# CHOOSING A COMPILER FOR AQM APPLICATIONS ON LINUX

George Delic *
HiPERiSM Consulting, LLC, Durham, NC
e-mail: **george@hiperism.com**
Web address: **http://www.hiperism.com**
Voice (919) 484-9803 Fax (919) 806-2813

## 1. INTRODUCTION

This is a status report on a project to evaluate industry standard fortran 90/95 compilers for IA-32 Linux™ commodity platforms when applied to Air Quality Models (AQM). There are several motivating factors for such a project:

1) large-scale scientific codes continue their migration to commodity hardware,
2) Linux™ has emerged as an alternative to proprietary UNIX operating systems,
3) the number of Fortran compilers to chose from in the IA-32 market sector has grown,
4) portability issues arise when moving legacy code into this environment.

In the case of both Air Quality Models, and Meteorological Models, important additional considerations include the time to solution and questions of numerical stability and accuracy of the solution. Thus it seems that a project such as this is timely. To substantiate this claim it need only be noted that, in the case of CMAQ, between the 4.2.1 and 4.2.2 Linux™ releases, there was a switch of compilers with no information, motivation, or discussion of numerical and performance issues.

Here, as a preliminary step, two simple benchmarks are used in the evaluation of compilers before launching into a full-scale AQM such as CMAQ. This approach has the advantage of identifying portability issues and compiler quirks. Also, a controlled empirical study will quickly lead to understanding which compiler switches affect performance and accuracy.

## 2.0 CHOICE OF HARDWARE AND OPERATING SYSTEM

Results for the wall clock time are compared for two benchmarks compiled using three different Fortran compilers with the Linux™ operating system and one with Windows 2000 (because the Linux™ version was not yet installed). For this project benchmarks were executed in serial mode on a dual processor Intel™ Pentium III 933MHz workstation with a 256MB on-processor L2 cache (this is the Flip Chip PGA version of the Pentium III). This architecture offers Streaming Single-Instruction-Multiple-Data Extensions (SSE) to enable vectorization of loops operating on multiple elements in a data set with a single operation. Where compilers specifically enable SSE it has been tested.

## 3.0 CHOICE OF COMPILERS

The choice of compilers for Linux™ IA-32 platforms now includes several vendor-supported products. The importance of this category is that vendor products have technical support and undergo continuous development with ports to new architectures as they arrive in the marketplace. The four compilers chosen in this survey are described separately in the following sections and compiler switches used in the two benchmarks are also discussed. However, it is noted here that while all compilers offer a switch to target the Pentium III, only two (Intel and Portland) offer the SSE option.

### *3.1 Absoft*

Absoft f77 and f90/f95 are the Fortran compilers included in the Absoft Pro Fortran™ 8.0 package for Linux™ offered by the Absoft Corporation (http://www.absoft.com). The f90/f95 version has a Cray front-end and resulted from a five-year collaboration with Cray Research.

### *3.2 Intel*

The Intel Fortran Compiler version 7.1 targets both Intel IA-32 and IA-64 (Itanium) architectures, but only the former has been used in this project

* *Corresponding author address:* George Delic, HiPERiSM Consulting, LLC, P.O. Box 569, Chapel Hill, NC 27514-0569

so far. A license for a non-commercial version is available from the Intel URL at http://www.developer.intel.com/software/products/compilers

### *3.3 Lahey*

The Lahey/Fujitsu Fortran 95 compiler (hereafter Lahey) for Linux™ is available from Lahey Computer Systems, Inc., (http://www.lahey.com). The Express version 5.6 for Microsoft Windows 2000™ was used because it was available from another project for the same hardware.

### *3.4 Portland*

The pgf90™ fortran compiler (Linux™ distribution) from the Portland Group, (http://www.pgroup.com) was used in the CDK 4.0 release where it supports OpenMP, MPI and OpenMP+MPI parallel applications on HiPERiSM's IA-32 Linux™ cluster.

### *3.5 Portability issues*

Portability issues come up when legacy Fortran code needs to be compiled. In this respect a compiler that allows extensions to the f90/f95 standard can save time and effort. The two compilers that offer the widest scope in portability are those from Absoft and Portland. Compilers from Lahey and Intel are less forgiving of such extensions. For example one of the benchmarks used logical operators such as the IAND, IOR, and NOT intrinsic functions that are now part of the Fortran 90 standard and require integer operands. The older FORTRAN 77 standard .AND., .OR., .NOT. for integer operands no longer applies under Fortran 90 because these operators are reserved for logical operands. Nevertheless, different compilers apply different extensions to the standard and two of the compilers discussed below (Absoft and Portland) compile the older FORTRAN 77 standard (with the default Fortran 90 options) without comment. Whereas the Lahey compiler does now allow the extensions and reports compiler errors if they are used. On the other hand the Intel compiler issues warnings but nevertheless produces an executable that generates erroneous results. Clearly, caution should always be applied in porting legacy code with any compiler.

## 4.0 CHOICE OF BENCHMARKS

The algorithms used here have been executed on a wide variety of platforms and are excellent benchmarks in studying how a compiler and architecture interact for the types of operation they use. A fuller discussion of the two benchmarks is available at the HiPERiSM URL. What follows is only a brief introduction.

### *4.1 Kallman Algorithm*

The Kallman algorithm computes the permanent of a (0,1) matrix with high efficiency using only integer and logical operations and some of the MIL-STD-1753 bit intrinsic functions that are now part of Fortran 90/95. There is no floating point work in the Kallman algorithm. A fuller discussion of results is given by Delic and Cash (2000). This algorithm is CPU intensive and performs a small amount of I/O only at the beginning and end of each run. Memory requirements are modest and because of the small instruction set, the instruction buffer fetch rates are amongst the smallest we have seen. This algorithm runs in scalar mode because of a complex branching structure that inhibits vectorization. Six cases where used in this analysis corresponding to data sets with matrix sizes N=30, 44, 48, 52, 56, 60.

### *4.2 Stommel Ocean Model Algorithm*

The Stommel Ocean Model (SOM) is a legacy Fortran 77 code with a compute kernel consisting of a double-nested loop that performs a Jacobi iteration sweep over a two-dimensional finite difference grid. The number of iterations is fixed at 100 and, because the data set is regular and the loop structure is conventional, this code should present compilers with good prospects for vectorization. Therefore, as a floating point algorithm, the SOM is useful in studying performance scaling with problem size N over the N x N grid. Six cases where used in this analysis for data sets with grid dimensions in the range N=2000 (1000) 7000, corresponding to problem size scaling from $N^2=4x10^6$ to $49x10^6$ data points.

## 5.0 COMPARING EXECUTION TIMES

The following sections summarize execution time with four compilers for the Kallman and SOM algorithms for their respective data sets.

### *5.1 Timing performance*

Whole code execution was measured with the Linux™ time command. This choice was due in part to the problem of portable timing procedures in the different compilers. While a f90 system_clock routine could have been used the time command introduced an error of approximately 2% and was therefore deemed to be of sufficient accuracy for these simple benchmarks.

### *5.2 Kallman Algorithm results*

For the Kallman algorithm the choice of compiler switches is summarized in Table 5.1 and timing results are shown in Table 5.2. Figure 1 shows the ratio of these times to the Absoft compiler (which reports the smallest execution time) for the six cases of Table 5.2.

Table 5.1 Compiler command and switches for the Kallman algorithm

| Compiler and version | Compiler command and selected switches |
|---|---|
| Absoft 8.0 | f90 –O3 –ffixed |
| Intel 7.1 | ifc –O3 –tpp6 -FI |
| Lahey 5.6 | lf95 –tpp –fix |
| Portland 4.0 | pgf90 –fast |

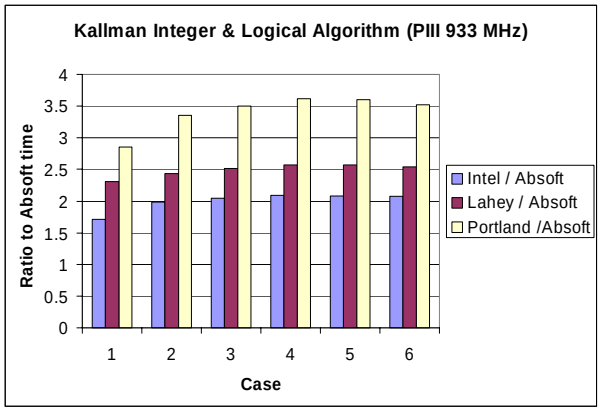


Fig. 1 Ratio of execution times of three different compilers to that for the Absoft compiler with the Kallman algorithm.

Table 5.2 Execution times (seconds) for the Kallman algorithm with four compilers on the Pentium III (933 MHz).

| N | Absoft | Intel | Lahey | Portland |
|---|---|---|---|---|
| 30 | 0.21 | 0.36 | 0.48 | 0.6 |
| 44 | 40.38 | 80.19 | 98.45 | 135.29 |
| 48 | 6.44 | 13.15 | 16.16 | 22.52 |
| 52 | 23.03 | 48.20 | 59.30 | 83.28 |
| 56 | 197.78 | 412.83 | 509.31 | 712.42 |
| 60 | 12891.58 | 26734.09 | 32833.08 | 45451.38 |

### *5.3 Stommel Ocean Model results*

For the SOM algorithm the choice of compiler switches is summarized in Table 5.3 and timing results are shown in Table 5.4 (without SSE enabled). Figure 2 shows the times of Table 5.4 and Figure 3 shows the trend in descriptive statistics for the six problem sizes. As expected the mean and standard deviation of the execution time rise. However, the coefficient of variation (standard deviation divided by the mean) of execution times within this group of compilers changes slowly as problem size increases. For the largest problem size the difference in execution time for different compilers diminishes.

Table 5.3 Compiler command and switches for the SOM algorithm

| Compiler and version | Compiler command and selected switches | Effect of switches |
|---|---|---|
| Absoft 8.0 | f90 –s –cpu:p6 –O3 –ffixed | Static and PIII Pro target |
| Intel 7.1 | ifc –O3 –tpp6 –FI<br><br>ifc –O3 –xK –tpp6 –FI | Optimize for PIII target<br>Vectorize and enable SSE |
| Lahey 5.6 | lf95 –tpp -fix | |
| Portland 4.0 | pgf90 –fast –Mvect<br>pgf90 –fast –Mvect=sse | Vectorize<br>Enable SSE |

## 6.0 EVALUATION OF SSE RESULTS

Two of the compilers (Intel and Portland) include specific switches to enable the SSE feature of the Pentium III architecture. For regular data structure and vectorizable loops this should produce enhanced performance on this generation of processors.

Table 5.4 Execution times (seconds) for the SOM algorithm with four compilers on the Pentium III (933 MHz) without SSE enabled.

| N | Absoft | Intel | Lahey | Portland |
|---|---|---|---|---|
| 2000 | 50.0 | 38.8 | 36.4 | 41.4 |
| 3000 | 110.5 | 94.4 | 87.7 | 92.7 |
| 4000 | 197.7 | 159.6 | 150.3 | 163.3 |
| 5000 | 305.3 | 224.3 | 246.8 | 253.1 |
| 6000 | 443.4 | 320.0 | 332.0 | 388.5 |
| 7000 | 586.5 | 427.6 | 477.9 | 524.4 |

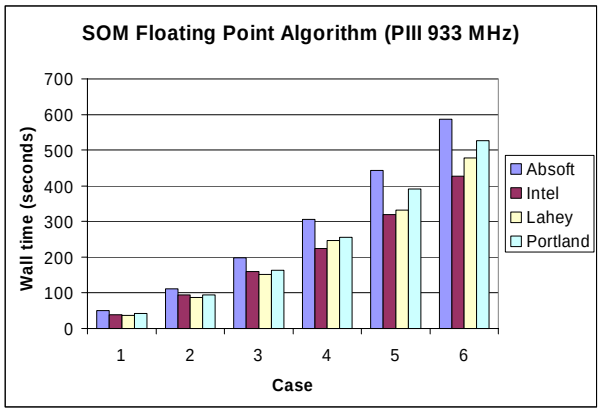


Fig. 2 Execution times of four different compilers for the SOM floating point algorithm (without SSE).

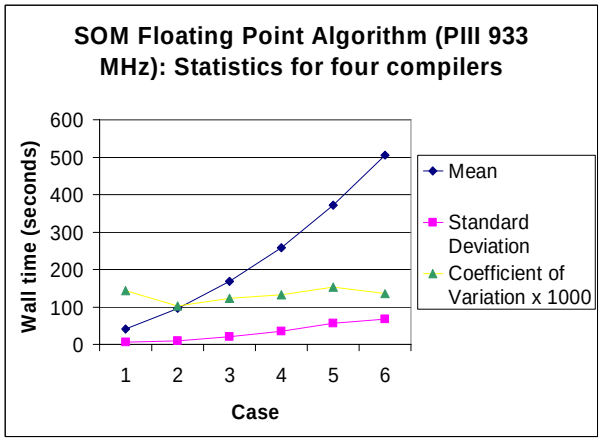


Fig. 3 Execution time statistics for four different compilers with the SOM floating point algorithm.

The SSE options are enabled as indicated in Table 5.3 for the SOM floating point algorithm (note that the SSE option is irrelevant for the Kallman integer and logical algorithm). Figure 4 summarizes the effect of the SSE for the Intel and Portland compilers. It should be noted that whereas the Portland compiler includes vector instructions with the –Mvect switch alone, the Intel compiler seems to produce vector instructions on loops only when the SSE option is enabled. The results of Figure 4 show that for regular data structures and vectorizable code with long loops dramatic performance enhancements are possible from enabling SSE where it is available.

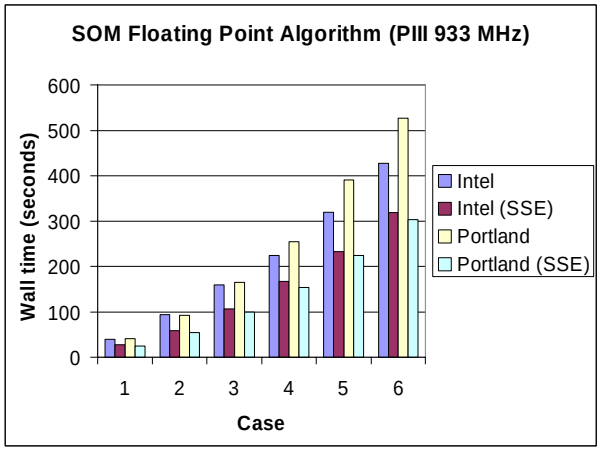


Fig. 4 Execution times of two compilers for the SOM floating point algorithm without and with SSE enabled.

## 7.0 TESTS FOR AIR QUALITY MODELS

The next phase in this project is to evaluate performance of this group of compilers for the CMAQ 4.3 release. Variability in performance results is expected but the details will depend on the balance of integer, logical, and floating point operations. Also in this phase the consequences of compiler switches for numerical precision and stability will be investigated. In testing CMAQ, community proposals for which scenarios and species are of particular interest, are welcome.

## 8.0 CONCLUSIONS

This report presented performance results of four fortran compilers in the IA-32 environment. The variability in performance found was specific to the two benchmarks selected and represented two extremes in arithmetic operation types. Real-world codes have different mixtures of such operations. Therefore an evaluation of the same group of compilers for real-world environmental models is expected to produce different results.

Delic, G.and Cash, G., 2000: The Permanent of 0,1 Matrices and Kallman's Algorithm, *Comput. Phys. Comm.*, **124**, 315-329.