HPCN and Air Quality Modeling

J.G. Blom, W.M. Lioen, and J.G. Verwer

CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
{gollum, walter, janv}@cwi.nl
http://www.cwi.nl/

Abstract. We discuss the implementation of an off-line air quality model (AQM). More precisely, how to design a code for an AQM that runs efficiently on a variety of computer platforms. We implemented our ideas in an AQM benchmark and we show the performance of this benchmark on the different architectural paradigms. A second subject of the paper is the I/O performance of the Cray T3E for an off-line model. We implemented the required I/O in different ways and show that none of these results in a truly scalable I/O.

1 Introduction

Atmospheric chemistry and transport models are used to simulate the change in the chemical composition of the atmosphere caused by changes in emissions, e.g., to determine the strategies needed to satisfy air quality requirements. The mathematical model of atmospheric chemistry and transport is a system of time-dependent partial differential equations in three space dimensions of the advection-diffusion-reaction type. 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. A second critical factor concerns computer capacity, both with respect to CPU power and with respect to memory (see, for example, the review article from Peters et al. [9]).

In a joint project with the TNO Institute for Environment, Energy and Process Innovation we are developing a three-dimensional regional atmospheric dispersion model. The current TNO LOTOS (LOng Term Ozone Simulation) model (see [6]) is used for a variety of environmental studies related to air pollution, but it is limited in the sense that it uses only four (physically determined) vertical layers and in the sense that a model change, for instance in the chemistry, requires much (implementational) effort. Moreover, the code is not developed for modern computer architectures, i.e., it is not easily vectorizable or parallelizable. The aim of the project is to significantly enhance the suitability, the accuracy and the computational efficiency of LOTOS by developing from scratch a fully three-dimensional Air Quality Model (AQM). Just like the old model, the new LOTOS model [5] is driven by an emission data base and analyzed meteorological data (a so-called 'off-line' model). To solve the mathematical model the

Eulerian grid approach is used. The numerical algorithms for space and time discretization aim at positivity, mass conservation, stability and efficiency. The computational model is implemented with two basic ideas in mind. Firstly, it should be a flexible code, i.e., it should be easy to add processes (like cumulus convection), to use different solution methods, and to replace the chemistry model. Secondly, the model is intended to run on different computer platforms and perhaps even on a heterogeneous network.

In this paper we will focus on the latter task: how to make an implementation of an off-line air quality model that is efficient on the most advanced computer systems like vector/parallel supercomputers and massively parallel distributed-memory systems and on (a cluster of) workstations. To have an easy-to-use problem at hand to evaluate various numerical algorithms we developed a 3D prototype of a regional atmospheric chemistry and transport model. We discretized this prototype on a computational grid with numerical algorithms that are either the ones that we intend to use in our full LOTOS model or which are typical for the kind of solution methods suitable for air quality modeling. With this computational model we studied in previous projects (see [10, 3]) the parallelization of the transport operators and the chemistry of an AQM on a massively parallel architecture, viz., the Cray T3D. As a result of these and other experiments we developed a *benchmark code* for our prototype which uses as basic parallelization strategy the SPMD (Single Program Multiple Data) approach through domain decomposition. The horizontal domain is divided into as many partitions as the numbers of processors available and on each of the processors a regional (sub)model is computed. The implementation of the basic regional model is done in data parallel / array syntax programming style to facilitate automatic vectorization and possibly also parallelization. For a precise description of the prototype and its benchmark implementation we refer to [1, 4]. In this paper we will show performance results with our AQM benchmark code on the successor of the Cray T3D, the Cray T3E, and on other platforms, such as a vector/parallel shared memory system (Cray C90) and a cluster of workstations (SGI O2).

A second subject of the paper is I/O related. A real AQM like LOTOS contains a considerable amount of I/O operations. Therefore, we added to our benchmark a 'realistic' amount of reading data from and writing output to disk. With this code we studied the I/O performance of the Cray T3E for our AQM. We implemented the I/O both in the master/slave approach, i.e., one PE performs all necessary I/O and takes care of the distribution and gathering of the data, and using parallel I/O.

Finally, we will draw some conclusions from the experiences with our prototype and we will discuss the computational design of the full LOTOS model.

2 Platforms

We compared the performance of our benchmark on the three different architectural paradigms, viz., a vector/parallel shared memory architecture (a

Cray C90 with 12 processors), a massively parallel distributed memory system (a Cray T3E), and a cluster of SGI O2 workstations coupled in a star-shaped ATM network. The difference in performance of these platforms is not caused by the clock frequency, which is for all platforms more or less the same. Also the peak performance per node is not significantly different. For 1 CPU of the C90 it is 1 Gflop/s (having 2 vectorpipes and chaining possibilities), for 1 PE of the T3E it is 0.6 Gflop/s (2 Flops/cycle), and even for the O2 it is 0.1 Gflop/s (all 64 bit Flops). But for the C90, and for vector machines in general, it is easy to reach half the peak performance, whereas for the other type architectures with their restrictive hierarchical types of memory (registers, primary and secondary cache, memory, disk) it is difficult to keep the CPU busy. For example, on the T3E with its relatively small cache it may be possible to get somewhere near the peak performance using blocked matrix algorithms, but for our application where the use of memory is inherently more or less random one reaches often less than 5% of the peak. For our application the only reason to choose distributed memory architectures is the scalability of the systems, and not only the scalability of the execution time with the number of PEs but even more the scalability of the memory. Very large models can often be only executed on distributed memory systems due to the inherent limitations of the (shared) memory in vector/parallel systems. Clusters of workstations have the same advantage but there the interconnecting network will often be the bottleneck, because it is slow and even more important because it is not dedicated. However, the price of such a platform is of course much lower.

3 Performance Benchmark

We ran our benchmark on a 32×32 horizontal grid with 32 layers in the vertical (so with concentration vectors of dimension [32,32,32,66]). For the explicit message passing we make use of PVM routines. Using another message passing interface, e.g., MPI, will not influence our findings significantly. We want to emphasize that we do not make changes in our code other than using two different 'calling' programs, one for the unpartitioned model and one for the horizontally partitioned model. We do not add directives to increase the performance and we use 'standard' compiler options, with the exception of the T3E where we compiled with '−O3,split2,unroll2' since that resulted in a performance gain of ca. 40%. For the C90 we use −Zv (vectorization) and −Zp (autotasking + vectorization), and for the O2: −O. In this way we test not only the quality of the hardware, but also the quality of the compilers and the user-friendliness of the system.

As normalizing computational unit we take the performance on 1 CPU of the C90 (compiler option −Zv). Our benchmark runs there at a speed of 500 Mflop/s, which is half the peak performance. It should be noted that the performance could have been better if the array syntax would not have been rewritten by the compiler to loops of which only the inner one was vectorized. Especially, the performance for smaller (sub)models could have benefited from this.

Table 1. Performance of the AQM benchmark on various platforms. First column: CPU performance on 1 processor normalized with respect to 1 CPU of C90 (−Zv). Other columns: Speed-up on N processors with respect to time on $N/2$ processors. For the C90 (−Zp) autotasking results are obtained with the Cray tool ATExpert, both the 'optimal' Amdahl (left entry) and the 'dedicated' (right entry) figures are given. For the SGI O2 the left entry in a column denotes the maximum CPU time and the right entry the wall clock time.

		Speed-up								
a	C90(−Zv)/a	2		4		8		16		32
C90(−Zp)	0.8	1.9	1.8	1.9	1.7	1.8	1.5			
C90(PVM)	0.8		1.9		1.9		1.6			
T3E	0.05		2.1		2.0		2.1		2.4	2.3
O2	0.03 0.008	2.0	5.6	2.2	1.9	1.9	1.4	2.1	1.8	

In Table 1 we show the performance of the benchmark on multiple processors of the C90, once for the complete model using autotasking to divide the workload over the processors, i.e., parallelizing at loop level, and once using the PVM program. The parallelization overhead is in both cases significant. On one processor as well the (unpartitioned) autotasked program as the PVM program show a performance drop of approximately 20%. As can be expected the PVM program scales better with the number of processors, but memory contention and a decreasing vector speed prevent an optimal scalability. On the Cray T3E, on the contrary, the scalability is even superlinear with the number of processors (due to cache effects), but here the performance on 1 PE is only 4% of the peak performance. One can calculate from the figures in the table that it needs 16 PEs of the T3E to outperform 1 processor of the C90. Experiments we performed on a 64×64 horizontal grid show that the scalability with the modelsize is also perfect, i.e., a run on N PEs for the 64×64 grid is as expensive as a run on $N/4$ PEs for the 32×32 grid. Finally, we present the results for a cluster of workstations, the SGI O2's coupled with an ATM network. Here one can clearly see the effect of the use of 'virtual' memory: the memory on these workstations is not large enough to contain a complete model, so the computer is more swapping than calculating, resulting in a wall clock time which is four times as large as the CPU time (and has a clear day/night rhythm: during daytime it is about 6 times as large). The speed-up in wall clock time using two O2's instead of one is therefore huge: a factor 5.6. The performance is then approximately the same as for 1 PE of the T3E. However, the scalability of a cluster of workstations is much less. The CPU time can also decrease sometimes superlinearly due to cache effects, but this does not show up in the wall clock time.

4 I/O experiments on Cray T3E

We consider our 3D prototype a good approximation of a full-scale implementation of an AQM in the *numerical* and the *computational* sense. On the other hand, a real AQM like LOTOS contains a considerable amount of I/O operations such as the reading of the meteo data (advective flow fields, temperature, wet deposition, relative humidity, etc.), the reading of the emission data and of the boundary conditions obtained from a larger model, and the logging of concentration vectors at specific times.

In [8] Dabdub and Seinfeld discuss the parallelization of the I/O in atmospheric models in a foreman (host node) / workers (slave nodes) paradigm. They state that when all computational operators have been parallelized, the I/O becomes the bottleneck, since all 'workers' must wait idle when the 'foreman' is reading and writing data to the file system. They also claim that the strategy where all workers perform their own I/O is a disastrous idea, since the performance of the I/O operations will be severely hindered by the operating system overhead, which is for instance the case for a cluster of workstations with one file-server, or for the Cray T3D, where the filesystem resides on the front-end. However, if the workstations in the cluster have their own disk or in the case of the Cray T3E where only a small number of nodes share an I/O channel (architectures with a so-called scalable I/O architecture) it is a very promising strategy to let all 'workers' perform their own I/O.

In the domain decomposition approach which we used for our AQM benchmark code all nodes perform the same task, namely the computations for a local model on a subdomain, i.e., we did not adopt the master/slave paradigm. With this implementation we studied the I/O performance of the Cray T3E for our AQM. We implemented the I/O both in the master/slave approach, i.e., one PE performs all necessary I/O and takes care of the distribution and gathering of the data, and using parallel I/O.

4.1 Description of system

For our experiments we used the Cray T3E AC80/128 of HPαC, the Centre for High Performance Applied Computing, of the Delft University of Technology. In the sequel, if we refer to the Cray T3E, we will refer to this specific configuration.

The Cray T3E has 80 DEC Alpha processors running at 300 MHz. Every processor has 128 Mb memory. Four PEs, a system building block, share one I/O port. I/O devices are connected to the I/O ports via a GigaRing [11], a 1200 Mbyte/s total raw bandwidth channel. The sustainable I/O bandwidth is 267 Mbyte/s per PE [2]. The Cray T3E has two GigaRings. However, the user file systems reside on twelve DD-308 Fibre Channel disks, all connected via one Fibre Channel Node to one GigaRing. For our I/O experiments we used the striped `/home` filesystem: sixteen physical partitions evenly divided over eight disks, in turn evenly divided over the four channels of the Fibre Channel Node. The file allocation strategy is round robin, so without any user interaction, this is supposed to create maximum I/O throughput if the system is accessing multiple files.

A single DD-308 disk drive has a sustained transfer rate of 8–12 Mbyte/s (cf. the online man page `disksfcn(7)`). Clearly, this is the limiting bandwidth.

4.2 Description of I/O in model

Since LOTOS is an off-line model we need as input (analyzed) meteo data. At given points in time we have to read at least four different three-dimensional fields: one for the temperature and three for the different components of the wind velocity. We also have to read several two-dimensional fields such as the surface pressure. In our I/O test we are using a $64 \times 64 \times 32$ computational grid. We read each time step four three-dimensional fields and five two-dimensional fields. As output we write each time step all 66 computed concentrations (three-dimensional data). The amount of I/O can be seen as a maximum. In most cases a meteo time step consists of a few integration steps and the number of output concentrations will often be smaller, say in the order of 10. In 'real models' also the amount of computational work is larger, due to operations needed on the input data and due to the addition of subgrid processes to the model.

Because of the relatively low precision of the input data and because of the relatively low precision requirements on the output data, we use `REAL*4` for the I/O. Computations are done with precision `REAL*8`. For optimal performance we use in our experiments unformatted I/O; although our I/O is sequential in nature we use direct access files to avoid record blocking; furthermore, we use maximal record sizes ($64 \times 64 \times 32 \times 4$ bytes for three-dimensional fields and $64 \times 64 \times 4$ bytes for two-dimensional fields). For the record sizes we use in our I/O requests, we actually reach 10 Mbyte/s on average when using one disk(partition).

In a separate experiment, using `assign(3f)` (the preferred Fortran library interface to the `assign(1)` command) to create a 4-way user-level striped file, we showed that it was possible to reach 40 Mbyte/s writing a three-dimensional field. Another separate experiment, using the `cachea` FFIO (Flexible File I/O [7]) layer with appropriate buffers in order to allow read-ahead, showed that it was possible to read a three-dimensional field at 40 Mbyte/s from a 4-way user-level striped file. In the previous two experiments we used 4-way striping because the Fibre Channel Node has four channels. We should be able to get a speed-up of eight, using all available disks. Using user-level striping is an easy way to increase the I/O speed. Knowing the disk configuration and using the `setf(1)` command it is even possible to initialize data files using striping before getting them on the Cray T3E.

4.3 I/O experiments

There are many possible strategies to perform I/O on a parallel computer. For the input of the meteo data we chose the most simple approach: do the input from a single PE. Clearly, this is the most portable approach: it is possible on every parallel architecture and it is not necessary to divide the data up into multiple files before reading. The disadvantage is that it does not scale. Apart from reading the data we have to reorder the data according to the domain decomposition and to scatter the data across the PEs. Not only the reading time does not scale, but also the reordering time and scatter time do not scale. To stay completely portable, we did not use user-level striping of the input files.

For the output we tried three different approaches:

1. The similar portable approach as for the input: gather all concentrations on one PE, reorder them in one three-dimensional field, and do the output from one single PE.
2. The parallel output approach: every PE writes its own subdomain concentrations to a private file using standard Fortran output. The disadvantage is that the data has to be merged outside the scope of this program. This is not necessarily a big disadvantage: if we need the data for visualization, say, then some packages are able to collect their data from multiple files.
3. The parallel asynchronous approach: every PE writes it own subdomain concentrations to a private file using asynchronous `BUFFER OUT` requests, where a PE starts an output request and immediately can continue doing computations. This approach has the same disadvantage as the previous approach. Strictly speaking, `BUFFER OUT` is not standard Fortran, so this approach is not portable.

In Table 2 we show the CPU time (column CPU) of our model without doing any I/O. For the I/O we show the minimum, average and maximum wall clock times. Column I shows the input-only wall clock time, and columns I/O, I/PO, I/PAO show the combined input/output wall clock times for the standard output, parallel output, and parallel asynchronous output approach, respectively. The wall clock times for the I/O are determined by subtracting the CPU time from the measured wall clock times including the I/O. Because of the memory requirements, it is not possible to run our simulation on less than 4 PEs.

Table 2. wall clock times in seconds for I/O on the T3E.

# PEs	CPU	I			I/O			I/PO			I/PAO		
		min	avg	max	min	avg	max	min	avg	max	min	avg	max
4	293	11.9	13.0	18.4	70.8	78.7	100.6	38.4	52.2	72.7	8.1	10.8	18.3
8	142	2.7	3.5	5.1	70.0	72.4	81.6	42.0	47.8	64.8	8.3	10.3	13.1
16	69	2.8	3.7	4.6	70.0	82.7	103.6	47.9	51.8	60.1	10.0	11.1	13.8
32	34	3.3	3.9	4.8	70.1	83.4	91.9	43.4	51.2	71.7	11.5	12.4	13.6
64	14	2.9	3.0	3.3	80.7	80.7	80.7	57.0	57.0	57.0	13.2	14.1	15.6

From Table 2 we see a significant difference between minimum and maximum times. This is because we are dealing with shared physical devices. Disks always have seek times, but some I/O requests also have to wait until requests from other jobs on other PEs are finished. Isolated timing of a single I/O request that normally takes only 0.3 s showed times up to 5 s.

For the input we see an anomalous behavior for 4 processors, where the time to read the data takes more time than the combined time for input and parallel asynchronous output. We have no explanation for this, yet. Since the time to do the input on 64 PEs only takes 20% of the wall clock time of the total job, we are not interested in investigating other options (of course parallel asynchronous input would be the winner), so we settle for the portable standard input (without even using user-level striped data files).

For the portable output we see a combined input/output time of 70 s, say. This is what we might expect: the input reads 4 three-dimensional fields (neglecting the two-dimensional fields) and here we write an additional 66 three-dimensional fields. This is 70/4 times more I/O and that is approximately reflected by the increase in the wall clock time. As we expected, the combined I/O time does not scale. Since the combined I/O time is almost three times as expensive as the CPU time on 64 PEs, we tried the following two other approaches.

For the parallel output every processor writes to its own private file so we might hope for a parallel speed-up limited by the maximum of the number of PEs and the number of disks, in our case eight. However, we only see a performance gain of 30–50% probably due to the synchronous I/O and the resulting I/O contention. On 64 PEs the combined I/O time still is four times as expensive as the CPU time.

If we compare the input times with the combined input and parallel asynchronous output times, we see that we are not able to overlap the output and computations plus necessary communications completely. The reason for this is unclear: it should have been possible looking at the amount of I/O and the sustained average I/O rates. Doing a separate experiment, where we only wrote the output files without doing any computation/communication at all (just executing a '`sleep`' statement), we were able to overlap output and 'computations' completely.

Again we might have hoped for a parallel speed-up limited by the maximum of the number of PEs and the number of disks, but we do not see a performance gain going from 4 to 8 PEs. We also see that the combined I/O time slightly increases as the number of PEs increases. This must be due to I/O contention. On 64 PEs the combined I/O time is about as much as the CPU time. This seems acceptable, especially since 64 PEs give rise to small subdomains with a maximum amount of I/O. Moreover, it is the most efficient way to handle the output.

The final conclusion is probably the most disappointing one: although the Cray T3E has a scalable I/O architecture, the I/O does not scale. Of course this is due to the limited number of disks, but on the other hand we should have seen a speed-up going from 4 to 8 PEs when doing output.

5 Discussion

In this paper we evaluated the performance of our AQM benchmark code on three different computer platforms: a vector/parallel shared memory architecture, a massively parallel distributed memory system, and a cluster of workstations. The AQM benchmark [1] is composed of the following components:

- A 3D prototype of an atmospheric transport/chemistry model consisting of horizontal advection, vertical diffusion, and a state-of-the-art ozone chemistry scheme.
- Numerical methods aiming at positivity, mass conservation, stability, and efficiency. The methods used in the code are either the ones we actually will use in our full model or which are typical with respect to computational and memory requirements for numerical methods used in AQM's.
- A code based on the SPMD approach for parallelization. The horizontal domain of the regional model is partitioned and the submodels are distributed over the PEs. The underlying program is implemented using array syntax constructs to facilitate automatic vectorization and possibly also parallelization.

The results of our evaluation are not surprising: For real computer speed one should use a dedicated shared memory vector/parallel architecture or a distributed memory architecture in case of memory constraints. Both are expensive. Much cheaper and somewhat competitive is a number of coupled workstations.

We want to emphasize that adding diffusion or vertical advection will not influence the trend of our findings as long as the horizontal processes will be calculated explicitly.

The second part of our paper discusses possible implementations on the Cray T3E of the necessary I/O in a real AQM. In contrast to the expectations raised by the advertising slogan 'the Cray T3E has a scalable I/O architecture', I/O does not scale on the T3E. Scalability is of course limited by the number of disks (8 in our case), but we do not even find a speed-up going from 4 to 8 PEs when doing output. There is also an anomalous behavior doing only input on 4 PEs, taking more time than combined input/output on 4 PEs. Furthermore, we are not able to fully exploit asynchronous I/O in the sense that we are not able to fully overlap I/O and computations/communications, whereas I/O and '`sleep`' overlap completely. Clearly, as on most platforms, asynchronous I/O is the most efficient way to do I/O on the Cray T3E.

With respect to the implementation of the full 3D LOTOS model we draw the following conclusions from the experiments described in this paper:

- To avoid divergence of different implementations aimed at different computer platforms it is highly recommendable to have one implementation of LOTOS. The experiments with our benchmark code on various platforms show that this is possible without loosing efficiency.
- I/O experiments on the Cray T3E show that
  - The necessary time to read the input data for LOTOS will be small compared to the computational time.
  - On the other hand, the output can have a significant influence on the 'through-put' time if one really wants to write all concentrations at every time step to file, which is for instance the case when an AQM is coupled to a powerful visualization / steering tool.

– The (almost) portable implementation of parallel asynchronous I/O will be an efficient choice on all architectures and will result in negligible I/O times on 'shared nothing' architectures where every processor has its own memory and its own disk.

Acknowledgements

Financial support for this research was provided by Cray Research, Inc., under Grant CRG 97.02. This work was partially sponsored by the Stichting HPCN with financial support from the Ministry of Economic Affairs through the HPCN-TASC project 'HPCN for Environmental Applications'. Computer time was provided 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).

References

1. `http://www.cwi.nl/~gollum/LOTOS/`.
2. E. Anderson, J. Brooks, C. Grassl, and S. Scott. Performance of the Cray T3E multiprocessor. In *Proceedings Supercomputing '97, San Jose, California, November 15–21*. IEEE Computer Society Press, 1997. Available from `http://scxy.tc.cornell.edu/sc97/program/TECH/ANDERSON/INDEX.HTM`.
3. J.G. Blom, Ch. Kessler, and J.G. Verwer. An evaluation of the Cray T3D programming paradigms in atmospheric chemistry/transport models. Report NM-R9604, CWI, Amsterdam, 1996.
4. J.G. Blom, W.M. Lioen, and J.G. Verwer. HPCN and air quality modeling. Report MAS-R9801, CWI, Amsterdam, 1998.
5. J.G. Blom and M.G.M. Roemer. Description of the 3D LOTOS model. Part I: Dynamics. Report MAS-N9701, CWI, Amsterdam, 1997.
6. P.J.H. Builtjes. The LOTOS – Long Term Ozone Simulation – project. Summary report. TNO-rapport TNO-MW – R 92/240, TNO Milieuwetenschappen, Delft, 1992.
7. Cray Research, Inc. *Applications Programmer's I/O Guide*, 1997. Publication SG-2168 3.0.
8. D. Dabdub and J.H. Seinfeld. Practical aspects and experiences. Parallel computation in atmospheric chemical modeling. *Parallel Computing*, pages 111–130, 1996.
9. L.K. Peters et al. The current state and future direction of Eulerian models in simulating the tropospheric chemistry and transport of trace species: A review. *Atmospheric Environment*, 29:189–222, 1995.
10. 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.
11. S. Scott. The GigaRing channel. *IEEE Micro*, 16(1):27–34, February 1996.