Centrum voor Wiskunde en Informatica
REPORT*RAPPORT*

An evaluation of the Cray T3D programming paradigms in atmospheric chemistry/transport models

J.G. Blom, Ch. Kessler and J.G. Verwer

Department of Numerical Mathematics

NM-R9604 March 30, 1996

Report NM-R9604
ISSN 0169-0388

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

An Evaluation of the Cray T3D Programming Paradigms in Atmospheric Chemistry/Transport Models

J.G. Blom
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

Ch. Kessler
Institut fuer Geophysik und Meteorologie
Aachener Strasse 201 - 209, 50931 Koeln, Germany

J.G. Verwer
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

Abstract

In this paper we compare the different programming paradigms available on the Cray T3D for the implementation of a 3D prototype of an Atmospheric Chemistry/Transport Model. We discuss the amount of work needed to convert existing codes to the T3D and the portability of the resulting codes. Tests show that the scalability with respect to the model size and the number of processors is linear except for the data parallel implementation.

AMS Subject Classification (1991): Primary: 65Y05. Secondary: 92-04, 65M20.
CR Subject Classification (1991): G.4, J.2, G.1.8.
Keywords & Phrases: parallel computation, distributed memory, Cray T3D, programming paradigms, long range transport air pollution models, time-dependent advection-diffusion-reaction, method of lines.
Note: This work was sponsored by the Stichting Nationale Computerfaciliteiten (National Computing Facilities foundation, NCF) for the use of supercomputer facilities, with financial support from the Nederlandse Organisatie voor Wetenschappelijk Onderzoek (Netherlands Organization for Scientific Research, NWO). Part of this work belongs to the TASC project 'HPCN for Environmental Applications' which is sponsored by the Stichting HPCN with financial support from the Ministry of Economic Affairs.

1. Introduction

The mathematical description of atmospheric transport, chemistry and exchange of trace constituents gives rise to a system of time-dependent partial differential equations (PDEs) of the advection-diffusion-reaction type

$$c_t + \nabla \cdot (\mathbf{u}c) = \nabla \cdot (K \cdot \nabla c) + R + S. \tag{1.1}$$

This system of PDEs describes the evolution in time t and space (x, y, z) of the concentration vector $c(t, x, y, z)$ of all chemical species in the model. The transport is modeled by advection in the windfield $\mathbf{u}(t, x, y, z)$ and by (turbulent) diffusion represented by the parameter $K(t, x, y, z)$. The chemistry is modeled in the chemical reaction term $R(t, x, y, z)$. Source and sink terms are represented by $S(t, x, y, z)$. To simulate with such a model real-life atmospheric chemistry/transport processes on a temporal and spatial scale of interest, first of all efficient and robust numerical algorithms are needed. Computer capacity, however, is also a critical factor, both with respect to computer power

and with respect to memory. This induces an unceasing quest for efficient implementations on the most advanced computer systems like vector/parallel supercomputers and massively parallel distributed-memory systems.

In this paper we compare the different programming paradigms available on the Cray T3D for the implementation of a 3D prototype of (1.1) with respect to their efficiency, both for the computer and for human beings. An important question in this respect is whether a code designed for a shared-memory supercomputer needs to be fundamentally restructured to accommodate a distributed-memory parallel architecture. Another issue is to what extent a code, written in (a dialect of) HPF (High Performance Fortran), can be efficient on distributed-memory parallel systems, on vector/parallel supercomputers, and on (a cluster of) workstations.

We consider three different programming styles to implement the 3D model on a Cray T3D:

- Data parallel: distribute the data with directives and use array syntax for the computations on the data. Array syntax constructs are automatically distributed over the processors.
- Data and work sharing: distribute the data and the loops that contain the computations on the data using directives.
- Domain decomposition: each processor does the computations on a subdomain using data residing on that processor. Explicit message passing is used to account for the boundary conditions between the subdomains.

As starting point for the implementations we used codes in Fortran 77 and Fortran 90 developed for the shared-memory vector/parallel architecture Cray C90. We will comment on the changes needed to convert the C90 codes to the T3D. The resulting codes are portable except for the main program in the domain decomposition implementation. The scalability with respect to the model size and the number of processors is linear for the second and third variant. For the data parallel implementation the code does not scale linearly due to deficiencies in the compiler.

In Section 2 we will give a short survey of the numerical algorithms used and their consequences for the implementation. In Section 3 we discuss the implementations themselves, the amount of work needed to convert the C90 code into a T3D code, and the portability of the resulting programs. Section 4 contains the performance results for the different implementations, Section 5 an evaluation of the scalability both with respect to the number of processors and with respect to the model size. In this section we also discuss future expectations and comparisons to other platforms like vector/parallel computers.

2. The Model

For our comparison we use the same 3D prototype of (1.1) on the sphere as in [8, 5] (see [8] for a complete description). The test problem contains horizontal advection, vertical diffusion and the chemical reaction model EMEP[7] (66 species, 140 reactions). The domain is defined by an horizontal area of 7.5^o square and a vertical height of $2000m$. The computational domain is given by a uniform longitude-latitude grid in the horizontal directions and a non-equidistant grid in the vertical.

The numerical solution method for (1.1) is derived along the method of lines. The horizontal advection operator is discretized by the mass-conservative, flux-limited finite-difference scheme proposed in [4], based on 3rd-order upwind-biased discretization. The resulting 9-point stencil has 5 grid points along both the longitudinal and the latitudinal lines. The vertical diffusion term is discretized on a non-equidistant cell-centered grid resulting in a 3-point coupling in the vertical. The time integration is performed by an IMEX (IMplicit-EXplicit) scheme, based on the 2nd-order backward differentiation formula, which handles advection explicitly and chemistry and vertical diffusion implicitly and coupled. The resulting nonlinear systems are solved with Gauss-Seidel iteration. The tridiagonal linear

systems that result from the diffusion term are solved directly within the Gauss-Seidel process (cf. [8]).

We consider this 3D prototype a good approximation of a full-scale implementation of (1.1) in the sense that adding diffusion or vertical advection will not influence the trend of our findings as long as the horizontal processes will be calculated explicitly. On the other hand a real model will contain much more I/O operations like, e.g., reading of and dealing with meteo and emission data and logging concentration vectors at specific times or even directly visualizing the solution values. An evaluation of the performance of that part of a simulation, however, can better be done on the successor of the T3D, the T3E, since that machine also has a scalable I/O architecture.

3. Implementation

The original code was developed for the shared-memory vector/parallel Cray C90 architecture (cf. [1, 8]). One of the goals then was to compare the efficiency and ease of use of Fortran 77 versus Fortran 90 for a full-scale implementation of (1.1). As data structure for the concentration vector c in (1.1) at a specific time a four-dimensional array containing the space indices and the species index was chosen. Parallelism was obtained purely by distributing the work (loops) over the processors. On a distributed-memory machine data on which a processor operates should as much as possible reside in the local memory of that processor. Therefore, the data arrays for the concentration vectors should be distributed over the local memories. Of interest then for a code on a parallel distributed-memory system is the coupling which exists between the data that is distributed over the processors. In equation (1.1) the transport and diffusion part gives rise to a coupling in space and the chemical reaction term to a coupling across the chemical species. If one uses operator splitting or an IMEX time-integration scheme it is customary to compute the chemistry part implicitly, possibly coupled with the vertical diffusion. The other parts of system (1.1) are in general integrated explicitly in time. Because the implicit computation of the chemistry coupled with the vertical diffusion is by far the most expensive part of a time step we choose in principle to distribute the data such that the implicit part of the time step will be computed on local data, i.e., the horizontal domain will be partitioned and the vertical and species indices of a specific grid point in the horizontal domain all reside on 1 processor (each local memory contains entire vertical columns and entire species vectors for a number of points in the horizontal plane). This means that for the computation of the advection in each direction the concentration values of the 2 nearest neighbors at an internal boundary should be exchanged. The flux over 2 of the 4 boundaries should also be communicated to the neighboring subdomain. However, we will also show the effect of using this data distribution only for the chemistry and vertical diffusion part and using for the rest of the computations a partitioning of the vertical (each local memory contains entire horizontal planes and entire species vectors for a number of points in the vertical direction) with (implicit) redistribution of the data in between. Using this data mapping no further communication is required.

The implicit assumption in the above is that the model is implemented on the entire domain in the *data parallel* and *data and work sharing* programming style (the SIMD or SPMD approach). A second way of looking at an implementation of the model is the *domain decomposition* approach. Here the physical domain of the *global* model is decomposed in subdomains that are distributed over the processors. On each processor a *local* model is computed, with, if explicit time integration is used, in every time step known boundary conditions between the subdomains. Explicit message passing is used to take these boundary conditions into account.

3.1 Programming Paradigms

The Cray T3D offers two different programming styles in Fortran. The first is the data and work sharing model Craft (Cray Research Adaptive ForTran). Using directives the data is distributed over the processing elements (PEs). Array syntax constructs are automatically distributed over the PEs,

such that the work is done on the PE on which most of the data resides. If the computation on the distributed data is programmed using loops, the programmer has to add directives to distribute the work. With the Craft programming model the (safe) access to non-local memory is organized by the compiler and the run-time system. The second programming paradigm, and more suitable for the domain decomposition approach, is the use of local or `PRIVATE` data in combination with message passing. This message passing can be done using the 'standards' PVM or MPI, or using the Cray explicit shared memory programming style (SHMEM). The latter is an order of 10 faster according to a comparison of message passing systems on the T3D[6]. In this case the programmer has to take care of the synchronization and data coherency in memory and in cache.

For a better understanding of the choices made in the implementation we first make some technical remarks about the architecture and the programming paradigms of the Cray T3D. For a more detailed description we refer to the T3D documentation and to [5, 3].

The Cray T3D (**T**orus **3** **D**imensional interconnecting topology) is a massively parallel machine with physically distributed but logically shared memory. A machine can contain a maximum number of 2048 processor elements (PE). The central processing units (CPUs) used in the T3D are 64-bit DEC 21064 processors with a theoretical peak performance of 150 Mflop/s. The clock frequency is 150 MHz. The processor has a separate instruction and data cache of 8 kbyte each. Each PE of the machine has local memory which can be accessed by all other PEs across the communication network. The term logically shared memory denotes the possibility to address memory on a remote PE with normal Fortran array-indexing commands. To achieve a high degree of parallelism for other than the so called 'embarrassingly parallel programs' the interconnecting network has a high bandwidth: the data transfer rate is 300 Mbyte/s in each of the 6 directions. All processors can be programmed individually (MIMD architecture).

The memory hierarchy on the T3D is not very austere for a distributed memory machine. Roughly speaking one can say that obtaining data from cache is approximately 10 times as fast as from local memory, which in turn is approximately 10 times as fast as remote memory (the distance of the communicating processors is not of influence on the access time). However, the (mis)use of the cache often results in a performance that is not as good as one would expect. Since the cache policy is direct mapping, it is easy to get cache thrashing, especially with `SHARED` arrays which have obligatory dimensions that are a factor of 2 (remember that the cache size is 1024 words). Because of a different address calculation loads and stores of `PRIVATE` data from local memory are cheaper than loads and stores of `SHARED` data from local memory.

In the following we describe the three different programming styles and the resulting implementations of the 3D model that we will evaluate in Section 4 and 5. The first two are based on the T3D Craft model, the third is the message passing variant. Our policy in the implementations using Craft were to add as much as possible optimizing compiler directives in the module *headers*, to limit the changes *within* the code to a minimum, and to keep the code portable (of course the parallelizing directives are machine-dependent).

We did not use `SHARED` to `PRIVATE` coercion as we did in [5]. The main reason for this is the fact that the code will be no longer portable, but we will discuss in Section 4, whether coercion would have lead to a significant better performance.

In contrast to the previous version of the compiler (see [5] for details) padding by hand of `SHARED` arrays is no longer necessary, since there is an automatic 16 word padding. Also loops are now unrolled by the compiler, although perhaps not in the most optimal way. However, adding directives by hand and testing the result for a complete code would take too much time without improving the performance significantly.

Since integer divisions are done in software one should replace these wherever possible. This is currently not done by the compiler.

The use of 'weighted' distributions of shared arrays will no longer be supported in the future and therefore it is not included in the tests whether a different distribution from the default `[:BLOCK,:BLOCK]` gives a significant better performance or not.

The directives we did use for optimization are:

- `PE_RESIDENT`. Ensures the compiler that data access is local; cache will be used, but local data access still uses global indices.
- Trusted `SHARED` array arguments where possible, so that it is not necessary to check whether data redistribution is needed.
- `NO BARRIER` inserted where allowed after `DOSHARED` loops or array-syntax statements. The compiler places an implicit barrier after each array syntax statement or `DOSHARED` loop, unless it is considered save to omit this. We expect that a future release of the compiler will reorder loops and remove implicit barriers more often than is done now, but for the moment it was necessary to do it by hand to prevent a lot of unneeded synchronizations.

3.1.1 Data parallel. Data parallel programming means that the data is distributed with directives and that array syntax is used for the computations on the data. Under the Cray T3D Craft model array syntax constructs are automatically distributed over the PEs. The implementation was based on the Fortran 90 version of the code. Unfortunately it was not possible to use the Fortran 90 code itself, since in the CF90 compiler currently available on the T3D (Version 0.1.2.2) the implementation of a great deal of essential Fortran 90 features like `MODULES`, etc. is deferred.

Converting a program where the computations are written in array syntax (and in Fortran 90 intrinsics) boils down to adding the data distribution directives in the header of the program and the subroutine and function headers. However, due to compiler deficiencies, we had to split the module that solves the chemistry coupled with the vertical diffusion into 11 parts. We also added `NO BARRIER` directives to remove some implicit barriers.

In the data parallel programming style it is also extremely easy to change the data distribution between the computations. So in this programming model we implemented two different data mappings:

1. Partitioning of the horizontal domain. With respect to the communication this results for each processor in the exchange of 8 data arrays (vertical and species dimension) before the flux computation and a 'send' and a 'receive' of 2 data arrays (flux) before the computation of the horizontal advection.
2. Partitioning of the horizontal domain (as in 1) during the computation of the chemistry coupled with the vertical diffusion and partitioning of the vertical direction during the rest of the program. This means that all data needed in the computations is local, but it implies also that a redistribution of the concentration arrays has to take place before and after the subroutine that computes the chemistry coupled with the vertical diffusion.

Note that the only differences between these two programs are different `SHARED` directives in all the subroutine headers except the ones that compute the chemistry / vertical diffusion, and the removal of the 'trusted' specification of the array arguments in the enveloping chemistry routine.

3.1.2 Data and work sharing. The use of this programming model implies distribution of the data and the loops that contain the computations on the data using directives. Our starting point for this implementation was the Fortran 77 code. Besides the data distribution directives and the optimization directives described above, we needed to adapt the routine that computes the norm of a (`SHARED`) vector.

3.1.3 Domain decomposition. In this approach each PE does the computations for a local model on `PRIVATE` data. Explicit message passing is used to communicate the necessary concentration and flux values across the internal boundaries. A 'master' program takes care of the message passing, the synchronization, and the data coherency.

For this implementation we used the original Fortran 77 code. All 'workhorses' in the code can be executed on each PE using `PRIVATE` data. The program flow implemented in the calling program had to be interspersed with calls to SHMEM routines to communicate the internal boundary data. We used the non-portable SHMEM routines instead of the PVM or MPI alternatives since according to the information in [6] the latter are approximately 5-40 times slower.

Note, that we need (almost) synchronous communication of the internal boundary data. The data is needed right after it has been computed (on a neighboring PE). Nevertheless we implemented both the `SHMEM_GET` variant and the `SHEM_PUT` + `SHMEM_BARRIER` + cache flush variant. `SHMEM_PUT` is about twice as fast as `SHMEM_GET`, but it is not a priori clear whether the total time, which includes also the time needed for the shared-memory-operation barrier and the cache flush, will also be less.

3.2 Portability

The first two programming paradigms result in codes that are standard Fortran since all changes needed are implemented through directives. Of course the parallelizability is not portable to other architectures without adding the machine-specific directives.

In the domain decomposition style the master program is restricted to the T3D since a heavy use is made of the non-standard SHMEM routines. It is however easy to convert it to a program to be executed on a PVM.

4. Results

In this section we compare the various implementations with each other with respect to their overall performance and with respect to the computationally expensive parts of the model, viz., the computation of the fluxes and the coupled solution of chemistry and vertical diffusion. We use the following abbreviations for the implementations:

- DP1: *Data Parallel* with data distribution 1.
- DP2: *Data Parallel* with data distribution 2.
- DWS: *Data and Work Sharing* (data distribution 1).
- SMG: *Domain Decomposition* implemented with `SHMEM_GET` (data distribution 1).
- SMP: *Domain Decomposition* implemented with `SHMEM_PUT` (data distribution 1).

We use two different grid sizes. Since for `SHARED` arrays all but the last dimension have to be a power of 2, we use 32 grid points in the vertical direction (in contrast to 40 grid points as used in the original tests in [8]). The horizontal domain is divided in a 16×16 grid and a 32×32 grid, respectively.

4.1 Hardware and Programming Environment

The Cray T3D machine used for this report is located at the EPFL (École Polytechnique Fédérale de Lausanne) and was connected via the front-end Cray YMP and the Internet network to the CWI. The T3D at the EPFL consists of 256 PEs with 64 Mbytes or 8 Mwords local memory per PE.

The results in this report were obtained with the CF77 6.2.2 Programming Environment. The Fortran compiler recognizes a subset of Fortran 90, including array syntax. The timings were obtained

with the `rtc` intrinsic function which returns real-time clock values (compiler options `-Wf"-oaggress -ounroll"`). We used the Apprentice tool to determine the performance and to look into details of (specific parts of) the code (compiler options `-Wf"-Ta -oaggress -ounroll"`).

4.2 Full Model Timings

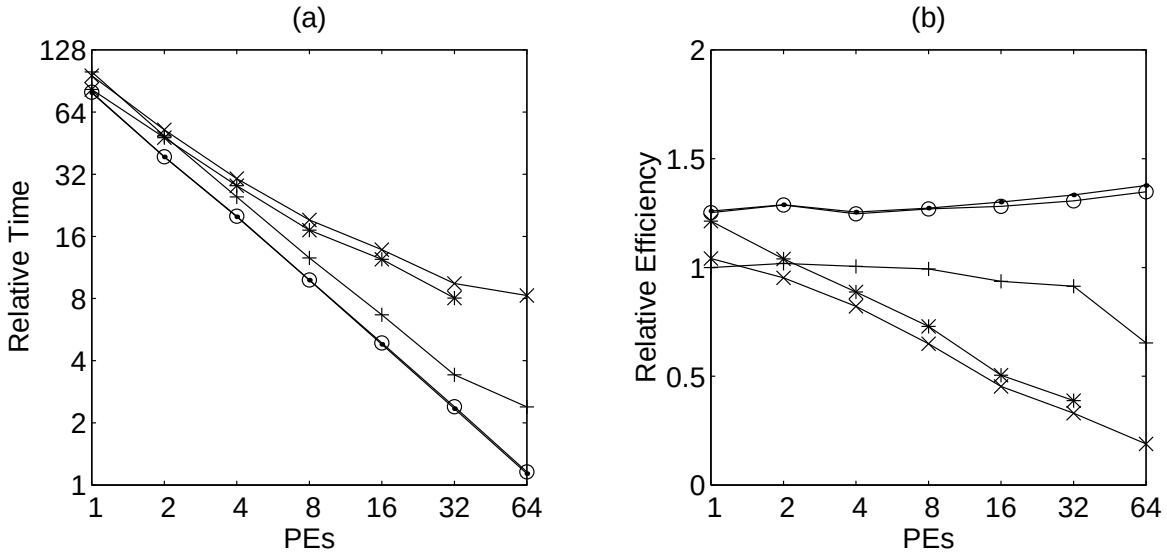

Figure 1: Execution time (a) and parallel efficiency (b) for a 16×16 horizontal grid.
DWS: $+$, DP1: $\times$, DP2: $*$, SMG: $\circ$, SMP: $\cdot$

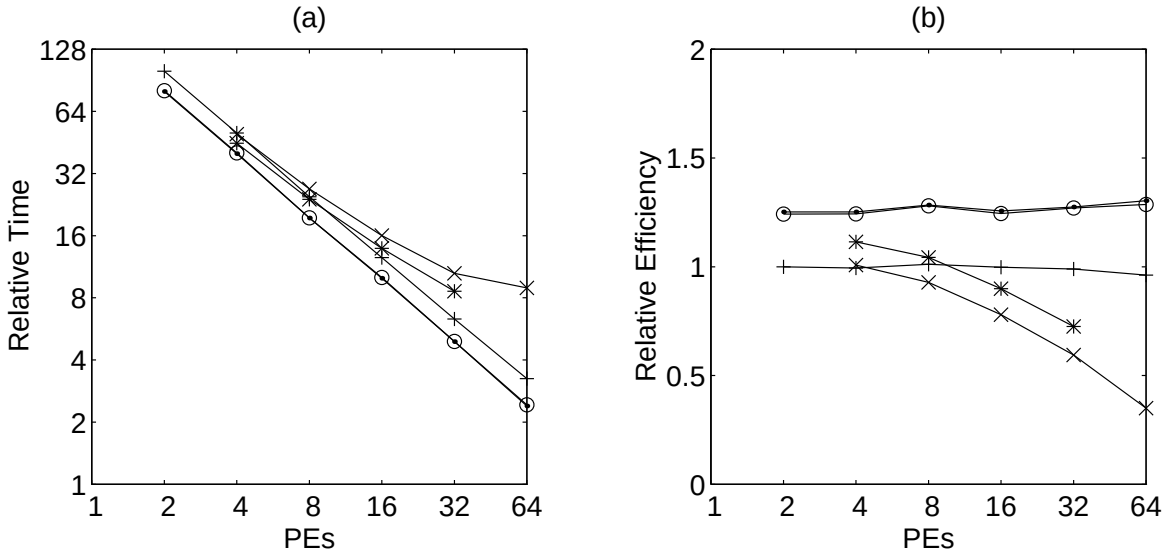

Figure 2: Execution time (a) and parallel efficiency (b) for a 32×32 horizontal grid.
DWS: $+$, DP1: $\times$, DP2: $*$, SMG: $\circ$, SMP: $\cdot$

In this section we give the results of the execution timings for a varying number of processors obtained for the various implementations. To depict the performance we use the same graphical representation as in [2], namely two plots displaying raw speed and scaling:

(a) Execution time: relative to the DWS implementation single processor execution time. The quantity

$$\frac{100 \times \text{Time } (N \text{ PEs})}{\text{Time(DWS 1 PE)}}$$

is plotted against the number of PEs, on a $\log_2$-$\log_2$ scale.

(b) Parallel Efficiency: speed-up per PE relative to the DWS single PE execution time. The quantity

$$\frac{\text{Time(DWS 1 PE)}}{N \times \text{Time } (N \text{ PEs})}$$

is plotted against the number of PEs on a $\log_2$ scale.

Plot (a) measures the execution time on N processors. It shows the relative performance of the implementations and it can be used to see how far increasing the number of processors decreases the execution time. Plot (b) measures the scaling behavior with the number of processors. Linear scaling corresponds to a horizontal line.

Due to a lack of memory it was not possible to execute the 32×32 test on 1 PE so that the quantities in Fig. 2 are relative to the time of DWS on 2 PEs. The Data Parallel versions (DP1 and DP2) of the code needed even more memory than available on 2 PEs.

Fig. 1 and 2 show that the Domain Decomposition approach scales nicely with the number of PEs. The same holds for the Data and Worksharing implementation unless the number of horizontal grid points (2×2), and thus the amount of work, on 1 PE is very small. However, the Data Parallel implementation, which for small numbers of PEs even outperforms the DWS implementation, falls quickly below the efficiency line of the other implementations. The reason for this poor scalability is twofold. Firstly, the compiler unexpectedly puts barriers *in front of* an array syntax statement thus inducing unnecessary synchronizations. These barriers can not be removed with `NO BARRIER` directives. Secondly, `SHARED` temporary arrays are allocated, presumably to store intermediate results. We have the impression that this only happens if the array syntax statements are placed inside a loop. We have no reasonable explanation for these phenomena.

The scalability with the model size is perfectly linear for the DWS and SM implementations. The runs for the 32×32 grid on N processors are as expensive as the runs for the 16×16 grid on $N/4$ processors.

4.3 Apprentice output

To get more insight in the performance of the various parts of the codes we used the Apprentice tool. For a various number of processors we show in Tables 1 and 2 the Megaflop rate per processor for the computationally important parts of the advection and the solution of the chemistry coupled with the vertical diffusion. Also the percentage of time spent in the 'communication' routines `barrier`, `SHMALLOC`, `shfree`, `REDIST_4_1W`, `SHMEM_GET`, `SHMEM_PUT`, and `SHMEM_BARRIER` is given. Note, that this percentage can vary for different runs. E.g., we think that the 14% communication time listed in Table 1 under SMG on 16 processors is an anomaly.

For the DWS and SM codes the time spent in the communication routines is determined by the `barrier` routine. This indicates that one can not expect a benefit from using `SHMEM_PUT` instead of `SHMEM_GET`. For the data parallel implementations much time is spent in `SHMALLOC` and `shfree`

	DWS			
	MFlop/s/PE			
NPE	Flux λ	Flux ϕ	Chem.	% comm.
1	3.5	4.2	5.2	0
4	3.4	3.8	5.3	2
16	3.0	3.1	5.5	7
64	2.4	2.1	5.7	28

	DP1				DP2			
	MFlop/s/PE				MFlop/s/PE			
NPE	Flux λ	Flux ϕ	Chem.	% comm.	Flux λ	Flux ϕ	Chem.	% comm.
1	3.8	4.3	5.2	4	6.9	7.1	5.1	5
4	3.2	3.5	4.6	20	6.8	7.0	4.6	27
16	2.2	2.3	3.2	42	6.6	6.8	3.2	51
64	0.8	0.8	1.5	53				

	SMG				SMP			
	MFlop/s/PE				MFlop/s/PE			
NPE	Flux λ	Flux ϕ	Chem.	% comm.	Flux λ	Flux ϕ	Chem.	% comm.
1	8.0	9.4	5.1	0	8.0	9.1	5.2	0
4	7.7	9.3	5.3	2	7.6	9.4	5.3	2
16	8.1	9.1	5.6	14	8.0	9.2	5.7	5
64	9.5	9.4	6.4	24	9.3	9.5	6.6	21

Table 1: Performance (in Mflop/s per processor) of flux computation and of chemistry coupled with vertical diffusion and the percentage time spent in communication routines. 16×16 horizontal grid.

	DWS			
	MFlop/s/PE			
NPE	Flux λ	Flux ϕ	Chem.	% comm.
4	3.5	4.1	5.2	1
16	3.3	3.8	5.3	3
64	3.1	3.3	5.5	9

	DP1				DP2			
	MFlop/s/PE				MFlop/s/PE			
NPE	Flux λ	Flux ϕ	Chem.	% comm.	Flux λ	Flux ϕ	Chem.	% comm.
4	3.5	3.9	5.2	7	7.0	7.2	5.2	14
16	2.8	3.1	4.6	20	7.0	7.1	4.6	27
64	1.3	1.4	3.2	39				

	SMG				SMP			
	MFlop/s/PE				MFlop/s/PE			
NPE	Flux λ	Flux ϕ	Chem.	% comm.	Flux λ	Flux ϕ	Chem.	% comm.
4	8.0	9.4	5.1	1	7.8	9.4	5.2	1
16	7.7	9.3	5.3	4	7.7	9.3	5.3	3
64	8.2	9.2	5.6	7	8.1	9.2	5.7	8

Table 2: Performance (in Mflop/s per processor) of flux computation and of chemistry coupled with vertical diffusion and the percentage time spent in communication routines. 32×32 horizontal grid.

together with the required `barrier` calls. The implicit redistribution of data needed in DP2 gives a significant increase in communication time but the gain in computational efficiency seems to outweigh this disadvantage according to the relative efficiency graphs in Fig. 1 and 2.

The flop-rate listed under Chem. is obtained for one of the 11 subroutines that solve the chemistry coupled with the vertical diffusion. However, the other routines show a comparable behavior. The performance of this routine is hampered by the fact that a lot of data values are needed which are used only once. Thus memory operations determine the speed. Apprentice output shows that 30% of the time is spent executing 'work' instructions, whereas 70% is required for loading instruction and data cache. Optimization by hand, using stripmining and prefetching data, can double the Mflop rate (as shown in [5]) but this would be hard and tedious labour. Undoubtedly future releases of the compiler will take care of this.

Tables 1 and 2 also show the effect of using local data for the computation of the advection: the Mflop rate more than doubles, but at the cost of communication time elsewhere. We attribute the high numbers in Table 1 for the SM codes on 64 processors to cache effects. The `PRIVATE` data arrays are of dimension $2 \times 2 \times 32$ and thus fit easily in the cache.

Finally the Apprentice output shows that it is not likely, that `SHARED` to `PRIVATE` coercion of the arrays when entering the chemistry routine will result in a significant increase of the performance. The computation of the chemistry coupled with the vertical diffusion implemented with `PRIVATE` data is approximately 5-10% cheaper than using `SHARED` data. Since `SHARED` to `PRIVATE` coercion results in non-portable code we do not advocate this implementation method. N.B. the fact that the Megaflop rates for DWS and SM for corresponding problems are almost equal (see Tables 1 and 2) is due to a 1.5 times as large number of floating point operations. This can possibly be the result of the more complex address calculation used for `SHARED` data access.

5. Discussion

The scalability both with respect to the model size and with respect to the number of processors is almost perfect, although the linear speed-up is perhaps more the result of a poor performance on 1 PE combined with a very fast interconnecting network. A different cache policy and a better optimizing compiler (cache prefetching, look-ahead, loop optimization by stripmining, unrolling, splitting and fusion) could probably result in an increase of the computational performance by a factor of 5.

Since logically shared non-local memory will never be addressed through the cache (this would imply a hardware update of *all* caches) it is our expectation that a better 1 PE performance would benefit the explicit message passing implementations. However, if the compiler deficiencies are resolved, the data parallel / HPF approach will also result in an efficient code. Moreover, such a program will be, without adaptations, efficient on a shared-memory parallel/vector, on a distributed-memory parallel architecture, and on a workstation. For a (heterogeneous) cluster of workstations it is unlikely that a very efficient implementation of HPF will ever exist. Besides, the interconnecting network presumably is too slow and the communication times too irregular to use the SIMD or SPMD approach in combination with an IMEX integration method.

Finally, we want to give an indication of the relative performance of a model like (1.1) on the Cray C90 and on the Cray T3D. On 1 processor of the C90 a 5-day run on a $33 \times 33 \times 40$ spatial grid took 1650 CPU seconds with a computational speed of 500 Mflop/s (cf. [8]). If we extrapolate our results for the SM runs on a $32 \times 32 \times 32$ grid a 5-day run would take 1300 seconds on 64 processors, which is approximately the same amount of time ($32/40 \cdot 1650 = 1320$). The good news about the T3D is that, because of the excellent scaling, one can run in the same amount of time on 1024 processors a problem with grid size $128 \times 128 \times 32$, thus obtaining a speed of 8 Gflop/s, whereas on the C90 the computational performance does not scale linearly with the number of processors and the amount of memory does not scale at all. Moreover, the compiler optimization techniques for a shared-memory

vector/parallel architecture have greatly improved in the last 10 years, whereas the compiler technique for the distributed-memory architectures is still in its infancy. Even the 1 processor optimization of the current T3D CF77 compiler leaves room for improvement.

6. Conclusions

In this paper we compared the different programming paradigms available on the Cray T3D - data parallel, data and work sharing, and explicit message passing - for the implementation of a prototype of an Atmospheric Chemistry/Transport Model. As mentioned in Section 3 the amount of work needed to convert a code is limited and the resulting codes are portable with the exception of the main program in the message-passing implementation. An evaluation of the performance tests done with these codes show a perfect scalability both for the data-and-work-sharing and for the message-passing variant. Deficiencies in the compiler precluded that the results for the data-parallel version are equally favorable.

In the previous section we discussed our expectations for distributed-memory architectures and we indicated that the Cray T3D can easily outperform the Cray C90 with respect to the computational part of an Atmospheric Chemistry/Transport Model. To model real-life processes, however, one also needs a scalable I/O architecture and further experiments are needed to see whether the T3E will be a good candidate for long-term simulations with an Atmospheric Chemistry/Transport Model.

Acknowledgements

We gratefully acknowledge the 'electronic' help of the PATP Project Support Team at EPFL, in particular of Michel Roche and Christopher Potter. We thank Bert van Corler of SARA (the Netherlands) and Hans Nelemans of Cray for the installation of the Apprentice tool on the C90 of SARA.

References

1. J.G. Blom, W. Hundsdorfer, and J.G. Verwer. Vectorization aspects of a spherical advection scheme on a reduced grid. Report NM-R9418, CWI, Amsterdam, 1994.
2. A. Ewing and D. Henty. HPF on the Cray T3D. A comparison with Craft. T3D Technical Report EPCC-TR95-05, Edinburgh Parallel Computing Centre,
   URL: <http://www.epcc.ed.ac.uk/t3d/documents/techreports/EPCC-TR95-05/>, 1995.
3. D. Henty. Craft performance optimisation. T3D Technical Report EPCC-TR95-04, Edinburgh Parallel Computing Centre,
   URL: <http://www.epcc.ed.ac.uk/t3d/documents/techreports/EPCC-TR95-04/>, 1995.
4. W. Hundsdorfer, B. Koren, M. van Loon, and J.G. Verwer. A positive finite-difference advection scheme. *J. Comput. Phys.*, 117:35–46, 1995. Revision of CWI Report NM-R9309.
5. Ch. Kessler, J.G. Blom, and J.G. Verwer. Porting a 3D-model for the transport of reactive air pollutants to the parallel machine T3D. Report NM-R9519, CWI, Amsterdam, 1995.
6. M. Sidani. ScaLAPACK. CRAY-EPFL PATP HTML information,
   URL: <http://patpwww.epfl.ch/scalapack/blacs/perf/blacs-perf.html>, 1995.
7. D. Simpson, Y. Andersson-Skold, and M.E. Jenkin. Updating the chemical scheme for the EMEP MSC-W model: Current status. Report EMEP MSC-W Note 2/93, The Norwegian Meteorological Institute, Oslo, 1993.
8. J.G. Verwer, J.G. Blom, and W. Hundsdorfer. An implicit-explicit approach for atmospheric transport-chemistry problems. Report NM-R9501, CWI, Amsterdam (to appear in *Applied Numerical Mathematics*), 1995.