

## HPF to OpenMP on the Origin2000: A Case Study

*Leesa Brieger, Geophysics Group, CRS4, C.P. 94, I - 09010 Uta, Italy (leesa@crs4.it).*

### The Parallel Application

The geophysics group at CRS4 has long developed echo reconstruction codes in PGHPF on distributed memory machines. The problems are typically intrinsically data parallel; for example, the 3D imaging technique under consideration in this presentation decouples in the frequency domain. At each depth, concurrent calculations determine the contribution of each frequency to the image plane at that depth. The sum (over frequency) of these contributions yields the final 2D image at that depth. Thus the only inter-processor communications required by the algorithm come from a global sum, elementwise for a 2D array, once at every depth.

The performance of the HPF implementation of this imaging code suffered greatly from its porting from the SGI Power Challenge to the Origin2000. Because of this and apparent waning vendor support for HPF, converting from HPF to native OpenMP on the shared memory Origin2000 seemed to present the logical next step in our code development strategy. While HPF is centered around data distribution, the philosophy behind OpenMP is work distribution. For the application in question, however, task distribution and data distribution are practically synonymous, since the concurrent tasks correspond to different data (frequencies). It is natural and should be efficient to distribute frequency information among the processors on a distributed memory machine or to distribute frequency-related tasks among the processors on a shared memory machine.

### Data Locality on Shared Memory Machines

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 toward some form of non-uniform memory access. One consequence of this is that for code optimization, data position and "communication" costs cannot in general be ignored, even on shared memory platforms.

In particular, the Origin2000, the machine used for running the code reported on here, is a highly scalable distributed shared memory (DSM) machine. Its ccNUMA nature gives the cache coherency which assures the single-memory model, and yet the scalable architecture imposes varying memory access costs: less expensive for local access, more expensive for remote access (considered here as communication).

OpenMP, used to distribute work on shared memory machines, was not originally intended to handle data placement, and any data placement directive is cur-

rently outside the OpenMP standard. Such directives are nonetheless furnished by vendors for their specific machines; for example, on the Origin2000, the SGI directives form an extension of OpenMP. 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, in an implementation-independent way, if it is to take its place as a standard in parallel programming.

### HPF vs OpenMP on the Origin2000

Because of the data-parallel nature of our problem (it decouples into concurrent operations on different data), it is particularly well-adapted to distributed memory architectures. Because the HPF code was already structured for efficiency on distributed-memory machines, conversion to OpenMP was undertaken so as to preserve the original distributed data structure of the HPF code. This structure on the DSM architecture of the Origin2000 should serve to 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" 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. A user can thus be surprised by finding some elements (those which happen to overlap on the page allotted to another processor) residing on processors other than where they were thought to have been placed. This can be avoided by padding the array so as to separate data blocks by at least a page of memory space and thus avoid the page overlapping between them. This also requires a level of hand tuning of the array that was left behind long ago on distributed 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.

So first-touch default placement was utilized. This is a convention which dictates that the processor which first "touches" (carries out some operation on) a datum will take the page containing that datum into its associated local memory. Simply initializing the array via a parallel do-loop whose iterates 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 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" to specific processors, because the OS can override, depending on machine use, the user-specified directives and environment variables.

HPF on the shared memory machine does not seem to suffer from the same problems of data distribution. Since HPF utilizes local/private variables to define even the subarray blocks of distributed arrays, there seems to be no influence of page granularity nor of OS interference on data placement.

Having accepted that control over data placement will be limited for OpenMP, the next problem posed by the OpenMP conversion was how to carry out a reduction sum on a 2D array. Efficient OpenMP reduction is currently only defined for scalars, and this is another point on which OpenMP is not yet "up to speed". The reduction sum for an array in OpenMP must be carried out "by hand", using either critical regions or barriers to avoid conflicts between processors. On the other hand, HPF requires only the F90 intrinsic "sum" to define the reduction sum.

## Performance

The HPF code had initially appeared to be drastically penalized by the Origin's architecture since it ran much faster on the SGI Power Challenge than on the Origin. Now new compilers and operating system have brought impressive performance gains on the Origin2000, and we are back to a fast, scalable HPF code.

OpenMP performance for this application on the Origin2000 is significantly inferior to HPF performance. The OpenMP version seems to be penalized by data placement as well as by OS interference when the machine is in heavy use. The fluctuations in elapsed time, visible for some of the OpenMP runs in figure 1, correspond to moments of heavy machine use; one run with 16 CPUs which took 15 minutes to finish, is not even included in the figure. The HPF code never manifested such fluctuations. Figure 2 reports speedup measured using the favorable runs only. The HPF code scales well; the OpenMP code does not.

Our results with OpenMP have so far been disappointing. Obstacles to programming in OpenMP are induced by the immaturity of the standard. Performance on the Origin seems penalized by the immaturity of the implementation on the SGI machine – and SGI seems to be ahead of its competitors in implementing OpenMP! This, along with the fact that the HPF code is back to performing well on the Origin2000 have convinced us that we shall continue our code development in HPF.

## Timings: OpenMP vs HPF

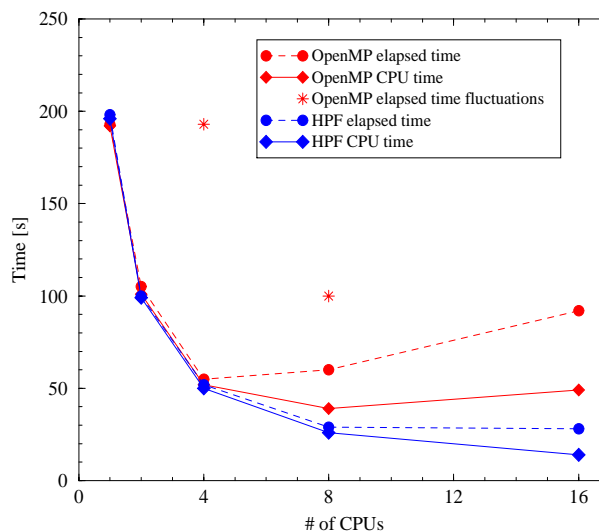


Fig. 1: Timings: OpenMP vs HPF. Fluctuations in elapsed time for OpenMP runs correspond to heavy machine use. No such fluctuations are observed for HPF.

## Speedup: OpenMP vs HPF

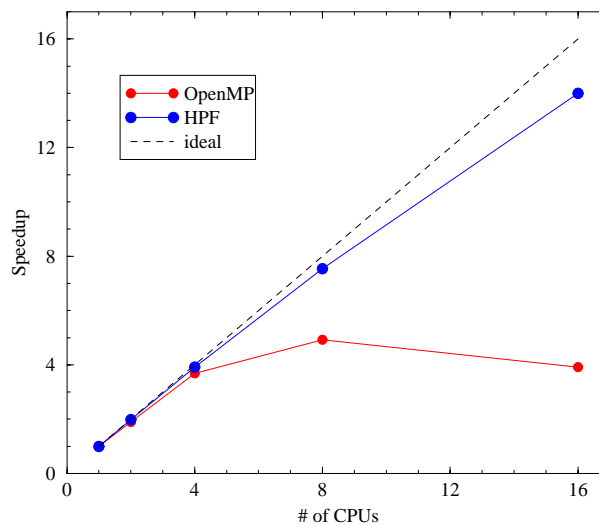


Fig. 2: Speedup: OpenMP vs HPF. The HPF code scales well; the OpenMP code does not.