HPF to OpenMP on the Origin2000: A Case Study

Leesa Brieger

Geophysics, CRS4

C.P. 94, I - 09010 Uta, Italy

(*leesa@crs4.it*)

This is a preprint of an article accepted for publication in

Concurrency: Practice and Experience,

Abstract

The geophysics group at CRS4 has long developed echo reconstruction codes in HPF on distributed-memory machines. Now, however, with the arrival of shared-memory machines and their native OpenMP compilers, the transfer to OpenMP would seem to present the logical next step in our code development strategy. Recent experience with porting one of our important HPF codes to OpenMP does not bear this out - at least not on the Origin2000. The OpenMP code suffers from the immaturity of the standard, and the operating system's handling of UNIX threads seems to severely penalize OpenMP performance. On the other hand, the HPF code on the Origin2000 is fast, scalable and not disproportionately sensitive to load on the machine.

Keywords: shared-memory programming, OpenMP, HPF, Origin2000

1 Introduction

The geophysics group at CRS4 has traditionally developed its seismic codes in High Performance Fortran (HPF) for distributed-memory machines, using the PGHPF compiler from The Portland Group [1]. HPF is made for distributed data computations [2] and is a natural choice for applications such as the current one, with an intrinsic data parallelism; in fact HPF offered us very good scalability for our application – until the HPF code was moved from the SGI Power Challenge to the Origin2000 and experienced a significant loss of performance. This and the fact that our codes are run increasingly on SGI shared-memory machines (specifically, on the Origin2000) convinced us to investigate OpenMP as part of our future code development strategy. OpenMP, a library of compiler directives used with Fortran or C for thread-level parallelism on shared-memory machines [3], is native on the Origin, leading one to imagine that it might be more efficient than Portland Group's HPF could be on that machine.

While HPF is centered around data distribution, the philosophy behind OpenMP is work distribution. For the seismic application in question, however, work distribution and data distribution are practically synonymous, since the concurrent operations which make up the algorithm correspond to different data (frequencies). It is as natural and should be as efficient to distribute data-related threads among the processors of a shared-memory machine as it is to distribute the data among the processors of a distributed-memory machine.

In the following sections are described the relevant characteristics of the parallel application under study and of the parallel environment in which it has evolved. This includes some description of problems encountered with OpenMP in general as well as on the Origin2000 in particular [4]; these problems cause OpenMP to be much less efficient than HPF for the current application on that machine. The last sections include performance comparisons between HPF and OpenMP for this application and our resulting conclusions. The HPF code had initially appeared to be drastically penalized by the Origin's architecture since it ran much faster on the Power Challenge than on the Origin. Now, compilers and operating system upgrades have brought impressive performance gains on the Origin2000, and we are back to a fast, scalable HPF code.

2 The Parallel Application

By describing the background of our application, we hope to give the reader a grasp of the problem, which requires high-performance parallelism in order to be competitive. This should also serve to expose the intrinsically data parallel nature of the problem; only a very small fraction of the computations require any "parallel programming" because most are carried out concurrently. As a consequence, this is an application which should not pose difficult problems of parallelism and should result in efficient, scalable code – in *any* parallel language.

Echo-reconstruction techniques for non-intrusive imaging have wide application, from subsurface and underwater imaging to medical and industrial diagnostics. The techniques are based on experiments in which a collection of short acoustic or electromagnetic impulses, emitted by a source at the surface, illuminate a certain volume and are backscattered by inhomogeneities of the medium. The inhomogeneities act as reflecting surfaces which cause signal echoing; the echoes are recorded by receivers at the surface and processed using a propagation model to yield a 3D image of the hidden geometry. The most demanding processing techniques for non-invasive imaging involve simple acoustic signals for seismic exploration, for which the huge data sets and stringent performance requirements make high performance computing essential. The application under study in this report, depth migration, is based on the scalar wave equation and represents a standard approach for seismic imaging.

In the migration process, the pressure waves (seismic traces) recorded at the surface are used as initial conditions for a wave field governed by the scalar wave equation in an inhomogeneous medium:

$$\frac{1}{v^2(x,y,z)}\frac{\partial^2 P}{\partial t^2} = \frac{\partial^2 P}{\partial x^2} + \frac{\partial^2 P}{\partial y^2} + \frac{\partial^2 P}{\partial z^2}\ . \tag{1}$$

In the exploding reflector model [5], signals are compressed into *zero-offset* traces simulating coincident source-receiver experiments. In such a model, the downward raypath and traveltime from the source to a point of reflection (reflector) is identical to the upward raypath and traveltime from the reflector to the receiver. By propagating the zero-offset seismic traces $P(x, y, 0, t)$ down and in reverse time according to Eq.(1) with halved velocity, it is possible to calculate the field of signal intensity $R(x, y, z)$ necessary to reproduce the ensemble of seismic traces, but now imagining the reflectors as the sources of the acoustic signal. In fact, R is given by the imaging condition $R(x, y, z) = P(x, y, z, t = 0)$.

This signal intensity can be interpreted as a map of the local reflectivity because large values of the field R correspond to sharp contrasts in the velocity field. Thus the exploding reflector model allows the interpretation of the migrated section $P(x, y, z, t = 0)$ as an acoustic image of the volume.

As long as velocity v is constant, the wave equation can be solved, and thus the acoustic image produced, using depth z as the advancing variable for the propagation of the seismic section $P(x, y, 0, t)$. This is known as depth extrapolation and is possible because the wave equation rewritten as

$$\frac{d^2 \hat{P}(k_x, k_y, z, \omega)}{dz^2} = -k_z^2 \hat{P}(k_x, k_y, z, \omega) \tag{2}$$

in the wavenumber-frequency domain (k_x, k_y, ω), has a characteristic solution of the form:

$$\hat{P}(k_x, k_y, z + \Delta z, \omega) = \hat{P}(k_x, k_y, z, \omega) e^{ik_z \Delta z} \tag{3}$$

where $k_z = \frac{\omega}{v}\sqrt{1 - \left(\frac{v}{\omega}\right)^2 \left(k_x^2 + k_y^2\right)}$.

This convenient form is used as the basis for depth migration, including the construction of approximate methods for non-constant velocity fields [6, 7, 8]. Since the signals which are transformed and then depth-propagated are real, the solution in the space-frequency domain can be used to calculate the imaging condition as follows: $R(x, y, z) = 2\sum_{\omega>0} Re[\hat{P}(x, y, z, \omega)]$.

Using this formulation, the 3D imaging technique used for the case study of this presentation is intrinsically data parallel. At each depth z, the processors divide among themselves the concurrent calculations (independent for each ω) for evaluating the solution $\hat{P}(x, y, z, \omega)$. This requires numerous FFT calculations and determines the contribution $F = Re[\hat{P}]$ of each frequency ω to the image plane at that depth. Then the sum of these contributions – the only operation of the algorithm which is not entirely concurrent – yields the 2D image for that depth. This is an array reduction step of the form $Image(x, y) = \sum_{\omega>0} F(x, y, \omega)$, which must be carried out once at each depth z. Since this is the only step of the algorithm requiring communication or remote memory access, the parallelism should be very efficient and scalability very good for this application.

3 Problems with OpenMP

As mentioned above, HPF was the language of reference for this study because it is so well-suited to the application on distributed-memory machines. To achieve the necessary distribution over frequency of the solution array $\hat{P}(x, y, \omega)$ at a given depth, there are simple directives of the form `!hpf$ distribute P(*,*,block)`. Interprocessor communications are either transparent or via intrinsic procedures, and Fortran 90 array syntax is parallelized.

Unfortunately, OpenMP is still under construction and problems of language immaturity are easy to find. The language proposes – for scalars only – a reduction operator which carries out an efficient evaluation of the sort of reduction sum required by our application. Unfortunately, our application requires reduction on three-dimensional arrays, for which OpenMP's reduction is not yet ready; the actual operation must be programmed using barriers or critical regions to avoid conflicts between processors. The extension of the reduction operation to arrays will apparently be incorporated into the next version (2.0) of OpenMP, but is as yet not present. (On the other hand, HPF requires only the Fortran 90 intrinsic "sum" to define the reduction sum across the processors.)

Dramatically lacking in the current OpenMP standard is support for Fortran 90. OpenMP directives can be used with Fortran 77 or C, but not, in theory, with Fortran 90. Most computer makers have implemented support for some Fortran 90 characteristics on their machines, but this remains inconsistent. Some of the best features of Fortran 90 cannot be counted on with OpenMP; dynamic allocation with OpenMP on the Origin was not operational before the latest compiler upgrades, and support for Fortran 90 modules is vendor-dependent. This state of things will apparently also be corrected with version 2.0 of the OpenMP Fortran specification, which should include support for Fortran 90, but for now this harsh restriction on the use of OpenMP remains. Needless to say, this means that Fortran 90 array syntax is not currently parallelized by OpenMP.

OpenMP, intended as it is for the shared-memory environment, was not originally meant to handle data placement and contains no data distribution directives. While a shared-memory machine guarantees that all data is logically equally accessible to all processors, the cost of that access is not guaranteed to be uniform. Requirements of scalability are pushing most vendors away from symmetric multi-processor (SMP) structure and toward some form of non-uniform memory access (NUMA) or even outright clustering of SMP nodes. One consequence of this is that for code optimization, data position and "communication" (remote memory access) costs cannot in general be ignored, even on shared-memory platforms. In particular, the Origin2000 is a highly scalable distributed shared-memory (DSM) machine. Its ccNUMA nature gives the cache coherency which assures the shared-memory model, and yet the scalable architecture imposes variable memory access costs: less expensive for local access, more expensive for remote access. While OpenMP has no data placement philosophy nor instructions built into it, such directives are nonetheless furnished by some vendors for their shared-memory machines; for example, on the Origin2000, the SGI directives form an extension of the OpenMP library [9]. While such machine-specific extensions are necessary for optimizing code, their use inhibits the portability that the OpenMP standard was meant to provide. OpenMP must in the future be equipped to manage data placement or otherwise to make thread-data associations in an implementation-independent way if it is to take its place as a standard in parallel programming. This is especially true if the current trend toward the clustering of SMP nodes continues.

3.1 Problems with OpenMP on the Origin2000

Because the original HPF code was already structured for efficiency on distributed-memory machines and because the Origin's is a ccNUMA architecture, it seemed advisable to preserve the same data structure, to the extent possible, even for the OpenMP version of the code. This structure on the DSM architecture of the Origin2000 should guarantee that local memory accesses are favored over the costlier remote accesses and that scalability is thus enhanced. In the absence of data placement directives in OpenMP, the SGI directives should have provided the means for controlling data placement and imposing the desired structure on the data.

A "page" (typically 16K bytes) is the minimal granularity of memory space on the Origin, and `!$page_place`, which is supposed to give the user control over placement of pages in memory, was the SGI directive of choice. The page granularity does not allow the fine control over data placement which one generally expects; for example, an array distributed among several processors is distributed by page, not by element. One can thus be surprised by finding some elements (those which happen to overlap on the page allotted to another processor) residing where the user did not intend them to be. This can be avoided by padding the array so as to separate data blocks by at least a page of memory space, thus avoiding the page overlapping between them. This level of hand tuning to separate subarray blocks is unnecessary in HPF even on shared-memory machines. However, even with this hand tuning, `page_place` was not operative on the Origin at the time of the conversion of our code from HPF.

Other SGI data distribution directives which should circumvent the page granularity restriction on data distribution, such as `!$distribute_reshape`, are incompatible with dynamic allocation of the distributed arrays. Such directives were not considered an option for use with our application, in which dynamic array allocation was judged of priority importance.

So first-touch default placement was utilized. This is the SGI default page-allocation policy which means that "each page is allocated from the local memory of the processor that incurs a page-fault on that page" [9]. Simply initializing the array via a parallel do-loop whose iterations are distributed conveniently among the processors will achieve a desired distribution – to within the usual constraints imposed by the page granularity. (Again, page overlapping can be avoided between subarray blocks by accordingly padding the array.) Unfortunately, even the first-touch convention is not guaranteed to give the desired data placement if the machine is under heavy use. If the operating system (OS) swaps threads among processors while the first-touch initialization is taking place, a subarray block can wind up scattered, page by page, among several memories. On the SGI Origin2000, this can happen even when threads are "locked" onto their processors, because the OS can and does, depending on machine use, override the user-specified directives and environment variables. Thus any control the user thinks she has over data distribution in the machine may well be illusory.

A further problem with OpenMP on the Origin2000 comes from the thread swapping that the OS carries out. Even with data laid out exactly according to a user's plan, if threads are continually hopping around among processors, data locality may never be respected – because the thread jumps away from the data that was supposed to be associated with it. The end result is to enforce that the memory accesses can essentially *never* be local and cheaper. This is a major problem with the implementation of OpenMP on the Origin and points to a major hole in the OpenMP specification; the standard itself provides no specification for associating particular threads with particular data, even though data parallelism and thread parallelism for many applications may be virtually synonymous.

4 Performance

Our migration codes were tested on a 16-CPU Origin2000; our jobs shared the machine with other memory-intensive applications in a production environment. Share II was used to manage resource sharing among applications, with priority use of 4 CPUs given to our programs.

OpenMP performance for the application in this environment is significantly inferior to HPF performance. Because of thread swapping during the first-touch placement phase, the OpenMP version is only rarely able to achieve data placement according to the desired stucture. HPF on the shared-memory machine does not suffer from similar problems of data distribution; it utilizes local/private variables for placing sub-blocks of distributed arrays and so is not affected by memory granularity nor OS interference for data placement. (HPF data distribution is also not adversely affected by dynamically allocated arrays.)

Further, performance of the OpenMP program is disproportionately penalized by seemingly uncontrolled OS swapping of threads among processors. This is the case even when sufficient availability of CPUs should make thread swapping unnecessary, but it becomes much more pronounced when the number of free CPUs is inferior to the number requested by our job. Then, OS thread management causes elapsed time for the OpenMP application to increase by unpredictable (and sometimes very large) amounts.

The fact that these same situations do not have a disproportionate effect on HPF elapsed times seems to imply that the HPF code does not suffer from OS interference the way the OpenMP code does. Apparently the Origin's operating system treats HPF processes differently from OpenMP threads – much to the disadvantage of the OpenMP threads!

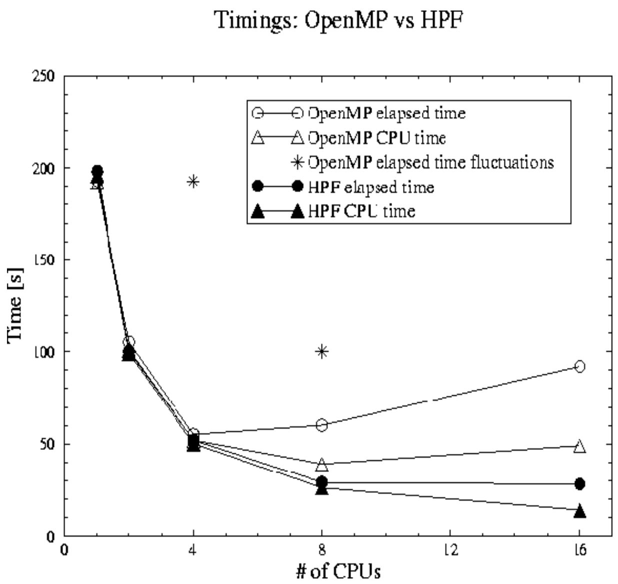

Figure 1: Timings of the OpenMP and HPF codes. Fluctuations in elapsed time for OpenMP show effect of OS interference when resources are shared. No such fluctuations are observed for HPF running in the same environment.

Figure 1 shows elapsed and CPU times for the HPF and OpenMP codes, running on 1 to 16 processors; both codes ran in exactly the same production environment. The extreme effect of OS interference in the OpenMP program is visible in the fluctuations in elapsed time for some of the OpenMP runs. The worst examples of the OS effect were not included in this figure. For example, some 16-CPU runs took 900 seconds to complete and others had accomplished only a very small fraction of the calculations after 840 seconds! Under the same conditions, the HPF code never experienced comparable fluctuations; HPF performance remained essentially predictable as a function of CPU availability.

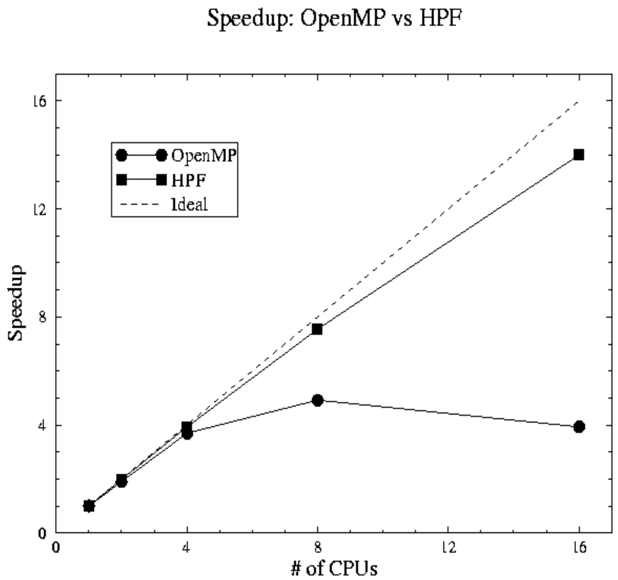

Figure 2: Speedup of the OpenMP and HPF codeson the Origin2000. The HPF code scales well; the OpenMP code does not.

Figure 2 reports speedup measured on the Origin2000 for the runs of Figure 1 without taking into account OpenMP fluctuations in performance, using the most favorable runs only. Scalability of the HPF code is far superior to that of the OpenMP code, presumably because data remain irrevocably local for the duration of an HPF run.

5 Conclusion

Results with OpenMP have been disappointing. With an application that is so naturally parallel and already coded for efficiency in HPF, we did not expect the problems with OpenMP that we did encounter on the Origin2000 and that are outlined in this article.

Some of the problems we have seen are due to the immaturity of OpenMP as a standard of parallel programming and are scheduled to be corrected with version 2.0 of OpenMP: reduction operations are supposed to be extended to arrays, and support for Fortran 90 should become part of the standard. However, the biggest problems, i.e. those having to do with data placement and data locality, do not seem to have any prospect for early resolution. OpenMP itself does not propose a method for controlling placement of data, nor does it propose to define thread-data associations to favor data locality. In the absence of such possibilities in the standard, each vendor's version of an OpenMP compiler will handle these issues in an implementation-dependent manner.

Other problems encountered in porting to OpenMP on the SGI Origin2000 are more associated with SGI's implementation of OpenMP on that machine: threads which continually jump away from the processors on which they have been running cause unnecessary interruptions in the calculations and also result in data non-locality. This is a significant problem which worsens when the machine is not in single-user mode, when OS interference can heavily influence run times, increasing them exaggeratedly.

As long as these problems remain, we conclude that OpenMP on the Origin2000 is not ready for production-level use in an environment which demands high-performance code. On the other hand, HPF has come of age by now and on the shared-memory Origin2000 yields the scalibility and performance demanded of our codes. For these reasons, OpenMP does not yet represent a viable alternative for our code development strategy.

References

[1] High Performance Fortran Compilers Survey home page. See http://www.ac.upc.es/HPFSurvey/Welcome.html.

[2] HPF 2.0 Language Definition, Jan. 1997. See The High Performance Fortran Home Page, http://www.crpc.rice.edu/HPFF/home.html.

[3] OpenMP Fortran API specification, November 1999. See http://www.openmp.org.

[4] E. Bonomi, L. Brieger, E. Pieroni, M.T. Arienti, L. Cazzola, and P. Marchetti, *PSPI: Streamlining 3D Echo-Reconstructive Imaging*, in the Proceedings of the Fifth European SGI/Cray MPP Workshop, Bologna, Italy, September 1999.

[5] R. H. Stolt, "Migration by Fourier Transform", *Geophysics* **43**, 23–48 (1978).

[6] E. Bonomi, L. Brieger, C. Nardone, E. Pieroni, "Phase Shift Plus Interpolation: A Scheme for High Performance Echo-Reconstructive Imaging", *Computers in Physics*, **12**, 126–132 (1998).

[7] C. Bagaini, E. Bonomi, E. Pieroni, *Split Convolutional Approach to 3D Depth Extrapolation*, 65th Ann.Internat. Mtg., Soc. Expl. Geophys, Expanded Abstracts, Houston 1995.

[8] F. Collino and P. Joly, “Splitting of Operators, Alternate Directions, and Paraxial Approximations for the Three-dimensional Wave Equation”, SIAM J. Sci. Comput., **16**, 1019 (1995).

[9] MIPSpro 7 Fortran 90 Commands and Directives Reference Manual, ”Parallel Processing on Origin Series Systems”, SGI Technical Publications Library. See http://techpubs.sgi.com.