Centrum voor Wiskunde en Informatica
REPORT*RAPPORT*

Porting a 3D-model for the transport of reactive air pollutants to the parallel machine T3D

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

Department of Numerical Mathematics

NM-R9519 1995

Report NM-R9519
ISSN 0169-0388

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

Porting a 3D-model for the Transport of Reactive Air Pollutants to the Parallel Machine T3D

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

CWI

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

Abstract

Air pollution forecasting puts a high demand on the memory and the floating point performance of modern computers. For this kind of problems massively parallel computers are very promising, although the software tools and the I/O facilities on those machines are still under-developed. This report describes the work involved to port a 3D model, which was optimized for the parallel vector computer Cray C90, to a parallel distributed memory computer, namely the Cray T3D. After some introductory remarks about the model and the communication therein, examples are given to describe the performance of the T3D-machine on simple problems. Thereafter performance measurements are shown with kernels of the model using shared and explicit programming.

AMS Subject Classification (1991): Primary: 65Y05. Secondary: 69L20, 65M20.
CR Subject Classification (1991): G.4, J.2, G.1.8.
Keywords & Phrases: parallel computation, distributed memory, Cray T3D, air pollution modeling, partial differential equations, method of lines.
Note: We gratefully acknowledge support by the Human Capital and Mobility Programme of the European Community - Project CT93-0407: The Equations of Fluid Mechanics and Related Topics. This work was also 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).

1. Introduction

Research in air pollution is meant to advance insight into an area with several levels of complexity. Computation of transport of air pollution requires the knowledge of windfields, the preparation of which already represents a computationally intensive task with a moderate amount of variables at each grid point. Another problem, often not fully considered in its complexity, is the compilation of emission data bases. If one deals also with the chemical transformation of the transported pollutants then there is the additional complexity of chemistry. The spatial resolution and the number of species considered in the chemical transformations are the factors determining size and complexity of the problem. Size and availability of current supercomputers still restrict severely the manageable problems both in resolution and number of species considered. One way of increasing computer performance is to use massively parallel computers for the solution of such systems. Although the software tools and in particular the (parallel) I/O facilities are in general not yet well developed, experimenting with such computers can give an indication to what extent a future generation of massively parallel systems can be used for simulating atmospheric transport and chemistry.

This report describes the porting to a massively parallel machine, viz. the Cray T3D, of an existing 3D-model (cf. [15]) which solves the equation system for transport and transformation of chemically active air pollutants. The algorithms used in this model are not restricted to the field of air pollution; they belong to the field of transport equations (partial differential equations) and chemical reaction equations (nonlinear ordinary differential equations). The focus of this study is to look at the feasibility of porting a code that is already optimized for a parallel vector machine (Cray C90). The code was

therefore not completely rewritten to gain an optimal performance, only structural changes were introduced which could be coded in Fortran. Although the C90 has parallel features its architecture is very different from that of the Cray T3D. The most significant difference is that the C90 is a shared memory system with a maximum of 16 vector processors, whereas the T3D is a system with physically distributed memory and with a maximum of 2048 scalar processors.

The following section describes the T3D machine and its software environment and briefly mentions some concepts which appear in later sections. In the third section some features of the 3D-model are presented together with the algorithms to solve the various parts of the model. In Section 4 the parallelism between and within the various subtasks is investigated and the choice of the data structures and the mapping of the data onto the different processors is discussed. In the fifth section performance measurements with a simple code fragment are discussed. In Section 6 kernels of the 3D-model are used to measure performance and speedup of two main subtasks in the model. Finally, conclusions are given.

2. The Cray T3D

First we will give an exposition of some concepts which occur frequently in this report.

cache Many modern CPUs can calculate much faster than they get data to calculate on. Memory and access connections which match the CPU speed are very expensive and used only for a small memory subset where data can be preloaded to and, possibly, be reused [11]. The T3D-processors have only cache located on the processor (L1-cache).

cache coherency Data can be accessed via cache on one processor and then updated by another processor without using the PE owned cache. The data in the cache might then represent 'old data' and wrong results are generated."A memory scheme is coherent if the value returned on a LOAD instruction is always the value given by the latest STORE instruction with the same address." [6]

race conditions Several PEs update the same data. Only data from the PE which writes last is kept and wrong results might be generated.

synchronization PEs can compute with unequal load and might compute results with data from other PEs which are slower in delivering 'new' data. Synchronization points are then used to ensure that some computations are only started when preceding ones (on other PEs) are finished. The absence of synchronization points can lead to wrong results.

load balancing Ideally each processor should have the same amount of work to do. Idle processors waste CPU-time.

latency The time which elapses between making a request and receiving the associated response [4]. The request can be for data located on the PE making the request or for data in shared memory on other PEs. Latency hiding can be done by overlapping instructions and instruction prefetch.

scalability of the machine The machine can be used with different numbers of PEs without changing its functionality. Connected to this is fault tolerance, i.e. the failure of one PE does not lead to failure of the machine.

scalability of object code At compilation time the number of PEs finally applied does not have to be known. The generated code should not reflect the machine configuration (number of PEs, interconnection topology).

scalability of the model This denotes the (simple) fact, that with the coding of a model onto a parallel machine more PEs should do more work.

data distribution The mapping of arrays on PEs. In general the mapping can be done by distributing dimensions of an array. An array with `a(8,4)` can be distributed, e.g. on 2 PEs as `a(1:4,4)`, `a(5:8,4)`. Here the first dimension is distributed, the second is not distributed as it resides completely on each PE.

data transposition The data distribution can be changed inside a model.

work sharing Computations in loops can be distributed over PEs.

2.1 The hardware

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 machine is configured to be driven by a front-end, e.g, a Cray YMP, which does compilation and job queueing thus freeing the T3D for numerical computations. The job queueing system allows to use $2^n, n = 0, 1, \ldots$ processors.

The 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.

2.2 The programming environment

Compilers for assembler, C and Fortran are available. The Fortran compiler which was mainly used for this report, includes a subset of Fortran 90, viz. array syntax. The Totalview debugger can be used to debug parallel programs and the Apprentice tool determines performance and other items of the code. The results in this report were obtained with the CF77 6.2 Programming Environment.

2.3 The programming paradigms

This denotes a style of programming applied to use various levels of complexity (and thus access various levels of performance). The lowest level language is assembler which is typically used for small routines to be called by higher level languages. With assembler almost no implicit mechanisms are provided to address the above mentioned items. The next level is explicit shared memory programming. Here access to the memory of other PEs is given by calling routines where the id of the PE and the address location have to be given (explicitly). The programmer has to specify how the PEs are to operate to solve a given problem [3]. Since this programming mode can use simple index and address calculations it will produce fast code. However, it is in the care of the programmer to ensure that the PEs are properly synchronized and that no race conditions or cache incoherencies occur. A more user-friendly approach is the data and work sharing model CRAFT (**C**ray **R**esearch **A**daptive **F**or**T**ran). Using directives the data is distributed over the PEs and loop iterations are done on the PE which 'owns' most of the data. With this model the access to shared, non-local memory is organized by the compiler/loader. The highest programming model is intended to make the transition between 'serial' and parallel programming quite easy. Apart from some directives for the declaration of arrays no other directives are necessary. The compiler properly distributes the work and provides

for synchronization where it is required. This model is accessed by using the CRAFT array syntax (as in Fortran 90). However, with array syntax, the current version of the compiler produces code which is not yet adequate for the capabilities of the T3D.

3. The model

A detailed description of the model is given in [15]. The model describes the transport and chemical transformation of air pollutants in the atmosphere and is given by the partial differential equation

$$\frac{\partial \rho}{\partial t} + \mathrm{div}(\underline{u}\rho) = \frac{\partial}{\partial \sigma}(K\frac{\partial \rho}{\partial \sigma}) + r(t,\rho), \tag{3.1}$$

in spherical coordinates. Here, $\rho = \rho(\lambda, \phi, \sigma, t)$ is a vector in $\Re^m$ of m concentration values, $\underline{u} = ue_\lambda + ve_\phi$ is the horizontal velocity vector with (e_λ, e_ϕ) the unit vectors on the sphere in the longitude (λ) and latitude (ϕ) direction, K is a scalar diffusion coefficient in the vertical direction (σ), and the horizontal divergence operator is given by [18]

$$\mathrm{div}(\underline{u}\rho) = \frac{1}{\alpha \cos\phi}\left(\frac{\partial}{\partial \lambda}(u\rho) + \frac{\partial}{\partial \phi}(v\rho\cos\phi)\right), \tag{3.2}$$

where α is the radius of the earth. Vertical advection and horizontal diffusion can be considered by simply including additional terms. Note that in reality these two transport processes are significantly less important than horizontal advection and vertical turbulent diffusion.

The vector function $r(t,\rho)$ defining the chemical transformation, emission and dry deposition, has the special form

$$r(t,\rho) = P(t,\rho) - L(t,\rho)\rho, \tag{3.3}$$

where $P(t,\rho)$ is the vector of production terms and $L(t,\rho)\rho$ the vector of loss terms with $L(t,\rho)$ a diagonal matrix. Currently the chemical transformations are determined using the state-of-the-art chemical reaction model EMEP[13]. Transformations of emitted air pollutants, e.g. CO, NOx and VOC due to chemical and photochemical reactions lead to secondary air pollutants which may be even more hazardous to the environment. The calculation of these chemical transformations leads to nonlinear ordinary differential equation systems (ODEs), which are termed stiff because of vastly differing time scales in the ODEs. For many species the reciprocal of their entry in L (see eq. (3.3)) is a good approximation of the physical time constant or characteristic reaction time [12]. The ratio between the approximate lifetime of rather inert species as methane and short lived species as, e.g. radicals might exceed the order of 10^{15}. Special care has to be taken in the integration of the ODEs to adequately treat this large range of variability. In [14, 17] a second order backward formula (BDF2) together with Gauss-Seidel iteration was proposed to solve the box model. This algorithm was shown to compare well to other stiff solvers, e.g. general stiff solvers as VODE[16] and special purpose solvers with quasi-steady-state-approximation (QSSA) [17]. The BDF2 solver is especially suited for large air pollution models. since it requires neither the large computer storage nor the considerable CPU time generally associated with the use of Jacobians in implicit stiff solvers. Apart from the straightforward transformation of reaction equations into differential equations no further 'expert' knowledge has to be coded into the model. Reaction models might thus be exchanged easily providing a means for comparing different chemical models.

Diffusive processes are only considered in the context of vertical turbulent diffusion. The influence of molecular diffusion is several orders of magnitude smaller than that of vertical turbulent diffusion. Horizontal diffusion is also neglected due to its small influence. When the windfield is highly turbulent the characteristic times for concentration changes due to turbulent exchange and those due to chemical

transformation can be of the same order of magnitude. In this case it is advisable to solve the chemistry and the turbulent diffusion coupled when this can be done without much additional work.

The numerical solution method for (3.1) is derived along the method of lines (MOL). The advection operator (3.2) is discretized by the mass-conservative, flux-limited finite-difference scheme proposed in [10]. This finite-difference scheme is based on the 3rd-order upwind biased discretization (the $\kappa = 1/3$-scheme in the terminology of van Leer) and has been found to be very suitable for this application (see also [2, 5]). The resulting 9-point stencil has 5 grid points along both the λ- and the ϕ-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 method used is based on a 2nd-order implicit-explicit backward differentiation formula (BDF) 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. [15]).

4. Data mapping

In the code developed for the Cray C90 ([15]) the data structure used for transport related items was a three-dimensional array (λ,ϕ,σ) and for the concentrations of species a four-dimensional array which has the number of species as last dimension. The computational grid is fixed, uniform in the horizontal directions and nonuniform in the vertical direction.

In the shared memory programming model one of the first steps of porting an already existing code is the mapping of data onto the PEs. This is done by inserting compiler directives for each array. It is possible to change the data distribution according to the computational tasks, i.e. from one subroutine to the next. In the present study this option was not used since the large data stream between PEs, necessary to rearrange the mapping, is expensive. The most computationally intensive task should therefore be used to decide which data distribution to use. In the following some features of the main computational tasks are given, i.e. transport, chemistry and diffusion, together with a discussion of the dependencies between grid cells.

The fluxes are computed along the λ- and ϕ-direction separately. This represents a coupling of the grid cells in these directions. There are two neighbour values needed in the upstream and downstream direction. A data distribution to make communication between PEs unnecessary would arrange data such, that at least one (horizontal) layer is completely mapped onto one PE. The numerical solution of the ODE-system for the chemical transformations gives a coupling along the concentrations of species. If no other processes are included in the solution process, the only requirement is then to keep the dimension of concentrations on each PE. Vertical turbulent diffusion by itself couples vertical grid cells. No communication between PEs is required when the (height or vertical) dimension of σ is kept on each PE. As the coupled solution of both chemistry and vertical turbulent diffusion requires the greater part of CPU-time of the model a data distribution was chosen which combines the needs of both solution processes. This is a distribution, which leaves the dimension of σ and the dimension of concentration on one PE but distributes the first two (λ and ϕ) dimensions. With this column-like mapping there is no restraint on the number of PEs which can be used since the number of grid cells in these two directions is, in practical models, much greater than that in the vertical.

It should be noted that in the current code the concentrations are stored as one dimension in the array. Another possibility would be to resolve this dimension of n concentrations into n arrays where each array only stores the concentration of one species. The reasoning for this splitting is as follows: The current limitation on the size of a dimension which is to be distributed across PEs requires that this size is a power of two. Memory addresses with a distance of 1024 words are connected to the same cache line. These two facts often lead to a situation where the address location of two species are just a multiple of 1024 words away. The resulting cache thrashing (see the example in Section 5.1) can be avoided by inserting dummy arrays between arrays for concentrations. The gain in performance is

```
      real a(nl), b(nl)
C preload the cache
      do il=1,nl
          a(il) = 0.0
          b(il) = 0.0
      enddo
C start measuring
      itime1 = irtc()
C perform axpy operation
CDIR$ unroll 7
      do il=1,nl
          b(il) = a(il) + dt* b(il)
      enddo
      itime2 = irtc()
```

Figure 1: Example of `axpy`

not so much obtained by the reuse of cached values but by the possibility to preload these data into cache prior to their use [1] (latency hiding). As it seemed from tests, done with several arrangements of data, the compiler does not currently provide automatic preloading.

5. Basic performance examples

In order to get insight into the performance of the T3D, first a simple example has been coded (Figure 1) which is also known as *axpy*. This simple code is a paradigm for the complete code in the sense that on the average processor loads and stores will be dominating the floating point operations.

5.1 Results on one Processor Element

Purpose of this example is to understand some basic concepts of cache usage and speedup by more directly controlling the way memory is used. This example uses no parallel features of the T3D. Some results are given in Table 1. The example with $nl = 512$ gives the best performance since both arrays (with $2 * 512$ elements) fit into cache and do not have to be loaded from the local memory. The arrays were also preloaded into the cache before measuring and therefore this example operates under favorable conditions. A loop unrolling number larger than 7 could overload the instruction cache and could also result in cache-thrashing.

nl	without unroll	with unroll 7
512	17.2	56.3
1024	4.9	14.2
4096	4.4	11.7

Table 1: Performance (in Mflop/s) of the code in Figure 1 with different array sizes

The drop in performance from arraylength 512 to 1024 results from the improper cache usage. The T3D has a direct mapped cache with 256 lines [7]. Each line can hold 4 words (one word has 8 bytes), the cache can thus hold 1024 words. With direct mapping the address locations, to which the cache lines are mapped, are fixed. Cache line no. 1 can hold the four addresses $1, \ldots, 4$ or $1 + 1024, \ldots, 4 + 1024$ etc.. In the case with array sizes of 1024 the addresses of $a(1)$ and $b(1)$ are connected to the same cache line. After loading the value of $a(1)$ (and also the next three values

nl	without unroll	with unroll 7
512	4.9	11.8
1024	10.8	15.8
4096	10.8	15.9

Table 2: Performance (in Mflop/s) of the code in Figure 1 with `common block a,c(512),b`

nl	$nc = 512$	$nc = msize$	$msize$
512	9.1	28.8	9984
1024	23.7	22.5	9472
4096	24.1	20.3	6400
8192	12.1	20.4	10496

Table 3: Performance (in Mflop/s) of the code in Figure 1 with `common block a,c(nc),b` to change cache usage and prefetch of data. Same *unroll* is used as in Table 2

$a(2), \ldots, a(4)$) the subsequent loading of b(1) will delete the already loaded values. Therefore the loading of $a(2)$ will fetch once more, because of the direct mapping scheme, the same $a(1), \ldots, a(4)$.

This *cache-thrashing* can be made less serious by placing arrays between a and b so that at least part of the arrays can be held in cache. With a cache of 1024 words half of it can be used for both a and b. Thus the insertion of a dummy-array with 512 elements can give a better cache use, especially if the loop is not unrolled (Table 2). The case with $nl = 512$ is inversely affected since here the use of a dummy-argument occupies the cache otherwise used for the second array b.

Extending the idea of preloading the cache further leads to *prefetching* of data into the cache[1]. A very fast assembler routine reads the data which is later used in the computation. The performance information given in Table 3 includes the execution time for the *prefetch*. As in the results cited above the example with $nl = 512$ is different from the other array sizes since again the cache thrashing occurs with a dummy size of 512. The additional time needed to prefetch the data leads to a lower performance than that given in Table 2. The larger array sizes, however, show a significant improvement, except for the case with $nl = 8192$. The local memory of a PE is divided into pages, which have a size of 4096 words[2] per page. Two pages can be addressed without changing a page frame. The addressing of memory outside a given pageframe requires additional operations, e.g. moving the page frame, and slows the loading of memory into cache. Thus the array size of $2 * 4096$ fits into one page frame while the array size of $2 * 8192$ does not.

To address the problem of page allocation for larger problem sizes a dummy size is given by Cray [8]. Using this size (*msize* in Table 3 right) improves the performance only for $nl = 512$ and $nl = 8192$. In the case with $nl = 512$ it is clear, that no dummy would serve best while in the case with $nl = 8192$ the increase in performance is due to the better page allocation.

5.2 More processor elements

The above mentioned examples have used no parallel features yet. The `axpy` is an operation which, even in parallel, uses no communication channels since all operations are local to the PE which owns the data. In a shared memory environment common blocks are handled differently and therefore

[1] This idea and assembler code were provided by David Tanqueray of Cray Research

[2] Machines with 2 Mwords per PE have a pagesize of 2048 words

```
      parameter (nl=64, np=64, ns=64)
      real a(nl,np,ns), b(nl,np,ns)
      ntot=nl*np*ns
      do is=1,ntot/16
      call pref(128,a(is,1,1),b(is,1,1))
CDIR$ unroll 7
      do il=is,is+511
        b(il,1,1) = a(il,1,1) + dt* b(il,1,1)
      enddo
      enddo
```

Figure 2: Example of `axpy` with three-dimensional array

# PEs	Performance
1	42.6
2	85.2
4	169.8
8	335.8
16	658.4

Table 4: Performance (in Mflop/s) of the code in Figure 2

improving of cache usage by means of dummy arrays is in the shared memory case replaced by information provided for the *loader*. The loader resolves memory addresses and adds system and library functions to produce executable code. The amount by which the different arrays (here a and b) are offset is called $SKEW$. Figure 2 shows once again the use of prefetch, in this example the `axpy` is computed for a 3D-array with 64^3 elements. Here a single loop optimization collapses two loop indices and operates on strips with equal length. This is termed *stripmining* [19, 9] and is a well known practice in optimizing compilers. In passing it should be mentioned that such optimization techniques were also used in the earlier days of vector computing but are now almost obsolete due to highly optimizing compilers. Table 4 shows the performance of the code in Figure 2 where 128 array elements are prefetched and a $SKEW$-value of 128 has been used.

6. Model experiments

The `axpy` example shown in the previous section worked on a simple dataset with only two arrays and it is easy to optimize such an example, e.g. by preloading, loop unrolling and arranging arrays in memory. In this section performance measurements are discussed which were done with the 3D-model. The starting-point was to parallelize the existing code using the shared-memory programming model with directives. The code was changed only for the array-structure of the boundary conditions, to allow for a uniform data distribution. Compiler directives were included for data distribution and work sharing. To reduce compile time of the chemistry kernel the latter was split up in 22 small routines; results are given for one representative routine only.

6.1 Model size

For the following calculations a model size was chosen of $32 \cdot NPE$ grid points in the λ-direction (West to East) where NPE is the number of processor elements. In the ϕ- and the σ-direction (South to North and vertical) the grid size was fixed at 32 and 8 grid points, respectively. The total number of grid points was therefore $8192 \cdot NPE$. This was the largest number of grid points with which the

NPE	Advection λ	Speedup	Advection ϕ	Speedup	Chemistry	Speedup
1	3.45	1	3.41	1	5.64	1
2	6.97	2.02	6.53	1.91	11.33	2.01
4	13.70	3.97	13.03	3.82	22.64	4.01
8	27.61	8.00	24.11	7.07	45.30	8.03
16	55.10	15.97	48.16	14.12	90.52	16.05
32	110.67	32.08	88.47	25.94	181.41	32.16
64	221.47	64.19	177.42	52.03	363.01	64.36

Table 5: Performance (in Mflop/s) and speedup of chemistry and advection kernel. For the advection fluxes are calculated in two dimensions separately

model could be tested and which also fulfilled the requirement that all but the last, i.e. the species, dimensions were a power of 2. The number of unknowns (# grid points · # concentrations) which had to be solved in each timestep, was $540,672 \cdot NPE$.

The scaling of the grid size with the number NPE was chosen to simulate a more practical example where a large machine is applied to solve a large problem. If a fixed grid size is used then the explanation of the speedup has also to consider memory-access times which change according to different memory page addresses and cache usage. With a fixed grid the result of a comparison of speedup versus NPE most often shows a superlinear increase with NPE, e.g. 64 PE might do 80 work units which is not in accordance with the concept of speedup. The chosen scaling leads to a physical model size which is stretched in the λ-direction. This represents a domain which would not be used in practical models, however, it serves to enhance the influence of communication in the ϕ direction, where fewer grid cells reside on each PE with increasing NPE. This gives a higher ratio of communication operations to floating point operations.

6.2 Results

Table 5 and Figure 3 show results for performance and speedup of different parts of the model namely computation of fluxes for the calculation of advection and the coupled solution of chemistry and vertical turbulent diffusion. The performance, given in Table 5 and Figure 3, per PE is considerably lower than that given in Table 2 for the optimized `axpy` operation. The flux calculation in the advection routines is characterized by communication operations and also max and min operations which reduce the number of flop/s.

The computation of the coupled solution of chemistry and vertical turbulent diffusion is characterized by the need for many data values. Especially for the calculation of the chemical transformation between 5 and more than 100 values are necessary to calculate the production and loss terms (see eq. (3.3)) for a single species at one grid point. Most of these values are needed only once, i.e. the number of (slow) loads from memory are approximately equal to the number of (fast) floating point operations.

Both advection in λ-direction and chemistry show an almost linear speedup. Although, in itself, this gives a very good scalability, it should be mentioned that the linear speedup occurs at a low performance level which does not stress the communication network. The advection in ϕ-direction shows the influence of the already mentioned effect of the scaling and gives 52 work units out of 64 PEs.

In contrast to the advection calculation the chemistry part of the model uses (with the given data distribution) only data which is on the PE which does the computations. Therefore there is no need for the full shared memory programming model. There are explicit functions which allow to write routines

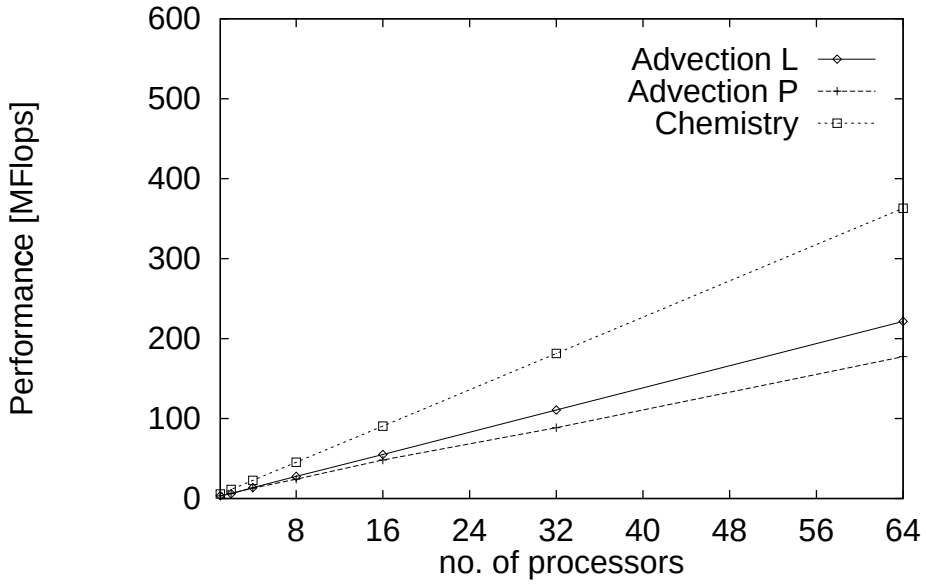

Figure 3: Performance (in Mflop/s) of advection and chemistry kernel

NPE	Chemistry
1	8.48
2	17.50
4	35.05
8	71.15
16	142.47
32	288.20
64	576.26

Table 6: Performance (in Mflop/s) of the chemistry with private data

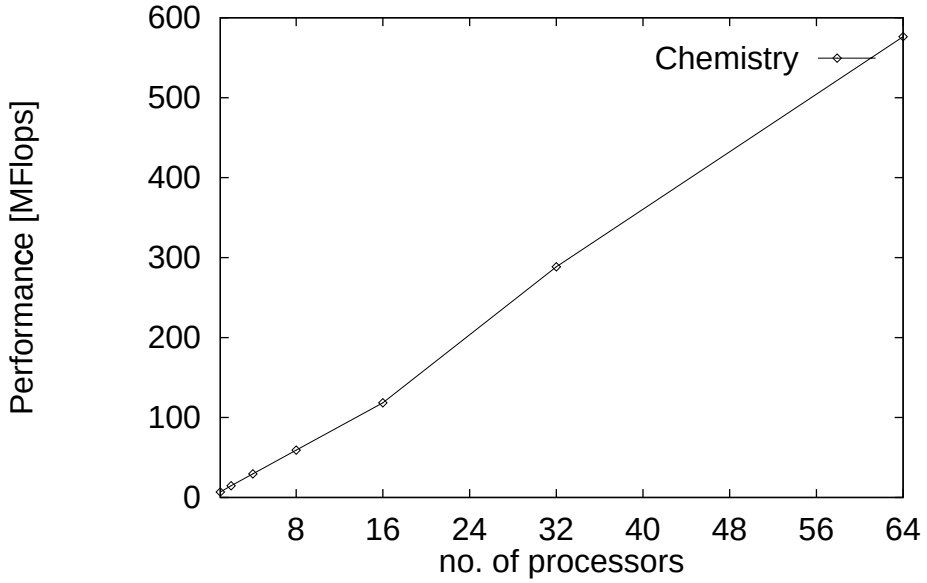

Figure 4: Performance (in Mflop/s) of chemistry kernel. Chemistry with private data

NPE	Chemistry
1	13.1
64	832.8

Table 7: Performance (in Mflop/s) of chemistry kernel with private data and with single loop optimization

which operate on local memory. The advantage of local memory is that smaller, local addresses are created while shared memory addresses can increase this overhead (measured in integer operations) by a factor of 10.

The benefit of coding the chemistry routine with private data is seen in Table 6 and Figure 4. Compared to the previous performance on one PE the increase is better than 50 percent. The amount of coding to implement explicit programming was less than that of including directives for work sharing in each loop but in general it requires a better understanding of the intricacies of the parallel design.

An increase in performance for the advection routines could also be gained by using explicit shared memory get/put routines. The reorganization of code will, however, be considerably larger since also synchronization aspects have to be taken into account. We plan to investigate this in the near future.

A further increase in performance for the chemistry can be gained by optimization of the loops (see Section 5.2, *stripmining*). Together with prefetching selected arrays the results in Table 7 show an increase of 130% for one PE compared to the results given in Table 5. However, this prefetch has to be determined separately for each species, due to the different data for the production and loss terms (eq. (3.3)). Nine arrays were included in `common` with an offset (SKEW) of 16 lines (64 words).

With the size of 1024 grid cells in the horizontal this was the highest number which could easily be used with an `unroll` of 7 and for which no additional loops were needed to treat the remainder of the division of 1024/offset.

7. Conclusions

This study on the porting of a 3D-atmospheric dispersion code showed that the T3D architecture can be easily utilized to perform dispersion calculations at a performance as high as 400 Mflop/s on 64 processors for the kernel (computing chemistry and vertical diffusion) which represents the main computational task of the 3D-model.

The shared memory programming model represents a convenient environment to the programmer and provides a clear method to port existing codes to the parallel architecture. The shared memory model also allows the use of explicit means of programming, i.e. coding for single PEs and is thus a way to further increase the performance of selected routines. This is especially helpful in that it greatly reduces the amount of startup time to have a working code. Subsequent fine tuning of computationally intensive subtasks are then performed from a fully operational base using the experience gained in the initial steps of porting. Extended optimization of the chemistry kernel, e.g., showed a performance of more than 800 Mflop/s for 64 processors.

The measurements showed a less than ideal performance compared to that reported for selected test problems (partly coded in assembly). This was mainly due to a not yet fully optimized cache utilization. The application of look-ahead and cache preloading should give a remarkable performance increase even for codes which do not often reuse cache. It is to be expected that future Fortran compilers will take care of these and other optimizations, like, e.g. stripmining. It is also likely that the different cache policy of the successor of the T3D will result in a more efficient cache usage.

The almost linear speedup shown in the experiments indicates the powerful interconnections of the different parts of the machines. Extrapolating the speedup for large dispersion models to a fully equipped T3D with 2048 processors would give a performance of 5 and 26 GFlops for advection and chemistry respectively. However, further studies should determine if the speedup experienced here remains as good with a better cache policy and for more optimized and faster codes produced by new releases of the Fortran compiler.

In this paper we discussed the feasibility of porting an existing code optimized for a parallel vector computer to the Cray T3D. Since the aim was to change as little as possible in the code this implicated the use of the shared memory programming model. For that part of the code that did not need any communication (chemistry and vertical diffusion) we also experimented with the use of private data which resulted in a speedup of a factor 1.5. In a forthcoming paper we plan to compare the use of private data and explicit communication versus the shared memory programming model also for the transport part and for the complete model.

References

1. Alpha architecture handbook. Manual, Digital Equipment Corporation, Boxboro, Massachusetts, 1992.
2. D.J. Allen, A.R. Douglass, R.B. Rood, and P.D. Guthrie. Application of a monotonic upstream-biased transport scheme to three-dimensional constituent transport calculations. *Monthly Weather Review*, 119:2456–2464, 1991.
3. G.S. Almasi and A. Gottlieb. *Highly Parallel Computing.* The Benjamin/Cummings Publishing Company, Inc., Redwood City, California, 1994.
4. Arvind and R.A. Iannucci. Two fundamental issues in multiprocessing. In D. Mueller-Wichards R. Dierstein and H.-M. Wacker, editors, *Lecture Notes in Computer Science*, volume 295. Springer-Verlag, Berlin Heidelberg New York, 1987.

5. 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.
6. L.M. Censier and P. Feautrier. A new solution to the coherence problem in multicache systems. *IEEE Transactions on Computers C-27*, 12:1112–1118, 1979.
7. Cray Research, Inc. *Cray MPP Fortran Reference Manual*, sr-2504 6.1 edition, 1994.
8. Cray Research, Inc. *Cray-T3D Applications Programming TR-T3DAPPL*, revision c release 1.0 edition, 1994.
9. K. Dowd. *High performance computing*, volume 3. O'Reilly & Associates Inc., Sebastopol, California, 1993.
10. 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.
11. V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to parallel computing: design and analysis of algorithms*. The Benjamin/Cummings Publishing Company, Inc., Redwood City, California, 1994.
12. G.J. McRae, W. R. Goodin, and J.H. Seinfeld. Numerical solution of the atmospheric diffusion equation for chemically reacting flows. *J. Comput. Phys.*, 45:1–42, 1982.
13. 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.
14. J.G. Verwer. Gauss-Seidel iteration for stiff ODEs from chemical kinetics. *SIAM J. Sci. Comput.*, 15:1243–1250, 1994.
15. 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.
16. J.G. Verwer, J.G. Blom, M. van Loon, and E.J. Spee. A comparison of stiff ODE solvers for atmospheric chemistry problems. Report NM-R9505, CWI, Amsterdam (to appear in *Atmospheric Environment*, 29), 1995.
17. J.G. Verwer and D. Simpson. Explicit methods for stiff ODEs from atmospheric chemistry. *Applied Numerical Mathematics*, 18:413–430, 1995.
18. D.L. Williamson. Review of numerical approaches for modeling global transport. In H. van Dop and G. Kallos, editors, *Air Pollution Modeling and Its Application IX*, pages 377–394. Plenum Press, New York, 1992.
19. Dik Winter. Scalar optimisation. In H.J.J. te Riele, Th. J. Dekker, and H.A. van der Vorst, editors, *Algorithms and applications on vector and parallel computers*, volume 3 of *Special topics in supercomputing*. North-Holland, Amsterdam, 1987.