A Comparison of Parallel Programming Paradigms and Data Distributions for a Limited Area Numerical Weather Forecast Routine

Robert van Engelen[*] & *Lex Wolters*[†]

High Performance Computing Division
Department of Computer Science, Leiden University
P.O. Box 9512, 2300 RA Leiden, the Netherlands
{robert,llexx}@cs.LeidenUniv.nl

Abstract

In this paper the impact of parallel programming paradigms and data distributions on the performance of a parallel finite difference application is investigated. The finite difference application is one of the kernel routines of a limited area numerical weather forecast model that is in use for producing routine weather forecasts at several European meteorological institutes. Results are shown for CRAY T3D and MasPar systems.

1 Introduction

The HIRLAM (HIgh Resolution Limited Area Model) system [1] is a production code written in Fortran 77. This state-of-the-art limited area numerical weather forecast system has been optimized for efficient execution on vector machines. However, even the computer power of vector architectures limits the model resolution to values that are unsatisfactory from a physics point of view. Therefore lower resolutions are enforced, since the weather forecasts must be available within a reasonable amount of time. These considerations focused current investigations on the design of parallel implementations of the HIRLAM system [2, 3, 4]. These investigations were mainly aimed at data-parallel implementations and parallel sub-domain implementations employing a domain decomposition approach on distributed memory architectures. However, it is not clear yet what the impact of data distributions and the employment of other parallel programming paradigms will be on the performance of a parallel implementation of the system. Since it is impossible to investigate these topics for the complete HIRLAM code, we will consider them only for a kernel routine, the DYN routine, which is one of the most time consuming parts of the HIRLAM system. In general, solving nonlinear partial differential equations by means of finite difference methods results in explicit coding of difference schemes. In the HIRLAM system the DYN routine implements these schemes.

[*]Support was provided by the Foundation for Computer Science (SION) of the Netherlands Organization for Scientific Research (NWO) under Project No. 612-17-120.

[†]Support was provided by the Esprit Agency EC-DGIII under Grant No. APPARC 6634 BRA III.

Published in proceedings of the 9th ACM International Conference on Supercomputing, July 1995, Barcelona, Spain, ACM Press, pp. 357–364

Several data distributions and parallel programming paradigms will be investigated in this paper for the implementation of a parallel DYN code on two different distributed memory massively parallel computers. The selected architectures are the CRAY T3D and the MasPar computer systems. In general, the choice of data distribution and parallel programming style for a parallel DYN implementation have immediate consequences for the design of a parallel implementation of the HIRLAM system. Among the most important consequences are performance, portability, and generality of the parallel code.

This paper is organized as follows. Section 2 introduces the HIRLAM system and briefly reviews the DYN routine. Section 3 describes the parallel platforms used for the investigation in this paper. Data distributions are discussed in Section 4. In Section 5, several parallel implementations of the DYN routine are described using various parallel programming paradigms. Performance results are presented and discussed in Section 6. To conclude, Section 7 summarizes our results and conclusions.

2 Overview of HIRLAM

In this section the HIRLAM production code [1] and the DYN routine in particular, are briefly reviewed. The HIRLAM system was developed by the HIRLAM-project group, a cooperative project of Denmark, Finland, Iceland, Ireland, the Netherlands, Norway, and Sweden.

The HIRLAM weather forecast system contains five prognostic variables: two horizontal wind components, temperature, specific humidity, and surface pressure. All model parameters are kept in memory. The core of the HIRLAM system is provided by the 'dynamics' and the 'physics' routines. In the dynamics routines, a set of three-dimensional coupled nonlinear hyperbolic partial differential equations is solved. This set of so-called Primitive Equations, see e.g. [5], contains two horizontal momentum equations, a hydrostatic equation, a mass continuity equation, a thermodynamic equation, and a continuity equation for water vapor. Several solution methods are implemented. Most of them use explicit Eulerian gridpoint finite differencing on a staggered Arakawa C-grid with centered space-differencing and leapfrog time-differencing, resulting in a second order accuracy of the approximations. The physics routines compute the parameterized processes which are described by the aggregate effect of the physical processes with scales smaller than the model resolution. Almost all the physical processes are one-dimensional in vertical columns. The physics model can be easily solved by N disjunct processes if the data is horizontally distributed over a two-dimensional processor grid, where N is the number of gridpoints in the horizontal.

In the dynamics part the DYN routine computes the explicit tendencies of the five prognostic variables for each time step. In the routine finite difference techniques are adopted on a three-dimensional grid.

As an illustrative example of a finite difference technique, consider the one-dimensional advection of temperature. In analytic form, one has

$$\frac{\partial T}{\partial t} = u \frac{\partial T}{\partial x}. \tag{1}$$

In the most simple form (no staggering of the grid) finite differencing of equation (1) with a leap-frog time-stepping results in

$$T(x, t + \Delta t) = T(x, t - \Delta t) + \frac{\Delta t}{\Delta x} u(x,t) \cdot (T(x + \Delta x, t) - T(x - \Delta x, t)), \tag{2}$$

where $T(x,t)$ is the temperature in gridpoint x at time t, $u(x,t)$ is the wind-component in the x-direction in gridpoint x at time t, Δt is the time step, and Δx is the grid-distance in the x-direction. In the DYN routine the explicit tendencies of the five prognostic variables are computed for each horizontal layer of the three-dimensional grid, using similar finite difference equations. On parallel architectures the $\pm\Delta x$ in equation (2) can result in communications between processors if the grid is distributed over the processor array. Due to optimizations on vector architectures the three- and two-dimensional fields in the HIRLAM reference code are stored as two- and one-dimensional arrays, respectively, where the horizontal dimension is reduced. In total, each layer requires 28 one-dimensional loops over the horizontal grid to compute the explicit tendencies.

3 Overview of the Parallel Platforms

In this section the CRAY T3D and MasPar massively parallel processing systems are briefly reviewed. Both systems are used for the investigation in this paper.

T3D. The CRAY T3D system is a scalable MIMD system. This distributed memory concurrent computer (or *multicomputer*) comprises either 32, 64, 128, 256, 512, 1,024, or up to 2,048 processing elements (PEs) arranged in a three-dimensional torus. Each PE contains a DEC chip 21064 Alpha processor operating at a 150 MHz clock rate. This superscalar single chip processor has a nominal peak performance of 150 Mflops with floating-point operations being performed in 64-bit IEEE format. Thus, a T3D system has a peak performance of 4.8 Gflops (32 PEs) up to 300 Gflops (2,048 PEs).

Local memory within each PE is part of a physically distributed, logically shared memory system. Each PE can access the 8 or 32 Mbytes of local memory of another PE without involving the microprocessor in that PE. The data channels of the interprocessor network are bidirectional and independent in each of the three dimensions. The theoretical bandwidth of the network is 300 Gbytes/s for a 1024 PE T3D system. For more details the reader is referred to [6].

MasPar. A MasPar system has a SIMD architecture with from 1,024 (1K) up to 16,384 (16K) PEs, arranged in a two-dimensional mesh with toroidal wrap-around. Two types of MasPar systems are distinguished: MasPar MP-1 and MasPar MP-2 systems. In a MasPar MP-1 system, each PE is a 4-bit processor while the newer MP-2 systems contain 32-bit processors. Each processor has either 16 Kbytes or 64 Kbytes of local data memory. The theoretical peak performance of a 16K MasPar MP-1 system is 26,000 MIPS and 550 Mflops (64-bit IEEE floating point) or 1,200 Mflops (32-bit IEEE floating point). For a 16K MasPar MP-2 system, these numbers are 68,000 MIPS and 2.4 or 6.3 Gflops, respectively.

The Array Control Unit (ACU) of a MasPar system controls the PEs. The PE-array and ACU form the Data Parallel Unit (DPU). Besides the DPU, the MasPar system needs a front-end (FE) that serves as an interface to the DPU. The FE is the host for tools and compilers.

The communication networks of the MasPar MP-1 and MP-2 are exactly the same. Interprocessor communication takes place either through the Xnet or through channels controlled by the Router. The Xnet provides nearest-neighbor communication in the horizontal, vertical, and diagonal directions. The theoretical bandwidth of the Xnet equals 23 Gbytes/s for a 16K MasPar system. Global interprocessor communication is handled by the router, having a theoretical bandwidth of 1.3 Gbytes/s. For more details the reader is referred to [7].

4 Data Distributions

The data distribution or data layout within the hierarchical memory of a concurrent computer is critical in determining the performance and scalability of the parallel code. In the design of parallel DYN code one has to choose among several ways of distributing the data, in order to obtain a good load balance.

4.1 Blocked versus Scattered Data Decompositions

Two common data decompositions are the *blocked* and the *scattered* (or *cyclic*) decompositions [8, 9]. Consider a vector of length M and a multicomputer comprising N_p processors. The blocked decomposition, β_{M,N_p}, assigns contiguous entries in the global vector to the processors in blocks

$$\beta_{M,N_p}(I) \equiv (\lfloor I/m \rfloor, I \bmod m), \tag{3}$$

where $m = \lceil M/N_p \rceil$. That is, if $\beta_{M,N_p}(I) = (p,i)$, the entry in the global vector with index I is assigned to processor p where it is stored in a local vector with index i. The scattered decomposition, σ_{M,N_p}, assigns contiguous entries in the global vector to consecutive processors in the processor array

$$\sigma_{M,N_p}(I) \equiv (I \bmod N_p, \lfloor I/N_p \rfloor). \tag{4}$$

For matrix problems the blocked and scattered decompositions can be applied over rows and columns to obtain blocked and scattered matrix decompositions. Assume that the N_p processors are arranged in a two-dimensional processor array with P rows and Q columns, $N_p = PQ$. Each processor can be uniquely identified by the pair (p,q), $0 \le p < P$, $0 \le q < Q$ denoting its position on the processor grid. Then, the decomposition of a $M \times N$ matrix can be regarded as the tensor product, $\mu_{M,P}(I) \otimes \nu_{N,Q}(J)$, of two vector decompositions, $\mu_{M,P}$ and $\nu_{N,Q}$. The mapping $\mu_{M,P}$ decomposes the M rows of the matrix over the P rows of processors, and $\nu_{N,Q}$ decomposes the N columns of the matrix over the Q

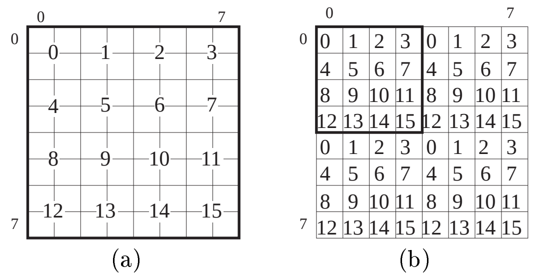


Figure 1: (a) Blocked and (b) scattered decompositions of a 8×8 matrix on a 4×4 processor array. The thick square represents one 'stamp' of the processors grid on the matrix.

columns of processors. Thus, for the blocked decomposition of a $M \times N$ matrix we have

$$\beta_{M,P}(I) \otimes \beta_{N,Q}(J) = ((p,q),(i,j)), \tag{5}$$

where $\beta_{M,P}(I) = (p,i)$ and $\beta_{N,Q}(J) = (q,j)$, see Figure 1a. Analogously, for the scattered decomposition we have

$$\sigma_{M,P}(I) \otimes \sigma_{N,Q}(J) = ((p,q),(i,j)), \tag{6}$$

where $\sigma_{M,P}(I) = (p,i)$ and $\sigma_{N,Q}(J) = (q,j)$, see Figure 1b. So, the matrix entry with global index (I,J) is assigned to the processor at position (p,q) on a $P \times Q$ processor grid, where it is stored in a local matrix with index (i,j). Informally speaking, the scattered decomposition of a vector or matrix can be described by 'layering' the vector or matrix on the processor grid. For more details the reader is referred to [9].

4.2 Matrix to Vector Reductions

As described in Section 2, the DYN routine in the HIRLAM reference code stores the three- and two-dimensional fields in two- and one-dimensional arrays by applying a dimensional reduction to the horizontal grid component. In general, a matrix can be reduced to a vector simply by applying a numbering scheme to the matrix entries. Two basic numbering schemes are the *column-major* and *row-major* schemes. More formally, the column-major numbering of the entries of a $M \times N$ matrix is defined as

$$\delta_N(I,J) \equiv I + NJ. \tag{7}$$

Thus, the (I,J) index of a $M \times N$ matrix receives number $I + NJ$. For the one-dimensional column-major blocked decomposition of a $M \times N$ matrix on a one-dimensional N_p processor array we have:

$$(\beta_{MN,N_p} \circ \delta_N)(I,J) = (p,i). \tag{8}$$

Analogously, for the one-dimensional column-major scattered decomposition we have

$$(\sigma_{MN,N_p} \circ \delta_N)(I,J) = (p,i). \tag{9}$$

So, the matrix entry with global index (I,J) is assigned to processor p, where it is stored in a local vector with index i. The row-major numbering of the entries of a matrix and the row-major matrix decompositions are defined analogously.

Standard Fortran 77 compilers adopt the column-major numbering scheme for storage of matrices, which is in accordance with the choice of dimensional reduction in the original DYN code.

5 Parallel DYN Implementations

In the previous section the data distributions were described that can be exploited for the distributed storage of the arrays containing the three- and two-dimensional fields in a parallel implementation of the DYN routine. Recall from Section 2 that due to the finite difference schemes in the DYN routine many references to nearest neighboring grid points have to be made. For sequential and vector computers this poses no problem. For distributed memory parallel computers, however, these references may become interprocessor communications with impact on performance. Furthermore, the performance of parallel codes employing different parallel programming paradigms can differ significantly. Table 1 depicts the various parallel programming paradigms employed for the parallel DYN implementations presented in the next sections.

5.1 Data Parallel and Work Sharing

The CRAY T3D and MasPar systems, among many other parallel platforms, allows data-parallel code for parallel execution. In Fortran 90 data-parallel code, array assignments are implicitly executed in parallel. Investigations of data-parallel HIRLAM code can be found in [3]. In work-sharing code the work done by loop iterations over array elements is equally shared among the available processors.

We have coded four data-parallel DYN implementations: two *one-dimensional* DYN codes employing blocked and scattered decompositions of the two-dimensional horizontal array components which are reduced to one dimension, and two *two-dimensional* DYN codes employing blocked and scattered decompositions of the two-dimensional horizontal array components. The vertical array component is *collapsed* (or *degenerated*), i.e. 'stacked' onto the processor grid.

Automatic translation of the sequential Fortran 77 DYN code to data-parallel Fortran 90 code, see Figures 2a and 2b, was provided by the Vast-II compiler [10], being part of the MasPar software. This way, both one- and two-dimensional data-parallel DYN codes were obtained. The translation resulted in data-parallel codes where each code contained 56 data-parallel array assignments for each horizontal layer of the integration area.

T3D. The blocked and scattered decompositions were obtained with the '`SHARED`' compiler directives. In this respect, it is important to stress that the current CRAY T3D systems require the non-collapsed dimensions of these distributed arrays to be integer powers of two.

The current implementation of the T3D Fortran compiler translates data-parallel code to work-sharing code [11], see Figures 2b and 2c. By specifying for each loop which array elements reside on what processor in the work-sharing

	global address space	local address space
implicit communication	*data parallel and work sharing*	
explicit communication	*shared memory put/get*	*message passing*
preceding/deferred communication		*sub-domain splitting*

Table 1: Parallel programming paradigms.

```
      DO 10 J=2,N-2
         DO 20 I=2,M-2
            B(I,J)=(A(I+1,J)-A(I,J))
   +              *(A(I,J+1)-A(I,J))
20       CONTINUE
10    CONTINUE

(a)
```

```
   B(2:M-2,2:N-2)=(A(3:M-1,2:N-2)
+                -A(2:M-2,2:M-2))
+                *(A(2:M-2,3:N-1)
+                -A(2:N-2,2:N-2))




(b)
```

```
CDIR$  DOSHARED (J,I) ON B(I,J)
       DO J=2,N-2
          DO I=2,M-2
             B(I,J)=(A(I+1,J)-A(I,J))
     +              *(A(I,J+1)-A(I,J))
          END DO
       END DO
(c)
```

Figure 2: Examples of (a) sequential code, (b) data-parallel code, and (c) work-sharing code.

code, the loop iterations over array elements will be equally shared among the available processors. Implicit barrier-synchronization points are placed at the end of the work-sharing loop constructs if the data dependency analysis of the compiler indicates this necessity. However, in order to obtain maximum efficiency on the T3D system, all implicit barriers were removed and explicit barriers were added at 8 synchronization points in the data-parallel DYN code.

MasPar. Arrays stored on the DPU are distributed over the PE-array with the default scattered decomposition. To obtain blocked decompositions for all distributed arrays, the '`-block`' compiler option was used. In addition, the proper alignment of arrays on the two-dimensional PE-array with respect to their use was controlled with '`MAP`' compiler directives. For the one-dimensional DYN code, vectors are row-by-row mapped on the two-dimensional PE-array.

From a portability and maintenance point of view, the data-parallel implementation of the DYN routine is the most preferred alternative of all parallel programming paradigms. The data-parallel code can be easily obtained from the original DYN code with the aid of the Vast-II compiler. The code can be run on both MasPar and CRAY T3D systems without many modifications.

5.2 Sub-Domain Splitting

Another attractive method for the parallelization of the DYN routine is provided by a domain decomposition approach, see also [4]. Informally speaking, the nature of the model provided by DYN allows a decomposition of the global model into local sub-models which can be solved separately on parallel architectures. The local models can be distributed among the available processors such that each processor executes the sequential DYN routine in parallel. Distributing the local models can only be accomplished if the data of the horizontal component of the global three- and two-dimensional fields are decomposed using the two-dimensional blocked decomposition. As described in Section 2, computation of the explicit tendencies of the prognostic variables on a grid point involves references to data at the neighboring grid points. As a result, the sub-grids of the local models have mutually overlapping zones, see Figure 3. The size of the overlap at the upper and left borders and the size of the overlap at the lower and right borders are one and two grid points, respectively. Therefore, application of sub-domain splitting in DYN involves the storage of additional data in the overlap zones that has to be communicated between the processors in advance. This way, the domain decomposition approach 'pushes' all communications to the outside of the routine and no synchronization of the processors is required within the routine. In principle, sub-domain splitting of DYN provides portable code for MIMD platforms. No modifications within the original DYN code are necessary, which makes maintenance of the parallel DYN code easy. However, this method is not very suitable for MasPar systems, since overlapping zones are difficult to express in data parallelism. For this reason we have coded sub-domain splitting on the CRAY T3D only.

5.3 Adding Explicit Message Passing

In the previous section, the domain decomposition approach was used to obtain a parallel DYN code by combining the execution of sequential DYN code on separate processors in parallel. This way, a coarse-grain parallel implementation was obtained with all communications 'pushed' to the outside of the DYN routine. In this section two similar but fine-grain parallel implementations of DYN will be presented. Again, the two-dimensional blocked decomposition of the horizontal components of the three- and two-dimensional fields is applied. Furthermore, each block is extended by a 'ring' measuring one grid point in all four directions. The rings buffer the data at nearest neighboring grid points that reside on the four logically adjacent processors. Explicit communications are required to exchange the updated data at grid points that reside on adjacent processors.

We have implemented two parallel DYN codes on the CRAY T3D system requiring explicit communications. One code utilizes *synchronous*, the other *asynchronous* communications:

- **Synchronous.** Parallel Virtual Machine (PVM) message passing.
- **Asynchronous.** Shared memory get using the T3D 'shared memory management' functions.

Shared memory functions, part of the CRAY T3D Fortran 77 libraries, perform remote 'put' and 'get' operations on the local memory of remote processors. No handshaking mechanism is required. Although a 'shared memory put' operation

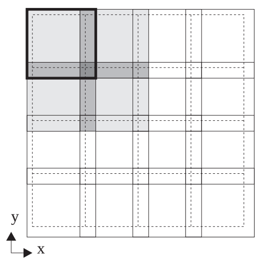
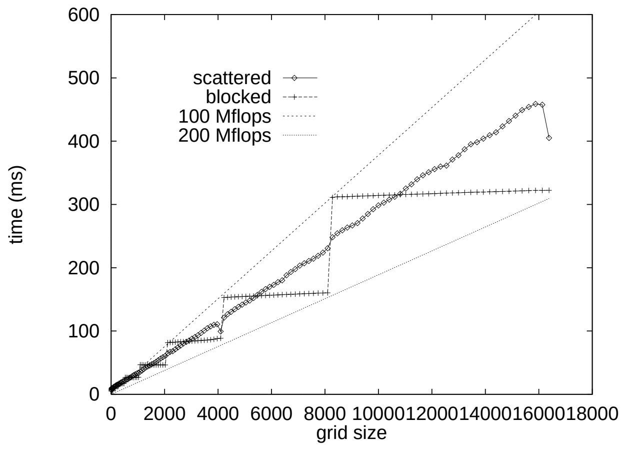
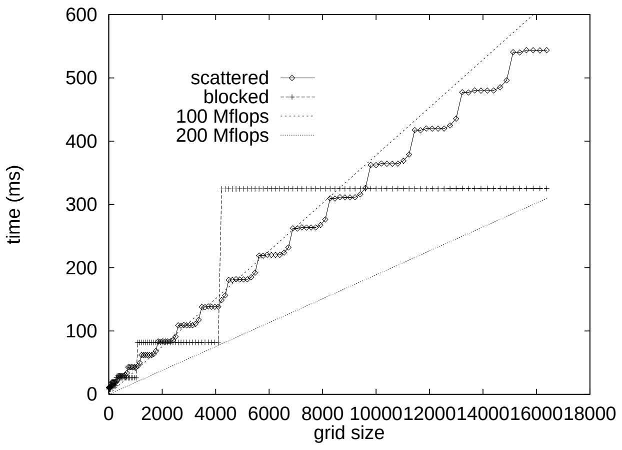


Figure 3: Decomposition of the horizontal grid into mutually overlapping sub-grids. Each sub-grid is assigned to one of the 4 × 4 processors.



(a)



(b)

Figure 4: (a) Total elapsed time of the one-dimensional and (b) two-dimensional work-sharing DYN codes executed with 64 PEs on the CRAY T3D.

is faster than a 'get', the 'put' operation requires the receiving processor to issue a data cache invalidate. Since this is too expensive in general, asynchronous communication is implemented using 'shared memory get' function calls only. For both codes, 9 communication phases are needed to update the data in each horizontal layer. In each phase several sends are issued resulting in a total of 26 data exchanges per layer.

From a portability point of view, the parallel DYN code with PVM communications should be preferred since the code with asynchronous communications adopts the T3D-specific 'shared memory get' function calls. On the MasPar system, however, no explicit message passing can be coded in Fortran.

6 Performance Results

In this section the performance of the various parallel DYN codes, described in the previous section, are compared on CRAY T3D and MasPar systems.

The CRAY T3D system used for the investigation in this paper is a T3D system with 32 Mbytes of local memory for each PE. The cf77 release 6.0.2 Fortran compiling system was used. The MasPar system used for the investigation in this paper is a 1K MasPar MP-1 system (32×32 mesh) with a DEC 5000/240 front-end. All tests were performed with system release 3.2.0 of the MasPar software, which included Vast-II 3.06 and the Fortran 90 compiler MpFortran 2.2.7.

The elapsed time presented in all figures was determined by taking the average of ten execution times of one call to DYN, using the '`irtc`' intrinsic function on the CRAY T3D system and using the '`mpTimerStart`' and '`mpTimerElapsed`' intrinsic functions on the MasPar system. The standard deviation turned out to be about 1% on both CRAY T3D and MasPar systems.

6.1 Data Parallel and Work Sharing Results

In this section the performance of the data-parallel DYN codes, described in Section 5.1, on CRAY T3D and MasPar systems will be presented.

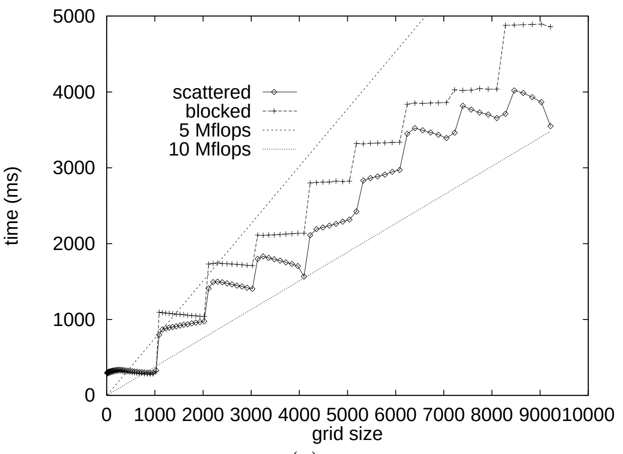
T3D. Unfortunately, the one-dimensional data-parallel DYN code with blocked data decomposition computed incorrect results on the CRAY T3D and generated run-time floating point exceptions with scattered data decomposition, rendering the one-dimensional data-parallel codes useless. For this reason, it was decided to translate data-parallel code to work-sharing code manually and use the work-sharing codes on the T3D system instead. The array alignment was chosen to be on the 'assigned' array in the work-sharing code.

Figure 4 depicts the elapsed time of the one- and two-dimensional work-sharing codes. The timings are given as a function of the $M \times N$ grid size in the horizontal, $M = N$, with 16 levels. Iso-100 and iso-200 Mflops curves are included in the figures, denoting the elapsed time of optimally scalable 100 or 200 Mflops (parallel) machines executing DYN. Here, the number of floating point operations executed by the two-dimensional DYN routine on a square $M \times N$ horizontal integration area, $M = N$, with 16 levels, is given by

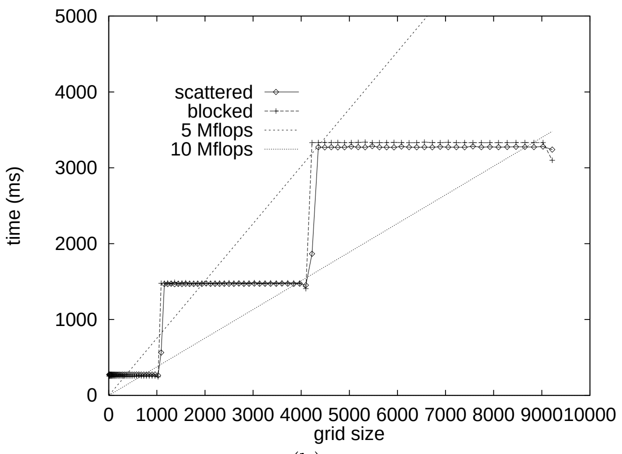
$$\#\text{floating point operations} = \\ 77.6 \cdot 10^3 - 13.2 \cdot 10^3 \cdot M + 3.78 \cdot 10^3 \cdot M^2. \quad (10)$$

This equation was derived using the CRAY T3D 'Apprentice' performance tool which reports the number of floating point operations executed in one run of the DYN routine. The average number of floating point operations of DYN executed on several different grid sizes was used to derive equation (10). This equation is independent of the number of processors.

From Figures 4a and 4b the following observations can be made. Firstly, the one-dimensional code is better scalable with respect to the grid size. The reason is that the reduction of the two-dimensional horizontal integration area to a one-dimensional area improves the load balance using $N_p = PQ$ processors if $\lceil M/P \rceil \cdot \lceil N/Q \rceil > \lceil MN/N_p \rceil$. This is satisfied for at least half of all grid sizes. Secondly, observe that the blocked one- and two-dimensional work-sharing DYN codes have peak performances of around 200 Mflops. The requirement that array dimensions should be powers of two has a big impact on the blocked distribution, as can be seen by the large staircase-like steps. These steps are a result of the improper load balance that oc-



(a)



(b)

Figure 5: (a) Total elapsed time of the one-dimensional and (b) two-dimensional data-parallel DYN codes executed on the 1K MasPar MP-1 system.

curs when the horizontal grid cannot be perfectly mapped on the processor array. Thirdly, both scattered one- and two-dimensional data-parallel codes have a lower peak performance than the blocked codes. This is due to the fact that for the codes with scattered data decompositions more communications are required: adjacent grid points always reside on different processors. The average performance of the scattered codes is 100–150 Mflops. Note that the one-dimensional scattered code has a somewhat higher performance and is well scalable. The scalability can be explained by the fact that for larger grids an increase of the horizontal grid results in the 'stacking' of one or more grid layers on each processor.

MasPar. On the MasPar system 32-bit floating-point arithmetic was used which suffices for the arithmetic performed in DYN. Figure 5 depicts the elapsed time of the one- and two-dimensional data-parallel DYN codes. Iso-5 and iso-10 Mflops curves are included in the figures.

The following observations can be made from Figures 5a and 5b. Firstly, the one-dimensional code is better scalable with respect to the grid size. Secondly, observe that the blocked one- and two-dimensional data-parallel DYN codes have a peak performance of around 10 Mflops on the 1K MasPar MP-1. The scattered one- and two-dimensional data-parallel codes have a somewhat higher performance with exception of the two-dimensional code at grid sizes measuring 32 × 32, 64 × 64, and 96 × 96, respectively. The lower performance is due to less efficient communications with run-time library calls for the codes with the blocked data distributions. As a result the amount of data exchanged is lower, while the communication costs are higher. Thirdly, observe that the one-dimensional code show irregular time curves. The reason is that processor communication in the one-dimensional code is mainly diagonal over the two-dimensional processor grid, because a reference to an adjacent grid point may require communication with a non-nearest neighbor processor. This is in contrast to the two-dimensional code where an adjacent grid point is always located on the same or a neighbor processor.

In comparing CRAY T3D and MasPar results, the following observations can be made. The general expectation that blocked decompositions require less communication and hence, less time, is true only for the CRAY T3D system. The codes with blocked data decompositions and specific grid sizes give the highest performance, while the code with one-dimensional scattered data decomposition gives a better average performance.

6.2 Sub-Domain Splitting and Explicit Message Passing Results

In this section the performance results of the parallel DYN codes obtained by sub-domain splitting and explicit message passing will be presented.

Theoretically, the parallel code that results from sub-domain splitting in DYN as described in Section 5.2, is less scalable compared to the data-parallel codes. This is due to the fact that each local model can only be solved separately if the local grid is extended with mutually overlapping zones. The additional computational overhead involved becomes apparent when the size of the overlapping zone is relatively large compared to the size of the local grid. For such problems, the scalability of the algorithm decreases with increasing number of processors N_p as follows. Consider a horizontal integration area that comprises $M \times N$ grid points. Let n, s, e, and w denote the size of the north, south, east, and west overlapping zones, respectively, and let P and Q denote the number of processors in the longitudinal and lateral directions, respectively, such that $N_p = PQ$. A lower bound for the efficiency of the parallel DYN code obtained by splitting the horizontal grid into sub-domains, is

$$\text{efficiency} = \frac{\text{sequential time}}{N_p \cdot \text{parallel time}} \approx \frac{\text{size of global grid}}{N_p \cdot \text{size of local grid}} \approx \frac{MN}{N_p \cdot \left(\left\lceil \frac{M-n-s}{P} \right\rceil + n + s\right) \cdot \left(\left\lceil \frac{N-e-w}{Q} \right\rceil + e + w\right)}, \quad (11)$$

since the sequential time is proportional to the grid size. Recall that an algorithm is perfectly scalable if the efficiency depends only on the grain size, $g = MN/N_p$, and not independently on M, N, and N_p. Only when $g \gg 1$, the scalability of the algorithm approaches linearity. For the DYN

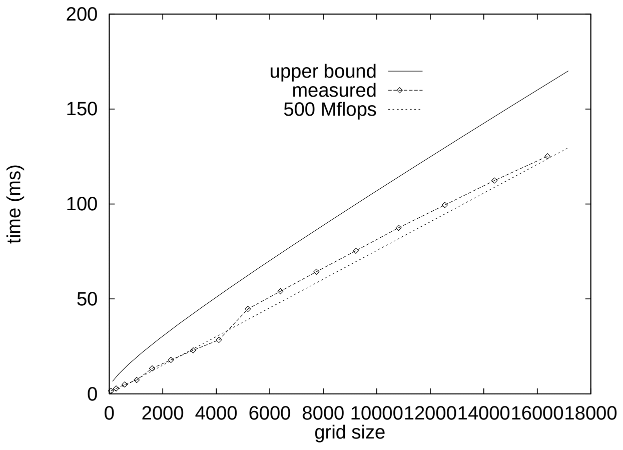


Figure 6: Theoretical upper bound and measured total elapsed time of the parallel sub-domain DYN code executed with 64 PEs on the CRAY T3D.

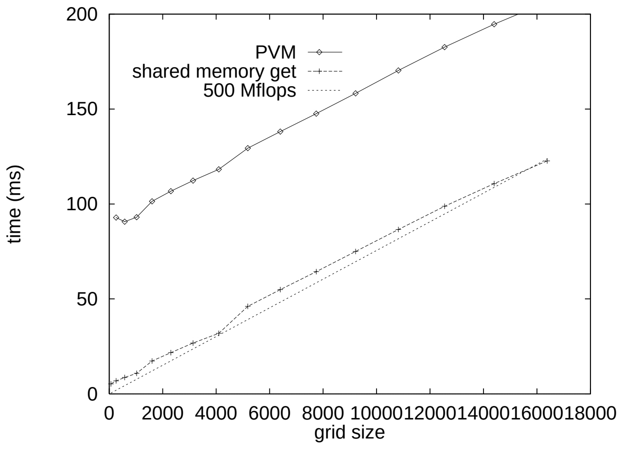


Figure 7: Total elapsed time of parallel DYN using PVM message passing and 'shared memory get' communications executed with 64 PEs on the CRAY T3D.

routine the size of the overlapping zones in the four directions are $n = 1$, $s = 2$, $e = 2$, and $w = 1$, respectively. For example, a theoretical efficiency of at least 50% corresponds to a grain size of $g > 36$, thus $MN > 2304$ for $N_p = 64$.

The performance results of the sub-domain parallel code on the CRAY T3D are shown in Figure 6. We have included a theoretical upper bound on the total elapsed time which corresponds to the efficiency curve of a 500 Mflops parallel machine with $N_p = 64$. This theoretical upper bound is based on equation (11).

The following observations can be made from Figure 6. Firstly, the peak performance of the parallel sub-domain DYN code is around 500 Mflops. Secondly, observe that the actual performance of the parallel code is higher than the theoretical performance based on equation (11). One reason for this difference can be explained by the fact that not all computations in DYN use all of the data stored in the overlap zones. The second reason is that the small data cache of the T3D has a higher hit ratio when the sub-domain DYN code executes with small integration areas.

The performance results of the parallel DYN codes with explicit message passing on the CRAY T3D are shown in Figure 7, from which the following observations can be made. Firstly, the peak performance of the parallel DYN code with 'shared memory get' communications is around 500 Mflops. Secondly, the 'shared memory get' communications are significantly faster than the PVM communications. In this respect the CRAY T3D 'Apprentice' performance tool reported that 25 ms of the total execution time was spent on sending data and 62 ms on receiving data with PVM message passing. These measures were independent of the grid size. Thus it can be concluded that PVM on the T3D results in a significant overhead. This in contrast to 'shared memory management' functions that utilize the hardware capabilities of the T3D directly with negligible overhead.

Finally, it can be concluded from Figures 6 and 7 that a peak performance of 500 Mflops is obtained for both the parallel sub-domain DYN code and the 'shared memory get' DYN code on the CRAY T3D with 64 PEs.

6.3 CRAY T3D Performance Comparison

The performance results of the various parallel DYN codes on the CRAY T3D with 4, 16, and 64 PEs and two different grid sizes are shown in Table 2. In the table the results of the parallel codes employing the blocked two-dimensional data decomposition are shown.

From Table 2 the following observations can be made. Firstly, the codes are well scalable with respect to the number of processors for both grid sizes. Secondly, the hand-written work-sharing DYN code is up to 33% faster than the data-parallel DYN code. Hence, the translation of the data-parallel to work-sharing code by the CRAY T3D compiler is sub-optimal. Thirdly, the 'shared memory get' code is about 2.5 times faster than the work-sharing code. The CRAY T3D 'Apprentice' performance tool reported that the number of instructions generated by the compiler that is executed in the work-sharing code per floating point operation is equal to 25 from which 6 are memory load/stores; for the 'shared memory get' code this is equal to only 2.2 instructions per floating point operation from which 1.2 are memory load/stores. This indicates that the compiler is currently not able to generate code that is equally efficient for both programming styles. Besides, it is worthwhile to mention that the data cache is invalidated if data-parallel or work-sharing codes are executed. However, in general, the 8 Kbytes data cache of the T3D is too small especially for vector-based computations. Therefore, future research on this subject is required to maximize cache reuse. Fourthly, the highest performance was obtained for the 'shared memory get' and sub-domain codes. Note that the performance of the sub-domain code is slightly higher when the local integration area is small.

As a final remark we want to mention that this and other investigations show that the single-node performance of the CRAY T3D is disappointing. For example in [12] a performance of only 12 Mflops per PE was reported for a parallel implementation of a weather forecast model on the CRAY T3D. This is only 8.0% of the peak performance which is somewhat higher than the performance obtained for the most efficient parallel DYN routine in this investigation being only 5.2% of the peak performance.

	64 × 64 × 16 grid			128 × 128 × 16 grid		
	4 PEs	16 PEs	64 PEs	4 PEs	16 PEs	64 PEs
shared memory get	500	121	33	2125	507	123
sub-domain splitting	538	132	29	2131	524	125
PVM	559	205	119	2241	600	207
work sharing	1356	333	82	5679	1423	325
data parallel	1594	412	109	6644	1665	413

Table 2: Total elapsed time (ms) of parallel DYN codes on the CRAY T3D.

7 Conclusions

To conclude, the main results of this investigation are:

- The data-parallel DYN code can be easily obtained by automatic translation from the sequential DYN code and runs on both CRAY T3D and MasPar systems. However, in general, this translation could be more difficult especially for codes exploiting dirty Fortran tricks. The data-parallel code is highly portable compared to the other parallel programming paradigms.
- The performance of the data-parallel and work-sharing codes on the CRAY T3D are significantly lower (2.5–3 times) than can be obtained by employing other parallel paradigms. This shows a clear tradeoff between portability and efficiency.
- For data-parallel code it has been shown that the two-dimensional blocked data decomposition results in the highest performance on CRAY T3D and MasPar systems if the grid can be perfectly mapped on the processor array. In case of an imperfectly mapping the best average performance can be obtained with the scattered one-dimensional data decomposition.
- All codes are well scalable on the CRAY T3D with respect to both grid size and number of processors, except the data-parallel and work-sharing codes with blocked data decompositions. The exception is due to the constraint that dimensions of distributed arrays should be powers of two.
- PVM message passing is portable, but inefficient on CRAY T3D systems. The shared memory put/get communications with the CRAY T3D shared memory functions are not portable but efficient and fast.
- The highest performance of the DYN routine with 64 processors on the CRAY T3D is equal to 500 Mflops. The highest performance was obtained with the parallel DYN codes employing the sub-domain or shared memory put/get parallel programming paradigms.
- The highest performance of the data-parallel DYN routine on the 1K MasPar MP-1 system equals 10 Mflops.

To summarize, the first main conclusion is that no optimal data distribution exists that for all grid sizes results in the best performance of the parallel code. The second main conclusion is that an efficient use of the CRAY T3D requires the inclusion of explicit message passing primitives by the programmer. Unfortunately, the CRAY T3D Fortran compiler is not able to generate efficient code for data-parallel and work-sharing implementations. Finally, despite the fact that explicit finite difference codes are considered easily parallelizable, this investigation demonstrates that many issues have to be considered to obtain optimal parallel code.

Acknowledgements

We would like to thank Gerard Cats of the Royal Netherlands Meteorological Institute (KNMI) for many valuable discussions. This project was sponsored by the National Computing Facilities Foundation (NCF) for the use of supercomputing facilities, with financial support from the Netherlands Organization for Scientific Research (NWO).

References

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

[2] L. Wolters, G. Cats, and N. Gustafsson, *Limited Area Numerical Weather Forecasting on a Massively Parallel Computer*, in proceedings of the 8th ACM International Conference on Supercomputing, July 11–15 1994, Manchester, England, ACM Press, pp. 289–296.

[3] L. Wolters, G. Cats, and N. Gustafsson, *Data-parallel Numerical Weather Forecasting*, accepted for publication in a special issue of Scientific Programming on 'Application Performance Analysis'.

[4] T. Kauranne, *The Operational Hirlam 2 Model on Parallel Computers*, to appear in proceedings of the Sixth ECMWF Workshop on the use of Parallel Processors in Meteorology, ECMWF, Reading, UK, November 1994.

[5] G.J. Haltiner and R.T. Williams, *Numerical Prediction and Dynamic Meteorology, second edition*, John Wiley & Sons, New York, 1980.

[6] CRAY T3D, *CRAY T3D System Architecture Overview*, Cray Research, Inc., March 1993.

[7] MasPar, *MasPar MP-1 Hardware Manuals*, MasPar Computer Corporation, July 1992.

[8] E. van de Velde, *Data Redistribution and Concurrency*, Parallel Computing, 16, December, 1990, pp. 125–138.

[9] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. DuCroz, A. Greenbaum, S. Hammerling, A. McKenny, and D. Sorensen, *Lapack: A portable linear algebra library for high-performance computers*, in proceedings of Supercomputing '90, 1990, IEEE Press, pp. 1–10.

[10] Vast-II, *MasPar Vast-2 User Guide*, MasPar Computer Corporation, July 1992.

[11] CRAY T3D, *CRAY T3D Applications Programming*, Cray Research, Inc., June 1994.

[12] D. Dent, L. Isaksen, G. Mozdzynski, M. O'Keefe, G. Robinson, F. Wollenweber, *IFS Model: Performance Measurements*, to appear in proceedings of the Sixth ECMWF Workshop on the use of Parallel Processors in Meteorology, ECMWF, Reading, UK, November 1994.