PEs is the Router communication. This class makes it possible to send/receive data between two arbitrary PEs, so it takes care of the global communications. The communication time is independent of the distance between the PEs, but its maximum speed is considerably slower than for Xnet communications: 1.3 Gbyte/s. Another limitation is that there is only one Router channel for 16 PEs.

Secondly, one has the communication between the Array Control Unit (ACU) and the PE array.

Finally we have communication channels between the FE and the DPU vice versa. Usually these channels are used to distribute input data over the DPU and return output data from the DPU. The number of data involved is big, and the transfer rate is limited. Therefore these communications are extremely expensive and should be limited as much as possible. The programmer is responsible for distributing the data over the FE and the DPU, and much of a conversion effort should be aimed at keeping the data as much as possible on the DPU.

## 3.3 Software

As stated above, the front-end serves as an user-interface to the MasPar system. It also determines the operating system and as a result one can use all software which is available for the front-end and operating system.

In addition we have the dedicated software to utilize the massively parallel system. We want to mention some examples. The first example is the MasPar Fortran (MPF) compiler. This compiler is an implementation of Fortran 90. This indicates the programming model for the MasPar system: data-parallel programming. Operations on Fortran-arrays expressed by the Fortran 90 array-syntax will be executed in parallel, and the arrays concerned will be distributed over the PEs. Operations with Fortran 77 syntax will be executed sequentially.

To translate Fortran 77 programs (like HIRLAM) to Fortran 90 MasPar provides the VAST-II compiler.

A second programming language is the MasPar Parallel Application Language (MPL). This language is based on C, and has extensions to provide access and explicit control over the Data Parallel Unit (DPU). It can be considered as the assembly language for a MasPar system.

Finally, we want to mention a very powerful software tool, the MasPar Programming Environment (MPPE). MPPE has been shown to be very useful for executing, debugging, profiling, and visualizing programs or program parts.

# 4 Implementation Issues

In this section we present several implementation issues one encounters during porting the 28,000 lines of Fortran 77 HIRLAM code to MasPar system. For most of these issues one has to make a decision how this topic should be treated.

But first we want to make clear what our motivation is to port the HIRLAM code to a massively parallel system. Basically there are three questions we want to answer:

1. To what extend should the HIRLAM code be adapted to run efficient on a massively parallel computer system?

2. What are the advantages and disadvantages to deploy these parallel systems for a numerical weather prediction system?

3. How do the different options within HIRLAM (explicit $\leftrightarrow$ semi-implicit, gridpoint $\leftrightarrow$ spectral, Euler $\leftrightarrow$ semi-Lagrangian) compare on a massively parallel processor?

From a computer science point of view, there are several other reasons and interests, but these fall outside the topic of this workshop.

To give some feeling about the HIRLAM code itself, in figure 2 the global program structure of HIRLAM is presented, where we restricted ourselves to gridpoint, explicit and Eulerian calculations.

**Data distribution.** The first issues concerns the distribution of the data. On the one hand one has to deal with 3-dimensional fields in the forecast model, and on the other hand the processors in the parallel system are organized in a 2-dimensional mesh. One way to solve this problem is to take into account the number of dependencies in north-south, east-west, and vertical directions, since these dependencies will result in communications between processors if the data is mapped on different processors. Inspecting the two time-consuming parts of HIRLAM, one finds that in PHYS mainly vertical dependencies are involved, while in DYN the dependencies are mainly in the horizontal plane. Besides the

```
Read start data GETDAT;
Read boundary data GETDAT;
Initialization;
Loop for each time-step:
   Dynamics DYN;
   Physics PHYS:
      Hybrid coordinates HYBRID;
      Radiation scheme RADIA;
      Vertical diffusion VDIFF;
      Temperature and specific humidity
         convection scheme KUO;
      Stratiform condensation scheme COND;
      'check-routine' QNEGAT;
      Soil processes SURF;
   Horizontal diffusion HDIFF4;
   Boundary relaxation BNDREL;
   Statistics STATIS;
   Print statistics PRSTAT;
   6-hourly input of new boundary data GETDAT;
   Output results PUTDAT;
   Time-filter TIMFIL;
End_Loop.
```

Figure 2: The global program structure of HIRLAM.

number of dependencies in PHYS is much larger than in DYN. One solution is to redistribute the data every time between the calls to DYN and PHYS. However, this will result in a serious performance degradation. Because the number of dependencies in PHYS is much larger than that in DYN, we chose for the solution where the data are mapped on the processors by projection of the vertical dimension onto the horizontal plane.

**Tables versus re-computation.** The original HIRLAM contained several large tables each specifying a function of some variable. There are several options how to treat these tables. One can keep these tables on the front-end, but thereby introducing many communications between the front-end and the PEs. A second option is to distribute them over the PE-array. However, this will result in many communications between the PEs. One can also keep copies of the tables on each PE. This results in a waste of memory. Therefore we choose for the option to recalculate an entry in one of these tables if necessary, so the tables are not stored anymore. In this way we reduce the memory requirements and number of communications. On the other hand the calculation time is increased.

**Disk storage.** Since communication between the front-end and DPU is very time-consuming, it should be reduced to a minimum. In the original HIRLAM code some values were stored on disk and read every time they were necessary in a routine. One example were the boundary values needed in the boundary relaxation routine BNDREL. To eliminate the expensive communications, these boundary values were distributed over the PE-array. An additional advantage is that these values could be distributed in a very natural way, because of their spatial meaning.

**Arrays as parameters.** As a consequence of the distributed memory system of the MasPar MP–1 one cannot assume a sequential address space. This means that the actual and formal parameter in a subroutine call should always have the same size and shape. So 'dirty' Fortran 77 tricks like passing only the first address of an array-portion with some arbitrary length or passing 1-dimensional arrays to routine, which require a 2-dimensional parameter, and vice versa, are not allowed. Unfortunately, these tricks were heavily used in the original HIRLAM code.

**Compiler directives.** The most important compiler directives in MasPar Fortran are ONDPU, ONFE, and MAP. With the ONDPU-directive the user can specify that an array should be placed on the Data Parallel Unit. This in contrast with the ONFE-directive, where the array is placed on the front-end. The MAP-directive tells the compiler how the array should be mapped on the PE array.

The use of the ONDPU and ONFE directives is straightforward. However, we should mention that these directives should be specified in all routines, which contain the arrays involved. For arrays in common blocks this can be simplified by the use of include-files, but for arrays which are passed as parameters to routines the only solution is to repeat the directive in each routine.

The reason we had to use the MAP-directive has a historical reason, since the default mapping of arrays on the MasPar system is very natural. An 1-dimensional array is mapped on the PE-array by distributing the array-elements in the $x$-direction of the PE-array. If there are more elements than processors in the $x$-direction, the distribution is continued on the second column of processors, and so on. If the number of array-elements exceeds the total number of processors, the rest of the elements is distributed in the same way as the first part. The first dimension of a 2-dimensional array is always mapped in the $x$-direction of the PE-array, and the second dimension in the $y$-direction. The third and higher dimensions of more dimensional arrays are placed in the memory of the PEs.

This natural default mapping is not suited for the HIRLAM model. Because the HIRLAM model was written for vector-computers, arrays have been arranged into long vectors, by combining the two horizontal dimensions into one. So an array with 'natural dimensions' U(MLON,MLAT,MLEV) is used as U(MLON$*$MLAT,MLEV). The default mapping would project the second dimension, i.e. MLEV, onto the $y$-direction of the PE array. With the MAP directives

the data are forced to be distributed with the first dimension, MLON∗MLAT, to fill the PE array. The second dimension, MLEV, is then a layer structure over this. The conversion effort would have been much facilitated if the default mapping could have been modified by e.g., compiler options.

To be more specific, we had to include 718 directives to port the restricted (only explicit, Eulerian and gridpoint) version of the HIRLAM model to the MasPar system. With the other options fully implemented this number will increase dramatically, and thereby striking the question if the usage/inclusion of compiler directives can be considered as handling a second programming language?

**Interface blocks.** An interface block is a Fortran 90 concept, in which the user specifies the type and size of the parameters for each subroutine called in a program or subroutine. In MasPar Fortran it is mandatory to use interface blocks for all subroutines with parameter mentioned in compiler directives, in particular the MAP-directive. Including these interface blocks in HIRLAM resulted sometimes in a fourfold increase of the number of source-lines of a subroutine.

**Fortran 90.** All Fortran 90 code, except for one routine, was translated from the original Fortran 77 source code by means of the MPVAST compiler. VAST was not able to produce efficient code for one small routine of about 10 lines. This was done by hand. The use of VAST resulted sometimes in the inclusion of VAST-compiler directives.

**Not implemented yet:** PUTDAT and RADIA. Due to several problems these two routines could not be ported successfully to the MasPar system yet. As a result no output-field could be generated, since that is the task of PUTDAT. The second routine, RADIA, executes a radiation scheme.

# 5 Results

In this section we present several timings achieved by porting the HIRLAM code to a MasPar MP–1 system. As stated before, the current implementation is restricted to the gridpoint version with explicit time stepping and Eulerian methods. Besides the current timings do not include the routine RADIA and PUTDAT, because they have not been ported to the MasPar system yet.

As test model we used a $64 \times 64 \times 16$ grid. This grid fits perfectly on a 4K ($64 \times 64$) PE array. On a 1K ($32 \times 32$) PE array it results in 4 layers of $32 \times 32 \times 16$ gridpoints. Varying the horizontal gridsize between 32 and 64 will not result in different timings for the parallel part of the calculation, since it only means that some PEs are not active during the complete calculation.

In table 1 the elapsed times are presented to complete the input phase and 61 time-steps. The third version, 4K (opt), has been compiled with the Nodebug and

Omax options. These options result in a code without any debug information, and the most optimized code will be produced by the compiler.

These results are food for thought. The improvement (speed-up) achieved by increasing the number of processors from 1K to 4K is only 2.5, while a factor of at least 4 should be expected (remember we deal with a SIMD architecture, where all processors perform the same instruction; since the our test-grid fits perfectly on the 4K processor system, one needs only one 'stage' to perform the necessary computations; however, on a 1K system the data is distributed in 4 layers, so one needs 4 'stages', which should result at least in a factor of 4 in performance difference). Furthermore, one sees an improvement of about 10% by taking advantage of the optimizing compiler options.

To understand these facts we need a more detailed analysis of these times. In table 2 the times are broken down into the time needed for the input phase, which can only carried out sequentially, and the time for executing all time-steps. The execution of a time-step is the part of the program which can be parallelized. Several observations can be made from table 2. Firstly, one sees indeed the sequential character of the input phase, since there is no improvement by increasing the number of processors. This is not only due to our implementation, but can be assumed to be true for all parts in parallel programs on distributed memory system, where data has to be read from disk and moved or distributed across the memories of the available processors.

A second and even more serious observation is the total amount of time the input phase takes. It varies from 8% for a 1K to 21% for a 4K system of the total execution time. This is a considerable - and even an unacceptable - amount of time for our application, since it is expected that the output phase (the routine PUTDAT, which unfortunately could not be implemented yet) will cost the same order of time. The input phase consists of two parts: reading the initialization values and boundary values, each responsible for roughly half of the total time spent in this phase. Each 6 hours simulation time new boundary values have to be read, which will take about 16 sec. But the output phase will be called each 1 hour of simulation time, and this will also take about 16 sec. Depending on the size of the time-step (but for explicit methods 1 hour will correspond with at most

Table 1: Elapsed times (in sec) to complete the input phase and the time-steps 0–60 of the HIRLAM model. Times are shown for a 1K and 4K processor machine. See text for the meaning of the 4K (opt) version. The italic numbers show the different ratio's.

| | 1K | 4K | 4K (opt) |
|---|---|---|---|
| Total time | 391.31 | 157.76 | 143.02 |
| (input phase | *1* | *2.5* | *2.7* |
| and steps 0–60) | *0.4* | *1* | *1.1* |

Table 2: Elapsed times (in sec) broken down into the input phase and the time-steps with overhead. See for notation the caption of table 1.

| | 1K | 4K | 4K (opt) |
|---|---|---|---|
| Read input (GETDAT) | 32.09 (*1*) | 32.86 (*1*) | 23.51 (*1.4*) |
| Steps 0–60 + overhead | 359.21 (*1*) | 124.91 (*2.9*) | 119.50 (*3.0*) |

Table 3: Elapsed times (in millisec) for 1 time step with the break down into the time spent in routines DYN and PHYS. See for notation the caption of table 1.

| | 1K | 4K |
|---|---|---|
| 1 Time step | 5800 (*1*) | 1930 (*3.0*) |
| Dynamics (DYN) | 2037 (*1*) | 680 (*3.0*) |
| Physics (PHYS) | 3393 (*1*) | 1105 (*3.1*) |

60 time-steps) these in- and output phases are very expensive. One way to solve this problem is to use special hardware, like the Parallel Disk Array provided by MasPar. This will be investigated in the near future.

The third observation is the fact that the compiler optimization mainly concerns the sequential part of the program, see table 2. If this optimization concerns I/O operations, sequential source-code in the program or both, has not been investigated. Since we are mainly interested in the parallel part of the program, we will not include the timings of the optimized version in the rest of this report. However, these compiler options result in an improvement of 40% for the input phase, and can therefore contribute to the problem mentioned above concerning the in- and output phase.

Finally, the speed-up is only 2.9 going from 1K to 4K processors. We will investigate this fact in the rest of this section.

To give an idea about the time one time-step takes, and how the different components contribute to this time, we present in table 3 these times, and concentrate specially on the two main time-consuming part: the routines DYN and PHYS. From this table we see that DYN contributes for 35% and PHYS for about 57.5% to the total time for one time-step, nearly independent of the number of processors used. Furthermore, again only a speed-up of roughly 3.0 is achieved by enlarging the PE array.

To investigated this disappointing speed-up of 3.0 in more detail, we timed two types of subroutines in more detail. The first type concerns routines, which should theoretically (= expected) contain relatively many PE communications due to the choice of data-distribution. The second type is just the opposite: routine with relatively few PE communications. In table 4 the results are presented. From this table we can conclude that the routines with 'many' PE communications

behave as expected considering the speed-up of 4.1. However, the routines with 'few' communications show a strange tendency by a speed-up of only 3.1–3.3.

Therefore, we analyzed the instructions generated by the compiler for the last type of routines. It turned out that the compiler generates many redundant Xnet communications for these routines! A work-around for this problem was to make sure that array-dimensions and several loop-bounds were known at compile time. From a research point of view this is a serious restriction, but for a production code this will improve the performance on all kind of platforms. However, even the fact that this improvement for the HIRLAM code could be achieved by changing some parameter statements in the subroutines, we can only present the resulting timing for three routines at the moment, see table 5. We included in this table also the resulting times for one time-step. The table shows a dramatic decrease of the elapsed times. Also the resulting speed-ups are accordingly: more than 5.0. Unfortunately we were not able to extend this observation to all routines before the deadline of this paper/workshop. However, it is expected that this improvement holds for all other routines, even for the routines with 'many' PE communications.

To compare the obtained times with other compute-platforms, we ran the same HIRLAM test-run on different systems. Results are shown in table 6.

To fit in with the 'common' way to compare different computer systems we present in table 7 the Mflop-rate for the DYN-routine on the two MasPar configurations and a 1-processor Cray-YMP. One important remark concerning these results: the Cray results were in 64-bits precision, while the MasPar calculation were in 32-bits. However, to justify this difference we want to stress the fact that it is not easy (or even possible) at the moment to switch from 32-bits to 64-bits variables on one machine, if one has to handle a large production code. In the production version one has to start with the default precision on a machine. In most cases this is 32 or 64-bits precision. By compiler options (like i8 or r8) about 90% of the job to achieve the required precision can be done, but the other 10% remains and should be modified by hand.

Table 4: Elapsed times (in millisec) of some routines with theoretically many and few PE-communications. See for notation the caption of table 1 and for the difference the text.

| many PE communications | | | few PE communications | | |
|---|---|---|---|---|---|
| Routine | 1K | 4K | Routine | 1K | 4K |
| DYN | 875 (*1*) | 215 (*4.1*) | KUO | 1682 (*1*) | 504 (*3.3*) |
| HDIFF4 | 496 (*1*) | 121 (*4.1*) | VDIFF | 640 (*1*) | 207 (*3.1*) |

Table 5: Elapsed times (in millisec) of the three improved routines compared with the original versions. Also the resulting time for 1 time step is shown. See for notation the caption of table 1.

| | 1K | 1K (improved) | 4K | 4K (improved) |
|---|---|---|---|---|
| KUO | 1682 | 871 (*1*) | 504 | 152 (*5.7*) |
| VDIFF | 640 | 430 (*1*) | 207 | 80 (*5.4*) |
| COND | 383 | 301 (*1*) | 118 | 59 (*5.1*) |
| 1 Step | 5800 | 4697 (*1*) | 1930 | 1392 (*3.4*) |

# 6 Conclusions

Before starting the conclusions, we want to stress the fact that all tables in the previous section contain preliminary results. Further and much more detailed performance analysis and improvement are required. Besides we do not want to restrict only to a few options in the existing HIRLAM model, but to parallelize the complete code with all options.

The conclusions can be drawn by answering the three basic questions formulated in section 4.

Question 1: *To what extend should the HIRLAM code be adapted to run efficient on a massively parallel computer system?*

Firstly, some of the routines had to be changed. Most of these changes do not degrade the performance on vector or sequential platforms (some of them even led to a better performance). These changes lead to recommendations for future developments on the HIRLAM code, and in fact should not be counted as efforts to port the code to a massively parallel machine, but rather as efforts to make the code more portable in a general sense.

Secondly, the inclusion of compiler directives is questionable. The main problem is that directives result in the introduction of some kind of second programming language. This is unacceptable from point of view of maintenance. This

Table 6: Comparison of timings (in sec) on different systems for 1 time step.

| | 1 time step (sec) |
|---|---|
| 1K MasPar | 5.8 |
| 1K MasPar (improved) | 4.7 |
| 4K MasPar | 1.9 |
| 4K MasPar (improved) | 1.4 |
| Dec 5000 (front-end) | $\approx 60.0$ |
| Convex C210 | $\approx 8.0$ |

Table 7: Mflops-rate for the DYN routine on several systems.

| | Mflops | percentage of peak performance |
|---|---|---|
| 1K MasPar MP–1 | 22.5 | 30% |
| 4K MasPar MP–1 | 91.4 | 30% |
| CRAY-YMP (1 proc.) | 193.8 | 58% |

experience is not restricted to the MasPar system in particular, but more to the usage of directives in general, where one assumes that that will solve most problems without introducing other difficulties.

Furthermore, the mandatory usage of interface blocks on the MasPar system is again from a point of maintenance not very welcome. To strengthen this fact, we do not see any need to use these interface blocks for the case they are mandatory now.

A fourth fact is the I/O problem. This holds for all massively parallel systems in general. It is open at the moment how efficient the input and output phase of HIRLAM can be handled by the MasPar system. We expect that the Parallel Disk Array will improve the current observations considerably, but this has not been investigated yet.

As a final point we can say that the already obtained performance of the MasPar system is quite acceptable. Especially, considering the improvements which could be achieved by a detailed analysis, we expect a very good cost/performance ratio.

Question 2: *What are the advantages and disadvantages to deploy these parallel systems for a numerical weather prediction system?*

The advantages are clear: performance, cost-efficiency and scalability (at least for pure explicit methods). An open question is if and how the analysis scheme could be implemented on a SIMD architecture. At the moment we expect that the analysis scheme is more suitable for a MIMD platform.

Question 3: *How do the different options within HIRLAM (explicit $\leftrightarrow$ semi-implicit, gridpoint $\leftrightarrow$ spectral, Euler $\leftrightarrow$ semi-Lagrangian) compare on a massively parallel processor?*

This question is still open and cannot be answered at the moment. Currently we are working on the incorporation of semi-implicit methods. This technique results in solving Helmholtz equations on a massively parallel system and therefore to global communications. Another option, applying spectral methods, is currently under investigation by Nils Gustafsson (SMHI), also on a MasPar system. Spectral methods require global communications as well, so there may be a relation with the semi-implicit option. Further results will soon become available.

Finally, we want to mention that we are also working on MIMD-implementations of HIRLAM. This is a continuation of the work presented in 1988 in this series of workshops, see [3]. Besides we will start next year an investigation on the parallelization of the analysis scheme.

# References


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

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

[3] G. Cats, H. Middelkoop, D. Streefland, and S.D. Swierstra, *A Meteorological Model on a Transputer Network*, pp. 47–75 in [4].

[4] 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).