COMPARING COMPILERS ON INTEL™ PENTIUM 3 and PENTIUM 4 PROCESSORS WITH THE POM FLOATING POINT ALGORITHM

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 part of a series of reports on a project to evaluate industry standard fortran 90/95 compilers for IA-32 Linux™ commodity platforms. This report shows results, in a side-by-side comparison for each compiler, for the Intel™ Pentium 3 (P3) and Pentium 4 Xeon (P4) processors for the Princeton Ocean Model.

2.0 CHOICE OF HARDWARE AND OPERATING SYSTEM

Results for the wall clock time are compared for benchmarks compiled using four 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 (256KB L2 cache) and a dual processor Pentium 4 Xeon 3.06GHz (1MB L3 cache). These architectures offers Streaming Single-Instruction-Multiple-Data Extensions (with version 2, SSE2, for the Xeon). This enables vectorization of loops operating on multiple elements in a data set with a single operation. Where compilers specifically enable SSE/SSE2 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 benchmarks are also discussed. However, it is noted here that while all compilers offer a switch to target the Pentium 4, only three (Intel, Lahey, and Portland) offer a specific SSE/SSE2 option (see also notes below).

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. With this compiler use of the –O3 compiler switch enables automatic architecture detection and selection of the Pentium 3 or 4 instruction set.

3.2 Intel

The Intel Fortran Compiler version 8.0 targets both Intel IA-32 and IA-64 (Itanium) architectures, but only the former has been used in this project so far. Details on the compiler features are available at HiPERiSM Consulting, LLC's URL. Code for target architectures are generated with either the –tpp6 (Pentium 3) or –tpp7 (Pentium 4) switch.

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 on the Pentium 3 because it was available from another project for the same hardware. With this compiler use of the –tpp compiler switch enables automatic architecture detection for the P3 only. However, release v7.1 (for Windows) and v6.2 (for Linux) support compiler switches –tp4 and –sse2 to target the Pentium 4 Xeon and the SSE2

instruction set. The v6.2 release and the new switches are studied in this report.

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. With this compiler use of the –fast compiler switch enables automatic architecture detection. Note that the CDK 5.1 release (not used here) may offer additional performance enhancement of the Pentium 4 Xeon processor with the use of SSE2 options.

3.5 Portability and migration 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.

Here we also mention some migration issues that came up with compiler and architecture changes. The change in architecture from P3 to P4 Xeon also involves changes in library versions. As a result, two of the compilers had to either be upgraded or have patches applied. Installation of the Absoft 8.0 compiler for the Xeon processor and the newer Linux Kernel does require download and application of two patch files to resolve glibc version issues (these patch files are available from the Absoft URL given in Section 3.1). Likewise, an attempt was made to install the 7.1 release of the Intel Fortran compiler on the P4 Xeon. However, again version skew with glibc suggested the simpler option of installing the 8.0 release. Whenever the version of a compiler is changed performance is also expected to change. This is especially true of the Intel compiler since major performance improvements are announced with the 8.0 release. Therefore, the changes in performance reported here for the Intel compiler are due to improvements in the compiler technology as well as the change in architecture.

4.0 CHOICE OF BENCHMARKS

4.1 Introduction

The Princeton Ocean Model (POM) algorithm is used here and has been executed on a wide variety of platforms. The serial version is used here in studying how a compiler and architecture interact for a real-world model that was optimized for performance on vector register machnines. A fuller discussion of the POM (in an MPI version) is available at http://www.hiperism.com/hc_6_10v30.htm. What follows introduces only the essentials of the cases studied here.

4.2 Princeton Ocean Model Algorithm

The Princeton Ocean Model (POM) is a legacy Fortran 77 code with compute kernels consisting of over three hundred vectorizable loops. Typically these are triple-nested loops (i,j,k) that perform operations over a three-dimensional finite difference grid. The vertical zones over the k range form the outermost loop in the nest. The number of iterations varies with the choice of data set as shown in Table 1. For the choices shown here, the k range is constant while the two inner loops scale substantially. The inner loop structure is conventional and this code should present compilers with good prospects for vectorization. Two important features of the POM should be noted: (a) the algorithm is unstable in single precision arithmetic and therefore double precision is used for all compilers, and (b) long integers are required and this option must be specifically requested with the Lahey compiler which otherwise produces run time errors.

Table 4.1 Problem sizes and scaling for the POM algorithm.

GRID	i_{max}	j_{max}	k_{max}	Scaling
1	100	40	15	1
2	128	128	16	4.37
3	256	256	16	17.47

5.0 COMPARING EXECUTION TIMES

The following sections summarize execution time with four compilers for the POM algorithm with the three data sets of Table 4.1 (GRID 1 to 3).

5.1 Timing performance

Whole code execution was measured with calls to the Fortran 90/95 system_clock routine for

all compilers as this was deemed to be the most portable and accurate timing method.

5.3 Princeton Ocean Model results

For the POM algorithm the choice of compiler switches is summarized in Table 5.1. Note the use of the target architecture switches (often these are implicit in the optimization level). Timing results (without SSE enabled) are shown in Tables 5.2 (Pentium 3) and 5.3 (Pentium 4). Figures 1 and 2, for Pentium 3 and Pentium 4 respectively, show these times as bar charts. For the largest problem size the Lahey compiler is noticeably less efficient than the others and this is due to the requirement of the --long option for large integers.

Table 5.1 Compiler command and switches for the POM algorithm on the P3 and P4 Xeon processors.

Compiler and version	Compiler command and selected switches	Effect of switches
Absoft 8.0 (P3), Absoft 8.0 (P4)	f90 –s –cpu:p6 –O3 –N113 –ffixed f90 –s –cpu:p7 –O3 –N113 –ffixed	Optimize for P3 or P4 Xeon target
Intel 7.1 (P3) Intel 8.0 (P4)	ifc –O3 –r8 –tpp6 –FI ifc –O3 –r8 –xK –tpp6 –FI ifort –fast –r8 –tpp7 –FI ifort –fast –r8 –xW –tpp7 –FI	Optimize for P3 Vectorize and enable SSE. Optimize for P4 Xeon target. Vectorize and enable SSE2.
Lahey 5.6 (P3) Lahey 6.2 (P4)	lf95 –long –tpp –fix –dbl lf95 --long --O2 --tp4 --fix --dbl lf95 --long --O2 --tp4 --sse2 --fix --dbl	Optimize for P3 target. Optimize for P4 target. Enable SSE2.
Portland 4.0 (P3 and P4)	pgf90 –fast –Mvect –r8 pgf90 –fast –Mvect=sse –r8	Vectorize Enable SSE

Table 5.2 Execution times (seconds) for the POM algorithm with four compilers on the Pentium III (933 MHz) without SSE enabled.

GRID	Absoft	Intel	Lahey	Portland
1	825.9	786.9	805.3	755.3
2	4412.7	4234.6	4795.7	3736.1
3	16761.0	16895.4	19721.9	15814.3

Table 5.3 Execution times (seconds) for the POM algorithm with four compilers on the Pentium 4 Xeon (3.06 MHz, 1MB L3 cache) without SSE enabled.

GRID	Absoft	Intel	Lahey	Portland
1	167.7	156.1	189.5	190.1
2	1925.9	1518.9	2809.7	1756.7
3	8685.2	7432.3	12731.8	8764.8

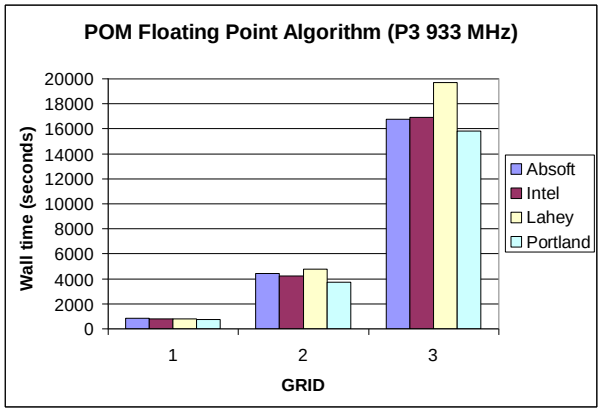

Fig. 1 Execution times of four different compilers for the POM floating point algorithm (without SSE) on the Pentium 3.

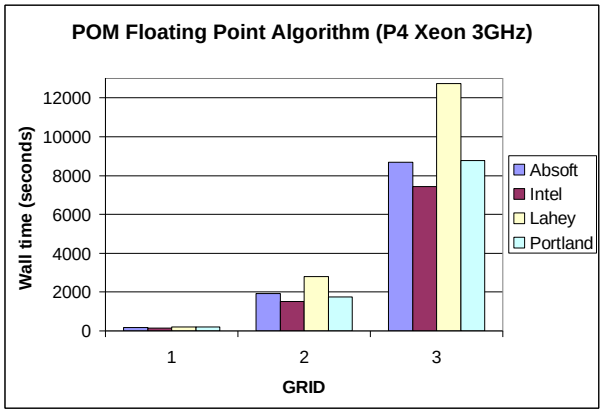

Fig. 2 Execution times of four different compilers for the POM floating point algorithm (without SSE) on the Pentium 4 Xeon.

It is interesting to observe the changes in performance between the Pentium 3 and 4. As expected the execution time rises for both processors as the problem size scales.

On the Pentium 3 the spread in performance is relatively small (with the exception of the Lahey

compiler result for GRID 3). The Portland compiler reported the lowest wall clock times for all three problem sizes. On the Pentium 4 the situation differs in two respects: (a) the Lahey compiler offers the longest execution times (noticeably for GIRD 2 and GRID 3), and (b) the Intel compiler delivered the shortest execution times for all three problem sizes. Hence, the variability in compiler performance is, in general, greater for the Xeon processor when compared to the P3.

To compare the relative performance gain for each compiler due to a change in architecture Figure 3 shows the ratio of the Pentium 3 execution times to those of the Pentium 4. For all four compilers the performance gain is in the range 4 to 5 for GRID 1, but declines to a range of 1.7 to 2.8 (GRID 2) and 1.6 to 2.3 (GRID 3). Since the trend in performance degradation is the same for all four compilers, it is surmised that this is due to increases in cache misses and enhanced memory accesses. In particular, the poor results for GRID 3 suggest that the largest problem size experiences relative performance loss due to memory bandwidth limitations since the memory traffic increases substantially as the problem size increases. This result is surprising in view of the sophisticated compiler optimizations employed by all four compilers. Presumably the additional layer of L3 cache on the P4 Xeon requires careful hand-tuning despite the deep vector character of the two inner loops (i,j). A deeper performance analysis of the underlying reasons for this behavior is the subject of a future report. Memory bandwidth issues on both the P3 and P4 nodes are the subject of a separate report in this series.

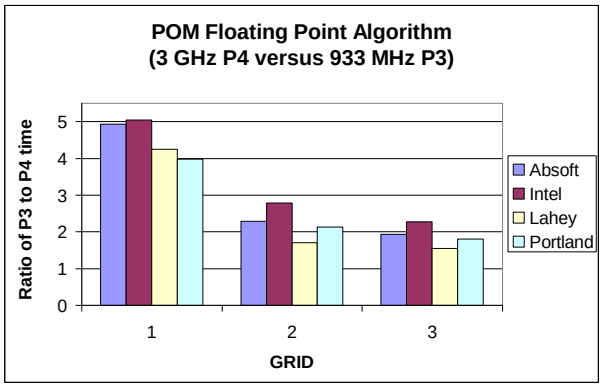

Fig. 3 Ratio of execution times of four different compilers on the Pentium 3 versus the Pentium 4 Xeon for the POM floating point algorithm (without SSE).

6.0 EVALUATION OF SSE RESULTS

Three of the compilers (Intel, Lahey, and Portland) include specific switches to enable the SSE2 feature of the Pentium 4 architecture. Two of these (Intel and Portland) also were used on the Pentium 3 for SSE testing. For regular data structure and vectorizable loops enabling these instruction sets should produce enhanced performance on this generation of processors.

The SSE options are enabled as indicated in Table 5.1 for the POM floating point algorithm. Figures 6 and 7 for Pentium 3 and 4, respectively, summarize the effect of the SSE for these compilers. These results show performance improvements from SSE instructions on both P3 and P4 processors. However, the results of Figure 7 for the Xeon processor show that performance enhancements from SSE are typically larger than those on the P3.

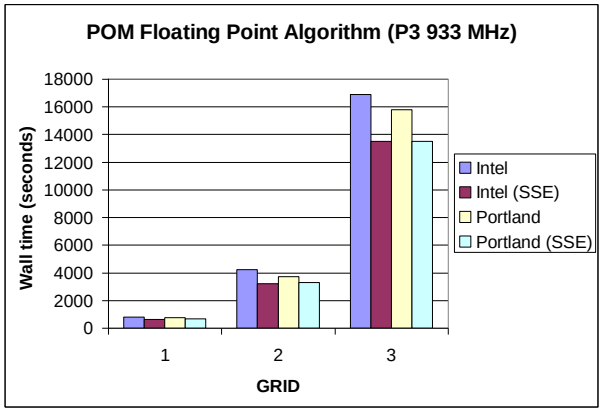

Fig. 6 Execution times of two compilers for the POM floating point algorithm without and with SSE enabled on the Pentium 3.

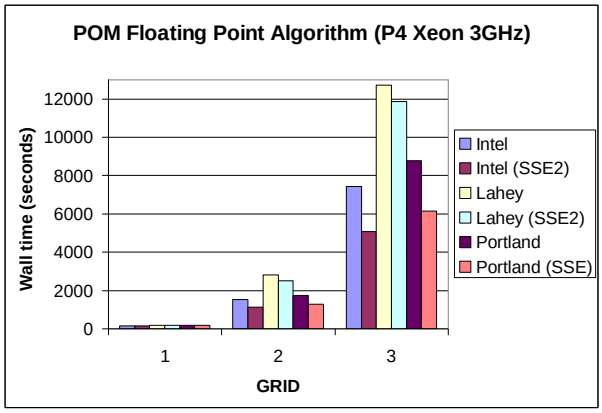

Fig. 7 Execution times of three compilers for the POM floating point algorithm without and with SSE enabled on the Pentium 4 Xeon.

A more precise comparison of SSE performance gains for each compiler tested here is shown in Table 6.1. This table shows the relative gain (SSE versus no SSE) as a percentage for each of the three problem sizes on both processors.

Table 6.1 Percentage gain in execution times for the POM algorithm with three compilers when SSE/SSE2 is enabled on Pentium 3 and 4 processors, respectively.

GRID	Processor	Intel	Lahey	Portland
1	Pentium 3	19.2	--	12.9
2	Pentium 3	23.6	--	11.4
3	Pentium 3	19.9	--	14.5
1	Pentium 4	1.2	-0.3	11.2
2	Pentium 4	25.5	10.3	27.6
3	Pentium 4	31.5	6.7	29.8

The largest gains in performance are for the largest problem size on the P4 Xeon processor, where it is clear from Fig.7, that the Intel compiler delivers the best performance once SSE is enabled. Nevertheless, this result for the largest problem size cannot be considered entirely satisfactory since it represents a performance gain of only 2.66 when the p4 (Intel 8.0) is compared with the P3 (Intel 7.1). Therefore, a deeper study of the causes for this mediocre performance gain with the newer hardware and compiler technology seems in order.

7.0 CONCLUSIONS

This report presented performance results of four fortran compilers in the IA-32 environment. The variability in performance found was specific to the problem sizes selected and represented extremes in cache and memory access operation types. As an example of a “real-world code” the Princeton Ocean Model reveals that performance gains depend in a similar way on the choice of problem size for all compilers. In particular, when the problem size is sufficiently large, there is a sharp decline in performance gain over earlier hardware and compiler generations for all the compilers discussed here. This suggests that considerable optimization work still remains to be done by the end user of cache-based architectures even when code with good vector structure is presented to compilers with powerful optimization strategies.

The analysis in subsequent reports will include in-depth evaluation of performance of this group of compilers with specialized software such as the Intel VTune™ Performance Analyzer. Also in this evaluation the consequences of compiler switches for numerical precision and stability will be investigated.