Stanford Exploration Project, Report 102, October 25, 1999, pages 187–248

Testing Linux multiprocessors for seismic imaging

Biondo Biondi, Robert G. Clapp, and James Rickett[1]

keywords: *Linux, parallel computers, Fortran 90, OpenMP*

ABSTRACT

Benchmarks of our "production" codes show that Pentium-based multi-processors computers running the Linux operating system are attractive options for SEP's next computational server(s). A four-processor SGI 1400 L (500 Mhz Pentium III Xeon) has consistently performed as well as, or better than, our SGI Origin 200 (180 Mhz R10000). We used the Portland Group's `pgf90` compiler to compile our benchmarking codes, that are parallelized with OpenMP directives. This compiler has proven to generate efficient code, though the support of some F90 and OpenMP features is still immature.

INTRODUCTION

SEP's present computer servers (18-processor Power Challenge and 4-processor Origin 200) fall short from delivering the computer power needed to perform research in advanced algorithms for 3-D wave-equation migration, 3-D velocity estimation, and 3-D wavefield interpolation. We are therefore evaluating the choices for our next computer server (or servers). Computers based on commodity processors (Intel) have attractive prices, and seem to have finally caught up in floating-point performances with computers based on processors specialized for floating-point computations (SGI-MIPS, SUN-Ultra, etc). Further, memory and disk-storage are much better priced for Intel-based computers than for any other. We are thus evaluating Intel-based Linux multi-processors systems. At the moment the choice is between systems based on dual-processor Pentium III and systems based on four-processor Pentium III Xeon. In the near future (end of 1999?), it should be also possible to purchase eight-processor Pentium III Xeon system. We evaluated a dual-processor Pentium III marketed by VA Linux Systems (StartX MP Workstation) and a four-processor Pentium III Xeon kindly loaned to us for evaluation by SGI (SGI 1400L).

The main goal for the tests that we report here is to determine whether we can run efficiently our "production" codes on multi-processors Linux systems. For several years now SEP has operated Linux computers as desktops. We are satisfied by the experience, to the point that all our desktops are Intel PC's running Linux. However, we perform the heavy-duty parallel computations on our SGIs. One of the authors (JR) performed some tests running

[1]**email:** biondo@sep.stanford.edu

parallel programs across our network of Linux desktop PC's using PVM message passing as a parallel-programming tool. However, no parallel "production" codes runs across the Linux network. Our question has a software component as well as a hardware component. First, Linux kernels before 2.2 had notoriously poor performance when running multi-threaded applications, and even for 2.2 the overhead of starting new threads is higher than on Irix (Bee Bednar, private communication). Second, the most of our production codes are parallelized with a shared-memory model, and the parallelism is achieved using SGI or OpenMP compiler directives. The F77/F90 compiler that we presently use on Linux (from NAGWare) does not support these parallel-programming style. Therefore, we need to look at alternative compilers; we evaluated the Portland Group F90 compiler.

We run our benchmarks on four computers:

1) 18-CPU - SGI Power Challenge
   - CPU: 75 Mhz - MIPS R8000
   - L2-Cache: 75 Mhz - 4MB
   - Memory: 2 GB
   - Operating System: Irix 6.5
   - Compiler: MIPSpro f90 version 7.2.1
2) 4-CPU - SGI Origin 200
   - CPU: 180 Mhz - MIPS R10000
   - L2-Cache: 120 Mhz - 1MB
   - Memory: 512 MB
   - Operating System: Irix 6.5
   - Compiler: MIPSpro f90 version 7.2.1
3) 4-CPU - SGI 1400L
   - CPU: 500 Mhz - Intel Pentium III Xeon
   - L2-Cache: 500 Mhz - 2MB
   - Memory: 1 GB
   - Operating System: Linux 2.2.5 (Red Hat 6.0)
   - Compiler: PGI pgf90 version 3.1-1
4) 2-CPU - VA Linux StartX MP Workstation
   - CPU: 500 Mhz - Intel Pentium III
   - L2-Cache: 250 Mhz - 512KB
   - Memory: 256 MB

- Operating System: Linux 2.2.7 (Red Hat 6.0)
- Compiler: PGI pgf90 version 3.1-1

Hardware-wise we were interested both in the relative performance of Intel-based computers compared with MIPS-based computers, as well as the absolute parallel efficiency of Intel-based computers when running multi-threaded applications. In particular, we tried to analyze the following issues regarding Intel-based multiprocessor performances:

- Floating-point performance.
- Effects of different secondary cache sizes (2MB and 512KB) and speeds (255 Mhz and 500 Mhz) on performance of production runs.
- Problems caused by memory-access conflicts when running multi-threaded applications.

To answer these questions in a context as much relevant as possible with our environment, we run three different programs that cover the most of the types of computations that we routinely perform: a FFT-intensive split-step migration, a Kirchhoff migration, and an implicit finite-difference migration.

SPLIT-STEP MIGRATION

This test was performed using the Gendown3D package that has been used to perform common-azimuth migration (Biondi, 1999) and wave-equation migration velocity analysis (Biondi and Sava, 1999). The package has been developed to facilitate the implementation of a wide range of frequency-domain depth-continuation operators, their adjoint operators, and their related scattering operators (Biondi and Sava, 1999). It is targeted to coarse-grain shared-memory multi-processors architectures, like our Power Challenge, and achieves good parallel performances. However, flexibility of the package and simplicity of the module implementing the individual operators had higher priority than efficiency. No attempt at all was made to optimize the single-processor performance for cache-based systems.

The package is multi-threaded across frequencies for each depth-continuation step. Each thread performs computations independently on a different slice of the data, including the FFTs and the application of the phase-shift operator. The summation over frequency is performed serially, as well as the I/O for both the data and the velocity function. The velocity function is shared (not replicated) between threads.

A few changes needed to be done to the code to make it run correctly on the PGI compiler. The most significant, and annoying of them, was related to F90 allocatable arrays. The PGI compiler requires the program to allocate these arrays when passed as arguments to lower level subroutines, even when they are never used in the computations. A simple work-around, to avoid allocating more memory than necessary, is to allocate these arrays with the axes length of one when they are not used.

To analyze the effects of out-of-cache computations, we run the benchmarks on two different sized problems. A "small problem", for which two frequency slices (one for the input and one for the output) fitted into the L2-caches of all computers tested, and a "large problem" for which two frequency slices did not fit into the L2-caches of any of the computers tested.

On the SGI platforms, the package usually uses the FFTs provided by SGI mathematical libraries. Obviously, these libraries are not available (yet?) on Linux platforms. Therefore, we downloaded a public domain FFT library called FFTW (Frigo and Johnson, 1999). This library achieves good performances, and is quite flexible. It supports multi-dimensional FFTs with arbitrary lengths of the axes, and a flag (`FFTW_THREADSAFE`) can be set during initialization to make safe sharing some pre-computed data (e.g., table of twiddle factors) between threads. We run the benchmarks with both the SGI native FFTs and the FFTW library and show the results for both cases. It turns out that the FFTW library is somewhat slower than the native SGI library, but it achieves respectable performances, on the Power Challenge. On the Origin 200 it is considerable slower than the native library even on a single processor. This slow down is more pronounced for out-of-cache problems than for in-cache ones.

To average-out abnormal temporary system behavior, the tests were run on many depth steps for a total computation time of at least five minutes even for the small problem running on several CPUs.

Small problem results

First, we analyze the results for the small problem. We display the results as relative speeds normalized by the speed achieved on one CPU of the Power Challenge by the program using the public domain FFT library; that is, the slowest possible run. The relative speeds are computed from elapsed time measured on systems as empty as possible.

Figure 1 shows these relative speeds for all the computations performed to solve the small problem, excluded one-time initializations. The measured speed are plotted with solid lines, while the dashed lines correspond to the ideal parallel speed up for each program. As expected, the slowest runs were on the Power Challenge (lines labeled 1 and 2), that is also the oldest among the computers we tested.

The four-processor Xeon (line 6) is the fastest of all computers, running even faster than the O200 with native FFTs (line 4). The dual-processor Pentium III (line 5) runs slightly slower than the Xeon, likely because of the slower I/O. The parallel speed-up for all cases is reasonable, though for such small problem the serial I/Os handicap the parallel runs.

Figure 2 shows the relative speeds for the whole parallel portion of the code. A good parallel speed-up is achieved in all cases, indicating that when the problem fits in cache there is no degradation of performances caused by contentions between threads in accessing memory. The dual-processor Pentium III runs about 10% slower than the Xeon, and shows slightly worse parallel speed-up. Since the CPU run at the same speed, this difference can be attributed to the smaller and slower secondary cache.

Figure 3 shows the relative speeds for the FFTs, and has a similar interpretation as Figure 2.

Notice, that this is the case for which the Xeon out-performs the Power Challenge the most, with the four processors running about fourteen times faster than one R8000. The Origin 200 and the dual-processor Pentium III scale the worse for the FFTs, probably because of slower caches.

Large problem results

Next, we analyze the results for the large problem. Figure 4 shows the relative speeds for all the computations performed to solve the large problem, excluded one-time initializations. In relative terms, the SGIs performs better than Intel-based computers for this large problems, both as single-CPU speed and as parallel speed up. The MIPS-processors' performances are less affected by out-of-cache computations than the Pentium III's because the memory bandwidth of the SGI systems is better balanced with the CPU speed. Actually, for the large problem, the parallel speed-up improves on the SGIs because the large problem is more computationally intensive than the small one, and thus the serial I/O are less of an handicap. On the contrary, the parallel speed-up of the Pentiums is little worse than for the small problems. This is a possible indication of memory contentions between threads in accessing memory.

Figure 5 shows the relative speeds for all the computations performed in parallel. The SGIs show almost perfect parallel speed-up, indicating no contentions in accessing memory. On the contrary, the parallel speed-up on the Xeon is similar to the one shown in Figure 4, indicating that the loss in parallel efficiency is likely caused by memory contentions, and not by the serial I/Os. Somewhat surprisingly, the FFTs seem to run in parallel without loss of performances on all machines (Figure 6), with the exception of the dual-processor Pentium. It is possible that the memory access pattern of FFTs benefits from the larger and faster caches more than the rest of the computations.

KIRCHHOFF MIGRATION

Ideally, Kirchoff offset migration can be written to be almost perfectly parallel. Each node can be given a region of image space and the data within a given aperture of the imaging volume. Each node then independently read in the portion of the traveltime table it needs, and sum the corresponding input data to form the output model. No communication between nodes is needed until all threads have finished their given imaging volumes.

Such an implementation was not possible for this test. In order to implement the parallel scheme above, each processor must be able to seek and read the traveltime table while no other thread is operating on the file. OpenMP, accounts for such difficulty with the `CRITICAL` construct. Unfortunately, the `CRITICAL` construct is not handled correctly by the Portland Group's `pgf90` compiler. To overcome this limitation a section of the traveltimes were read in and then the output CMP's within this region were parallelized over.

For each of the four computers, we tested the speed both within the parallel region (Figure 7), and of the entire program (Figure 8). Within the parallel region all four machines

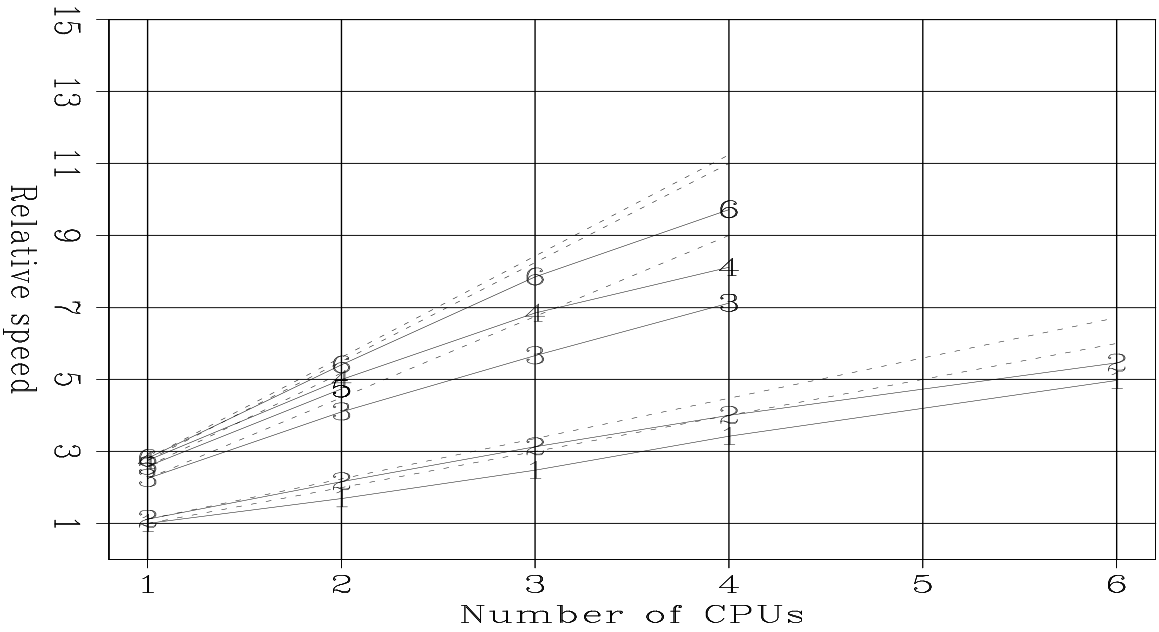

Figure 1: Relative speeds as a function of number of CPU for all the computations in the small problem: 1) Power Challenge (FFTW), 2) Power Challenge (SGILIB), 3) O200 (FFTW), 4) O200 (SGILIB), 5) Dual-processor Pentium III (FFTW), 6) Four-processor Xeon (FFTW). One processor Power Challenge has a speed of 1. The dashed lines correspond to the ideal parallel speed up. biondo2-Small-All [NR]

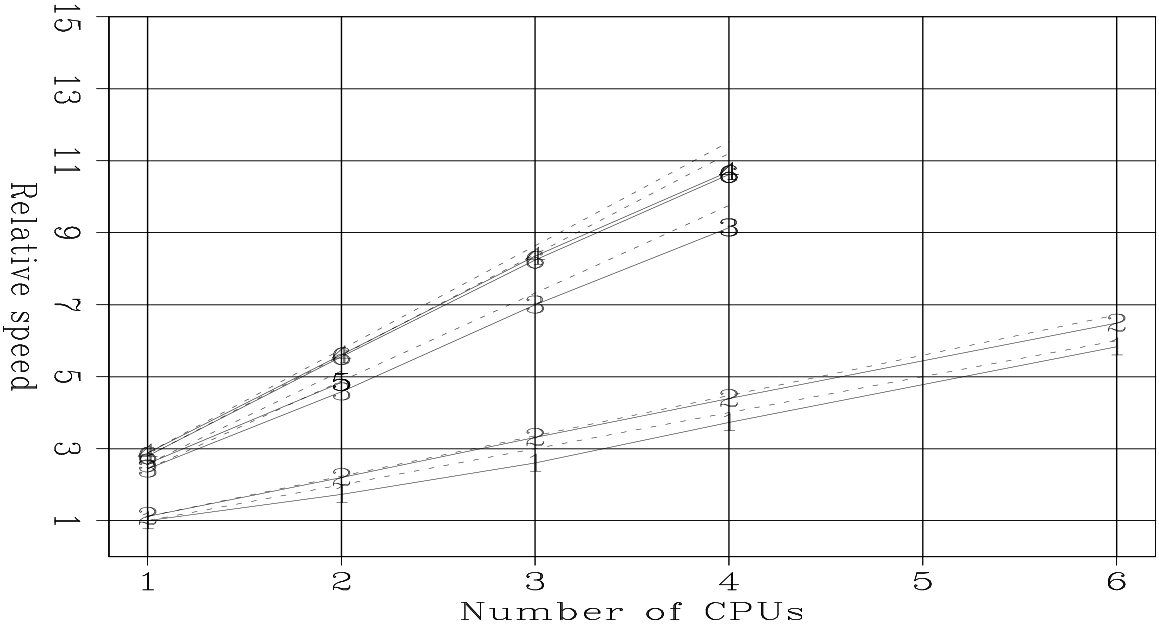

Figure 2: Relative speeds as a function of number of CPU for all the parallel computations in the small problem: 1) Power Challenge (FFTW), 2) Power Challenge (SGILIB), 3) O200 (FFTW), 4) O200 (SGILIB), 5) Dual-processor Pentium III (FFTW), 6) Four-processor Xeon (FFTW). One processor Power Challenge has a speed of 1. The dashed lines correspond to the ideal parallel speed up. biondo2-Small-par-All [NR]

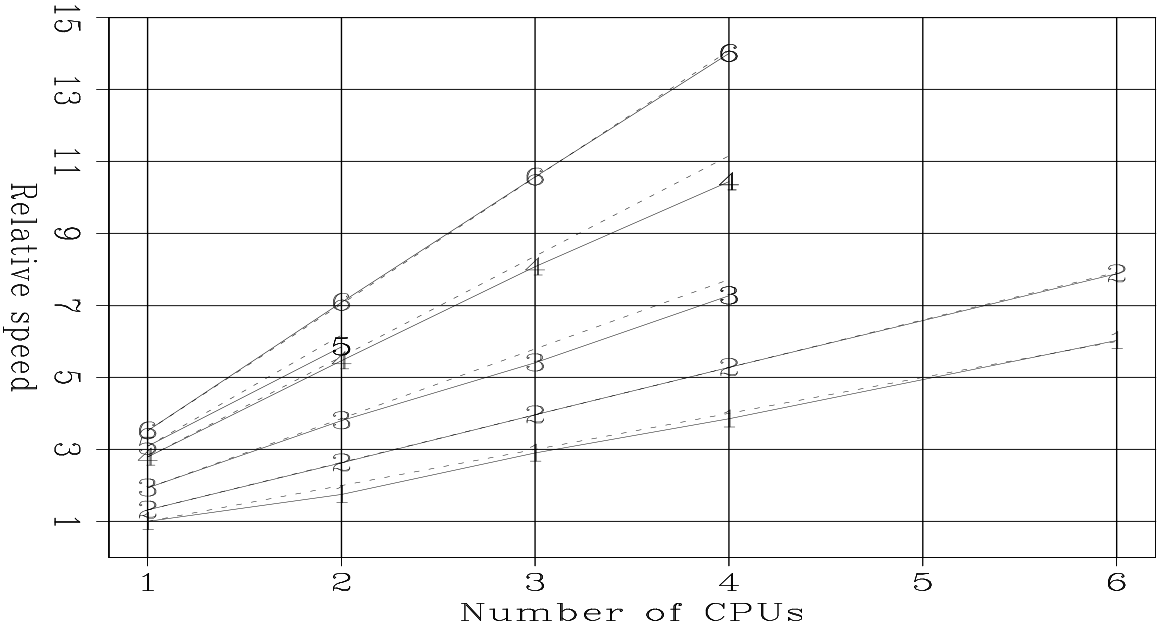

Figure 3: Relative speeds as a function of number of CPU for the computations of the FFTs in the small problem: 1) Power Challenge (FFTW), 2) Power Challenge (SGILIB), 3) O200 (FFTW), 4) O200 (SGILIB), 5) Dual-processor Pentium III (FFTW), 6) Four-processor Xeon (FFTW). One processor Power Challenge has a speed of 1. The dashed lines correspond to the ideal parallel speed up. biondo2-Small-fft-All [NR]

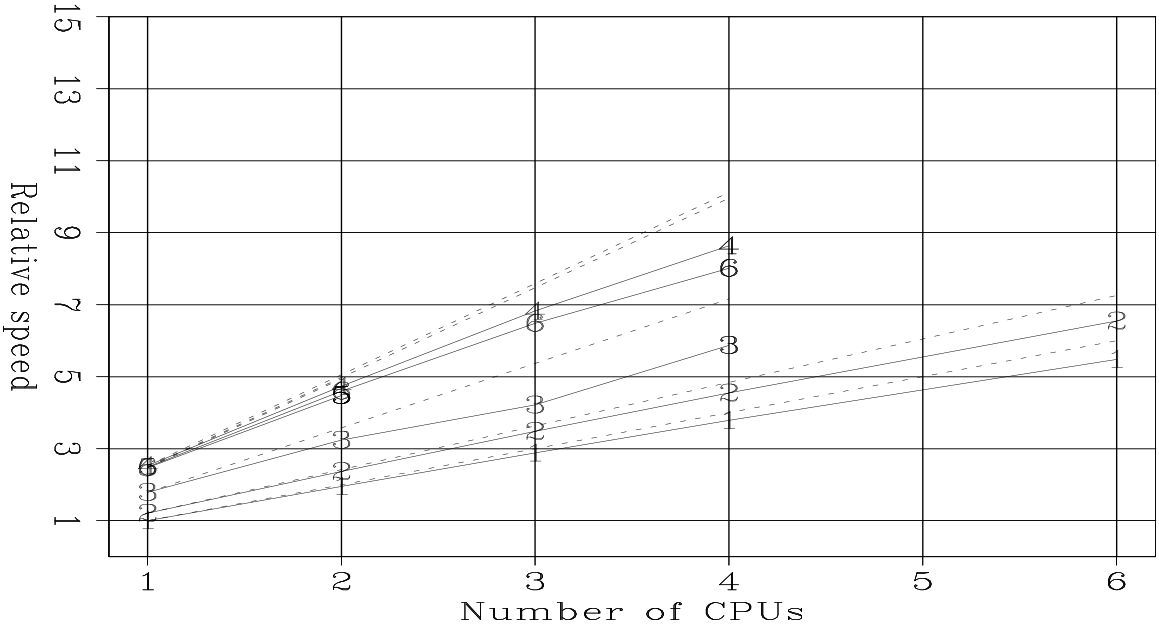

Figure 4: Relative speeds as a function of number of CPU for all the computations in the large problem: 1) Power Challenge (FFTW), 2) Power Challenge (SGILIB), 3) O200 (FFTW), 4) O200 (SGILIB), 5) Dual-processor Pentium III (FFTW), 6) Four-processor Xeon (FFTW). One processor Power Challenge has a speed of 1. The dashed lines correspond to the ideal parallel speed up. biondo2-Large-All [NR]

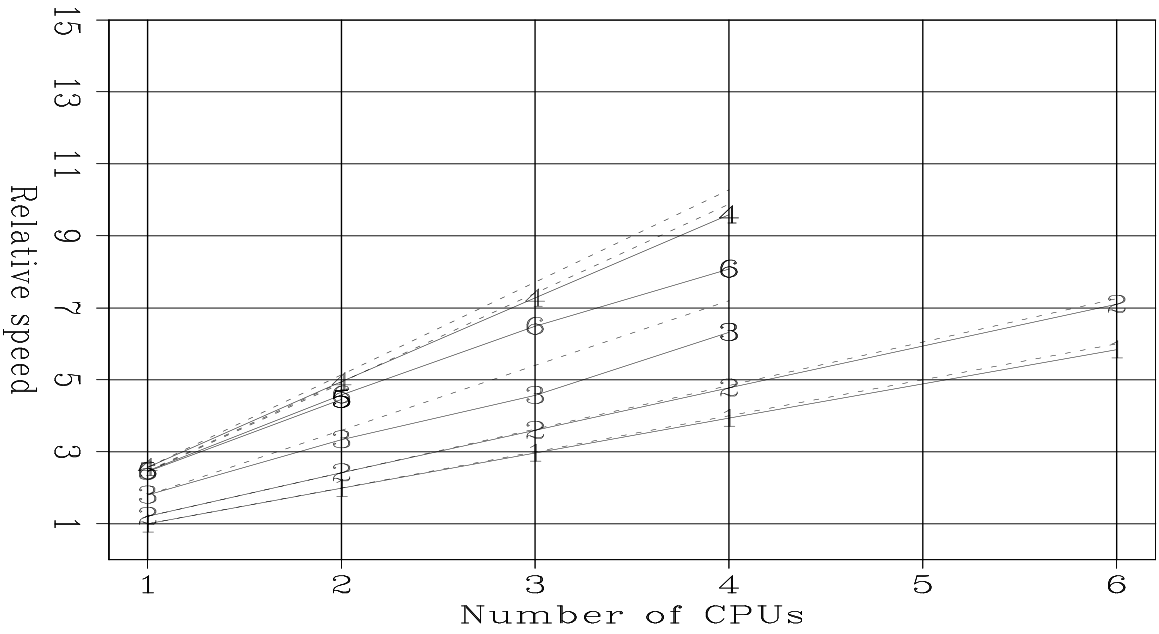

Figure 5: Relative speeds as a function of number of CPU for all the parallel computations in the large problem: 1) Power Challenge (FFTW), 2) Power Challenge (SGILIB), 3) O200 (FFTW), 4) O200 (SGILIB), 5) Dual-processor Pentium III (FFTW), 6) Four-processor Xeon (FFTW). One processor Power Challenge has a speed of 1. The dashed lines correspond to the ideal parallel speed up. biondo2-Large-par-All [NR]

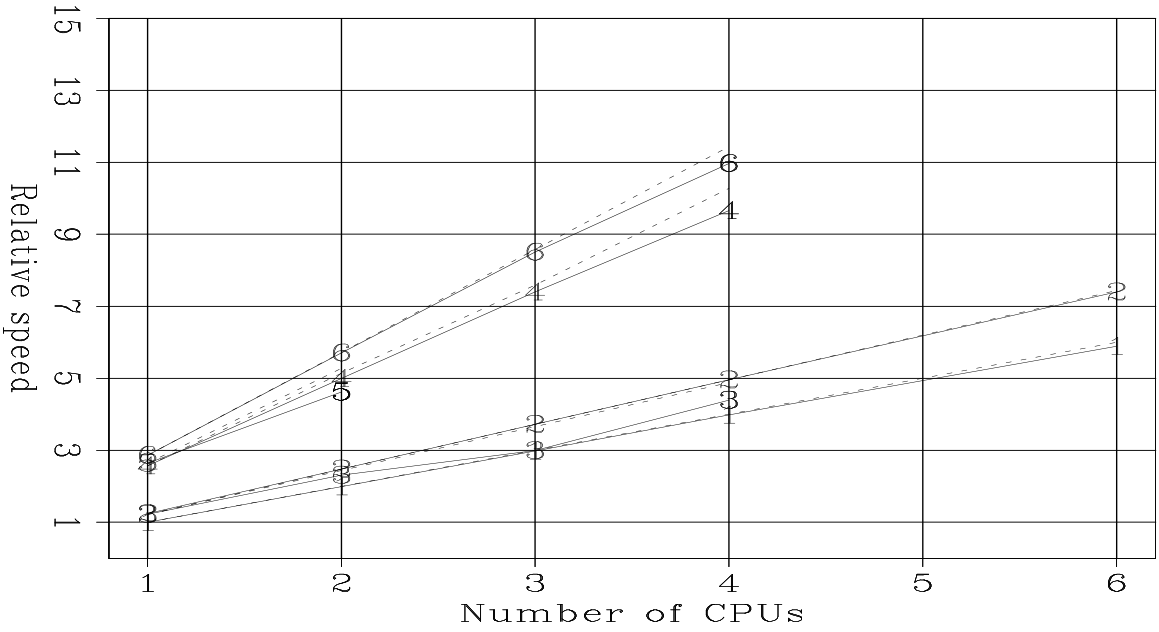

Figure 6: Relative speeds as a function of number of CPU for the computations of the FFTs in the large problem: 1) Power Challenge (FFTW), 2) Power Challenge (SGILIB), 3) O200 (FFTW), 4) O200 (SGILIB), 5) Dual-processor Pentium III (FFTW), 6) Four-processor Xeon (FFTW). One processor Power Challenge has a speed of 1. The dashed lines correspond to the ideal parallel speed up. biondo2-Large-fft-All [NR]

scaled fairly well. The notable exception being the Origin 200 when going across the Cray Link cable (from two to three processors). For overall speed within the parallel region the Xeon four-processor machine performed the best.

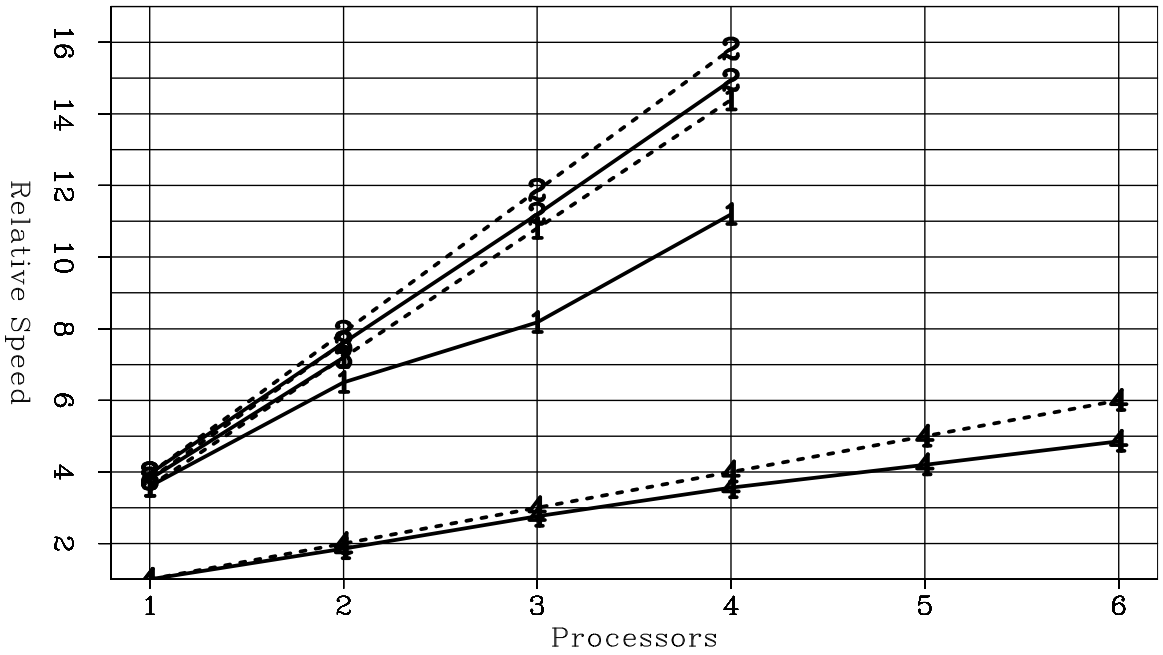

Figure 7: Relative speed of the parallel portion of the Kirchoff migration on the various testing platforms. **1** represents the SGI, Origin 200; **2** is the SGI 1400L ; **3** is the VA Start X MP; and **4** is the SGI Power Challenge. In each case the solid line represent actual performance, the dashed lines ideal performance. biondo2-kirch.par [NR]

For the entire code the results were more interesting. Because Kirchoff migration is so I/O intensive, the performances of the machine's I/O became important. Where the advantage of the Xeon versus the Power Challenge was nearly 4:1 on four processors within the parallel region it was only 2.5:1 when I/O was taken into account. The VA Start X performed particularly poorly when accounting for I/O. Its speed advantage compared to the Power Challenge dropped from 3.9 to 2.1 when accounting for I/O. The VA's I/O problem seems to be more hardware rather than OS related. The Xeon was running the same OS, and saw its speed advantage drop only from 4.1 to 2.7.

PGF90 compiler issues

To make the performance testing as equivalent as possible it was necessary to write a code that worked on the subset of OpenMP features supported by the SGI and Portland Group compilers. For these tests the limiting factor was definitely the Portland Group's F90 compiler. In the process, we encountered a number of bugs/features with the Portland Group's compiler:

`CRITICAL:` The critical region statement did not function properly.

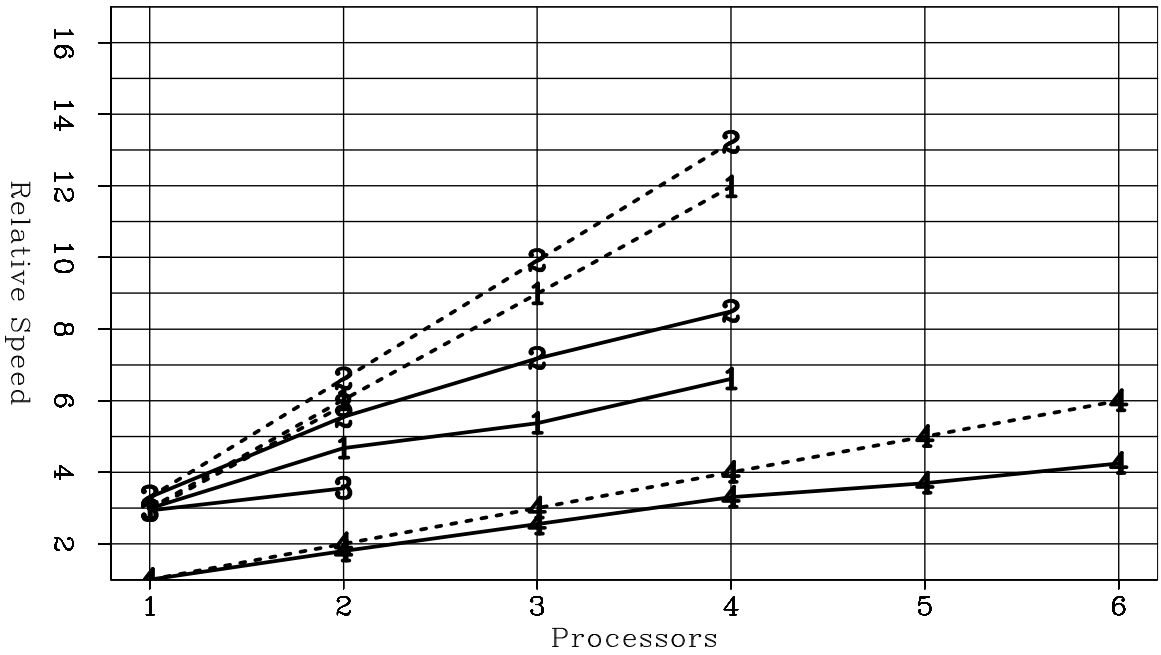

Figure 8: Relative speed of the total execution time of the Kirchoff migration on the various testing platforms. **1** represents the SGI, Origin 200; **2** is the SGI 1400L ; **3** is the VA Start X MP; and **4** is the SGI Power Challenge. In each case the solid line represent actual performance, the dashed lines ideal performance. biondo2-kirch.tot [NR]

array sections: Passing array sections through to subroutine was prone to produce incorrect results.

pointer arrays and C: The compiler sometimes had trouble communicating with gcc when passing F90 pointer arrays. This is not really a bug because according to F90 specifications the memory layout of pointer arrays is not defined. However, all other F90 compilers we used so far interface with gcc without problems.

IMPLICIT FINITE-DIFFERENCE MIGRATION

The third and final test benchmarking test was the migration of 2-D prestack and 3-D poststack datasets with implicit finite-differences. The code used for these tests was developed to implement a range of wavefield extrapolation operators, including implicit finite-difference migration in the helical coordinate system (Rickett et al., 1998). For these benchmarking test cases, we used the 65° operator described by Lee and Suh (1985), and for the 3-D test, rather than inverting the full extrapolation matrix, we made the splitting approximation. We used the LAPACK (Anderson et al., 1995) subroutine `cgtsv` to solve the resulting tridiagonal systems.

Compiler notes

The code was written in vanilla Fortran90 with OpenMP compiler directives, to ensure a level of portability. Initial development was done simultaneously on the SGI platform, and a Linux workstation with the NAGware F90 compiler, again for reasons of portability. Unfortunately, when recompiling the code on with the Portland Group F90 compiler, we ran into the compiler bugs described above that necessitated rewriting of large portions of the code.

Results

Figures 9 and 10 show the finite-difference benchmark results for the four platforms. As in the previous tests there was little separating the MIPS-based quad processor SGI with the Intel Xeon-based quad processor Linux. The results appear to be similar to the larger split-step migration benchmark.

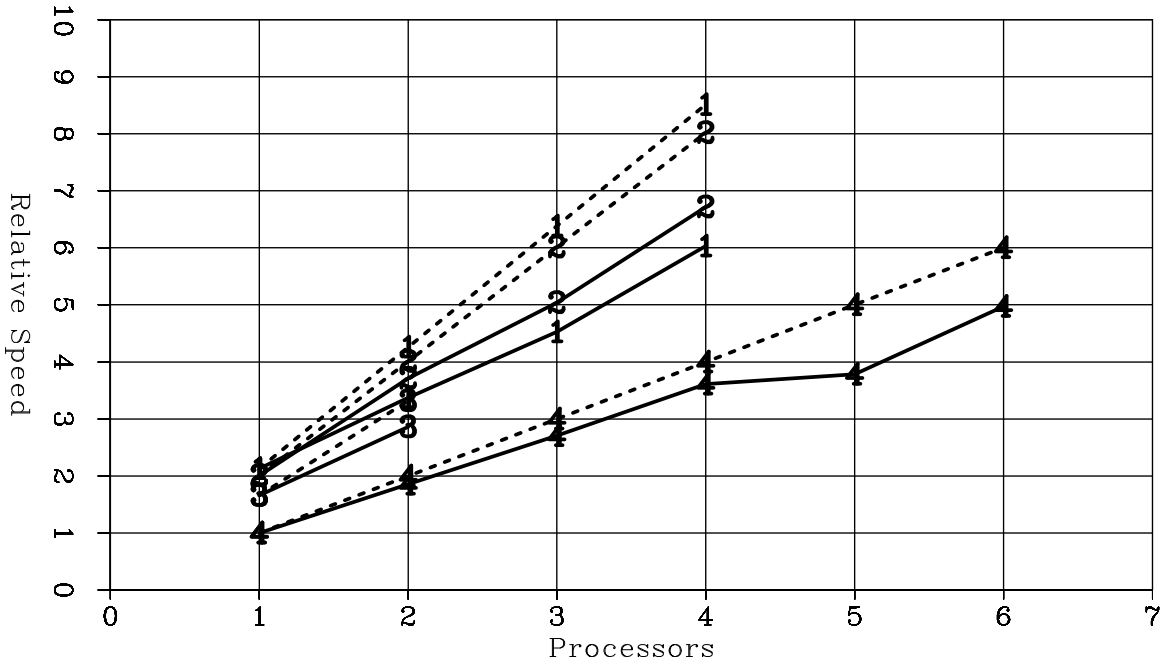

Figure 9: Relative speed of the parallel portion of the finite-difference migration on the various testing platforms. **1** represents the SGI, Origin 200; **2** is the SGI 1400L ; **3** is the VA Start X MP; and **4** is the SGI Power Challenge. In each case the solid line represent actual performance, the dashed lines ideal performance. biondo2-fdmig.par [NR]

CONCLUSIONS

The computational speed and the I/O speed achieved by the Linux server (SGI 1400L) when running our benchmarks give us confidence that we can run efficiently “production” codes

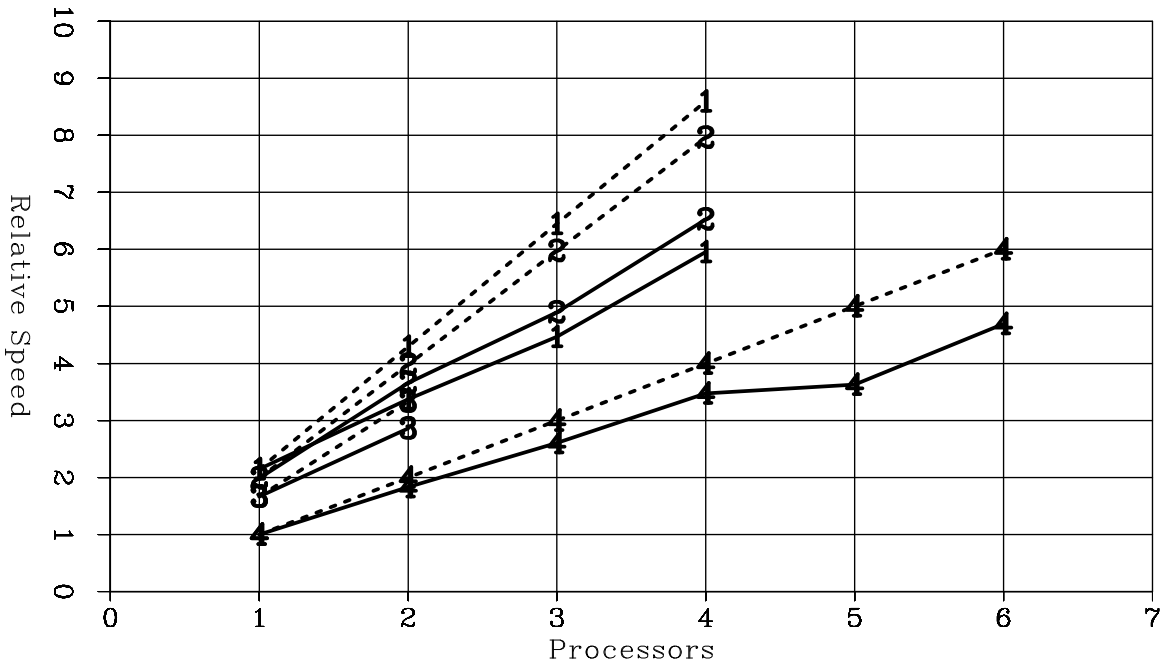

Figure 10: Relative speed of the total execution time of the finite-difference migration on the various testing platforms. **1** represents the SGI, Origin 200; **2** is the SGI 1400L ; **3** is the VA Start X MP; and **4** is the SGI Power Challenge. In each case the solid line represent actual performance, the dashed lines ideal performance. biondo2-fdmig.tot [NR]

on Intel-based multi-processors running Linux as OS. The price-performances of these systems are very attractive compared with the MIPS-based SGI systems. One drawback of the Linux-Intel solution is that only four-processor systems are available at the moment, and eight-processor systems in the near future. SGI's Origin 2000 can scale up to 128 processors. However, the price-performances of the Origin 2000 is not attractive when compared with clustering several Intel based multi-processors.

We measured a slight difference in computational performances between the dual-processor Pentium and the four-processor Xeon; but probably these differences are not enough to justify the large cost differential between Pentium III processors with small caches (512 KB) and large caches (2MB). The substantial difference in I/O performance between the two systems is safely attributable to the fact that the dual-processor system was packaged as a workstation instead of as a server.

The Portland Group's F90 compiler proved to be a workable solution to compile our F90 codes. It generates efficient code, though it does not support all F90 and OpenMP features as robustly as we wished. Being a relatively new product, we are confident that the problems we observed will rapidly disappear as new versions are released.

ACKNOWLEDGMENTS

We are grateful to SGI for loaning SEP the 1400L system that we used for our tests. We are also grateful to PGI for granting us several extensions to our evaluation licenses for the `pgf90` compiler, and for giving us access to a pre-release version of their 3.3-1 release.

REFERENCES

Anderson, E., Bai, Z., Bischof, C., Demmel, J., and Dongarra, J., 1995, Lapack users' guide: Release 2.0: Society for Industrial and Applied Mathematics, 2nd edition.

Biondi, B., and Sava, P., 1999, Wave-equation migration velocity analysis: SEP–**100**, 11–34.

Biondi, B., 1999, Subsalt imaging by common-azimuth migration: SEP–**100**, 113–124.

Frigo, M., and Johnson, S. G., 1999, FFTW: `http://www.fftw.org/`.

Lee, M. W., and Suh, S. Y., 1985, Optimization of one-way wave-equations (short note): Geophysics, **50**, no. 10, 1634–1637.

Rickett, J., Claerbout, J., and Fomel, S., 1998, Implicit 3-D depth migration by wavefield extrapolation with helical boundary conditions: SEP–**97**, 1–12.