

CHAPTER 7

SOFTWARE FOR HIGH-END COMPUTING

Katherine Yelick

BACKGROUND

The Earth Simulator (ES) is generally known for its physical characteristics and hardware performance, but the announcement of some of the earliest results on the system came with an equally surprising software story: some of the large, multi-teraflop/s codes were written in high-performance Fortran (HPF), a language that had been almost entirely abandoned within the United States. Just as the success of the hardware system is attributable to a focused and sustained design and development effort, so too is the success of HPF. The original visionary of the ES, Dr. Hajime Miyoshi, envisioned a machine that was not only a major leap in performance relative to previous machines, but was also easier to program. He therefore made HPF a required part of the initial requirements for the machine.

The use of HPF was the major innovation discussed in software for supercomputers, and in the case of the Earth Simulator, it was entirely an industrial effort by NEC. A consortium of industrial and academic members also had a role in the Japanese HPF effort to design their own variation in the language, which is described below. It was striking that there were few, if any, academic efforts on high-performance numerical libraries, problem-solving environments, or application frameworks for the ES or other vector supercomputers. Instead, all of these research problems were represented in the grid computing projects, often for cluster computers that formed computational and storage resources on the grid. Grid computing research is described in another chapter.

Application performance results on the ES demonstrate that the ES has a scalable operating system, collective communication, and a high-performance Fast Fourier Transform. While each of these might be part of a research effort in a grid environment, for the ES they were considered a standard part of the system that was the vendor's responsibility to provide. Thus, the cost of the ES includes a much higher level of software support than one would expect when purchasing a cluster. There was surprisingly little in the form of open source software running on the machine.

The remainder of this chapter will therefore focus on the HPF effort in Japan, beginning with a brief summary of high-performance programming languages to set the stage for further discussion. It then describes the Japanese extensions to HPF, followed by application performance results on the ES. Finally, it summarizes the impact of this effort and looks ahead towards the future of HPF in Japan, which is less promising.

HIGH-PERFORMANCE PROGRAMMING OVERVIEW

High-performance programming models include both integrated languages, designed specifically for a high-end computing environment, and libraries that are combined with standard languages such as Fortran, C or C++, where the parallelism constructs are provided in the library. HPF is an example of an integrated language, which requires its own specialized compiler, whereas the Message Passing Interface (MPI) is the most common library

extension used in high-end computing today. OpenMP is somewhere between a fully integrated language and a library, involving compiler directives and runtime library support. A fourth model of parallel machine use is to hide the parallelism from the application programmer entirely by having a sophisticated compiler automatically convert sequential loops into parallel code. While this latter model is the most attractive for programmers, it has proven impractical for machines of this scale and for applications that are as large and complex as those used on high-end machines today.

HPF falls into the general category of data parallel language. It is an extension of the Fortran 90 language in which programmers express fine-grained parallelism over arrays. For example, a simple array statement such as $A = B + C$ may express a parallel addition operation followed by parallel assignment, all for arrays that are spread across processors. HPF allows the programmer explicit control over the layout of arrays across processors, which the compiler then uses to partition the work across processors. The HPF language design effort began in 1991, and a *de facto* standard (HPF v1.0) was distributed in 1993. In the United States there was a concerted effort to implement and optimize HPF in the mid-90s, both in industry and in academia, but nearly all of that activity has been abandoned, in part because the initial performance results were disappointing.

MPI is a library of communication primitives based primarily on two-sided (send/receive) communication that is relatively easy to implement, since it does not require its own compiler support. It therefore runs on parallel machines across a spectrum of cost, size, and speed, giving the programmer precise control over both data layout and computation distribution, which are important for performance. The disadvantage is that the two-sided communication model can be cumbersome for some programs and that there is significant up-front investment in parallelism, hurdles some users never overcome.

Many of the high-end machines in use today have multiple levels of parallelism, with a set of nodes connected by a network at the top level, shared memory parallelism within the nodes, and either superscalar or vector parallelism within each processor in the node. This hierarchy is reflected in the programming models, with MPI used between nodes, and either MPI (implemented on shared memory), OpenMP, or threads within shared memory. At the lowest level, an automatic vectorizing compiler is often used on vector architecture, or code scheduling is done on a superscalar processor to enable hardware parallelism. In Japan the shift from vector supercomputers to cluster architectures has made vectorization less important and MPI more important, although there is a strong interest in trying to retain the programming models from vector machines on current and future hardware.

SUPERCOMPUTER VENDOR SOFTWARE

NEC has a significant ongoing investment in HPF. They participated in the international HPF language definition effort and also on the Japanese HPF/JA design effort, which is described below. They also support MPI versions 1 and 2, and vectorizing compilers for Fortran, C, and C++. Their compilers use different strategies for C/C++ when compared to Fortran, and as a result they see much better vector performance from Fortran code than from C/C++. They also provide debugging and performance tools such as TotalView and Vampir. Starting with their SX-3 generation of processors, they use Unix as the operating system on their machines.

The operating system and compilers for the ES were provided entirely by NEC, although there are some ongoing research efforts at the ES facility that involve work in these areas. The machine has several programming models available to users, including message passing or HPF between processing nodes. While NEC supports all the more widely used MPI and OpenMP, it is also still investing in HPF and automatic parallelization, as illustrated in Figure 2.9 in Chapter 2. The MPI implementation has a latency of 5.6 microseconds and achieves a maximum bandwidth of 11.8 GBytes/sec, out of a hardware peak of 16 GBytes/sec. Within a processing node, the eight application processors communicate by shared memory, and the parallelism can be expressed using HPF or OpenMP, or the parallelization may be left to an automatic parallelizing compiler.

Fujitsu has a similar set of programming models, strongly influenced by their former vector product line. Fujitsu developed their own parallel data decomposition language, which is similar to HPF, but is unique to Fujitsu machines. In addition to the layout directives, this VPP Fortran language has parallel regions, barriers, and semaphores, and intrinsic functions to find the number of processors. The PrimePower systems are non-vector machines, but they support a similar set of tools, including an XP Fortran compiler, which is a port of the VPP Fortran compiler. The PrimePower systems also support MPI and PVM for communicating between nodes and they run a cluster operating system based on Score, which was developed as part of the Real World Computing Project.

The JAXA site is an interesting example of one of Fujitsu's major customers. JAXA has a long history of buying Fujitsu machines, to the extent that their working relationship with Fujitsu is more important than the architecture or software characteristics. Rather than prioritizing portability of their application codes across all architectures, they place a higher emphasis on backward compatibility with the software they wrote on the VPP vector line. Thus, their codes are written in Fujitsu's XP Fortran rather than the somewhat more portable HPF or much more portable MPI. This strong allegiance between vendors and customers is common in Japan.

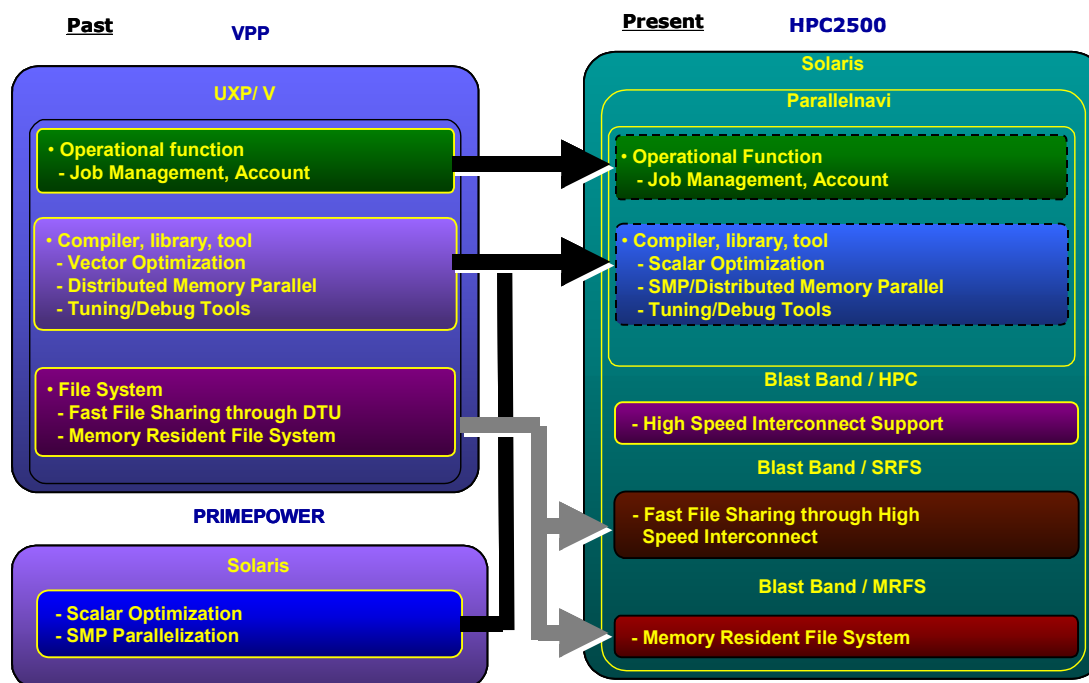


Figure 7.1. Programming models on Fujitsu machines (Courtesy Fujitsu)

Hitachi's supported programming models are similar to those of Fujitsu, because they have abandoned true vector processors in favor of the IBM Power processor line through a partnership with IBM. Also unwilling to give up the vector programming model on which their customers' code is based, Hitachi has pursued a line of pseudo-vectorization, with varying levels of hardware support to make vectorization easier. The current Power4+-based product has little hardware support for vectors, but continues to use the vector programming model within processing nodes. In addition to automatic vectorization, Hitachi continues to develop automatic parallelization techniques, which are also useful within a node. For the higher level, coarse-grained parallelism, they have offered MPI, PVM, and HPF, although the HPF effort is no longer active.

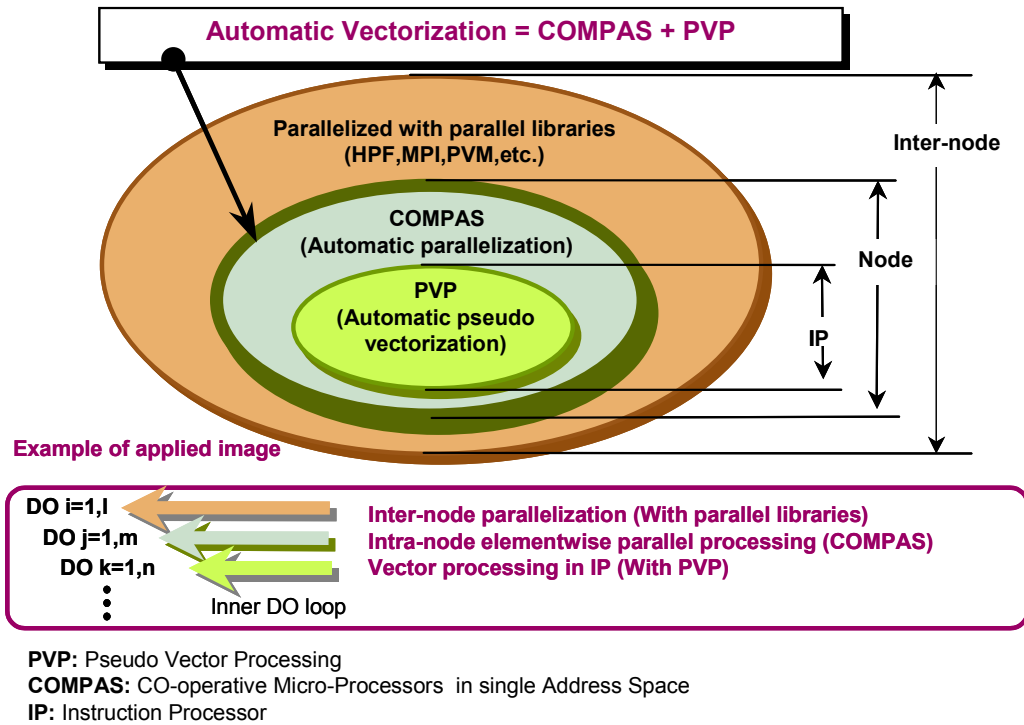


Figure 7.2. Parallel programming models for Hitachi machines (Courtesy Hitachi)

HPF/JA

The HPF effort in Japan involved significant compiler development efforts at the three main supercomputing vendors. In addition, though, some early applications experience was used to extend the language to make it more useful. HPF 2.0 extends the original HPF language for irregular applications, while HPF/JA is an orthogonal extension primarily altered for performance on applications with regular data structures. HPF/JA was developed by a Japanese consortium that included the three major vendors: NEC, Hitachi, and Fujitsu. NEC has an additional set of extension designs for the ES, with the resulting language called HPF/ES. Figure 7.3 gives an overview of the features in each of these languages.

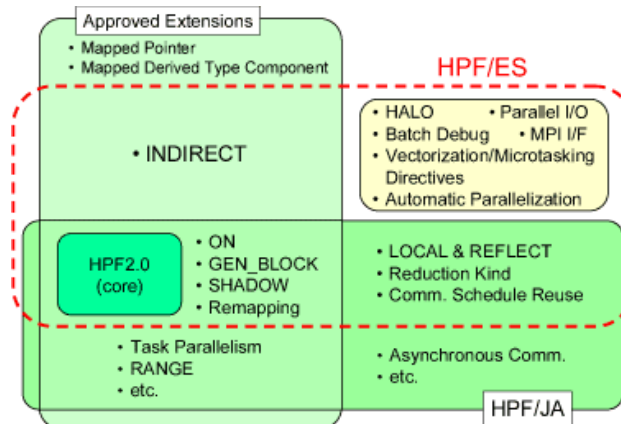


Figure 7.3. HPF/JA, HPF/ES, and HPF 2.0 extensions

Some of the key extensions in HPF/ES and HPF/JA include:

- REFLECT: a data layout used for nearest-neighbor computation.

- LOCAL: an assertion that no communication is needed within a given scope, thereby enabling better compiler optimizations with less sophisticated analysis.
- Extended ON HOME: partition computation replication which is useful in some problems if it avoids more expensive communication.
- HALO regions for updating ghost cells on unstructured meshed.

Of more general interest than the details of the language extension or particular performance results is the evidence of online parallel language activity in Japan, which lasted much longer than the HPF effort in the United States.

HPF ON THE EARTH SIMULATOR

The HPF language was used in at least two major application developments on the ES with remarkable results. The first is a plasma simulation code, IMPACT3D, which uses a Total Variation Diminishing (TVD) scheme programmed in HPF/EX. This code achieved 14.9 Tflop/s on a 512-node execution on the ES, which corresponds to 45% of the peak. This application earned the Gordon Bell Award for language in SC2002. Figure 7.4 shows visualizations of IMPACT3D results.

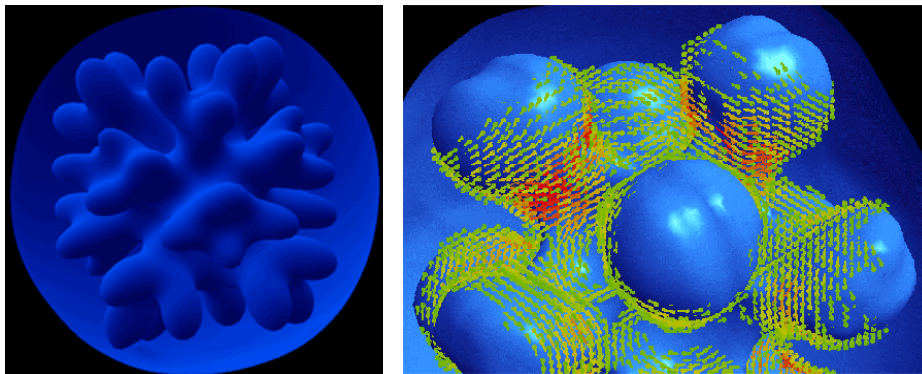


Figure 7.4. Visualization results from the IMPACT3D plasma code. The left picture shows the isosurface of density at the maximum compression of imploding targets in laser fusion; the right shows the pusher-fuel contact surface and vorticity.

The second major HPF/ES result is using the PFES code, an Oceanic General Circulation Model based on the Princeton Ocean Model. This code achieved 9.85 Tflop/s with 376 nodes (41% of the peak performance of the machine), using entirely automatic parallelization and vectorization within a node. This same code runs at 10.5 Tflop/s on the same number of nodes if some explicit parallelism is used within a node.

CONCLUSIONS

The software research agenda in Japan is currently skewed towards grid, which overlaps with cluster computing, and there are only modest research programs at the Japan Marine Science and Technology Center (JAMSTEC) for compilers, programming languages, and tools. HPF was much more successful in Japan than in the United States, probably due to a variety of factors:

- A reluctance of high-performance computing consumers to change programming languages, which leads to a demand for familiar vector programming models even on non-vector machines
- A good match between vector architectures and the fine-grained parallelism available in HPF. Message passing models are still used on these Japanese machines, but they are not required in all applications, and even when they are used, the extremely high-end vector/SMP compute nodes mean that the scale of parallelism at the MPI can be smaller relative to a commodity cluster.

- Longer sustained industrial projects to develop high-performance compilers, probably due to customer demands.
- Vision of the ES that included easier programming through HPF.

In spite of this relative success of HPF in Japan, especially on the ES, interest in the language has declined significantly. MPI is now the dominant model for programming most Japanese supercomputers today. The reason given by some vendors is that customers want portability of their code across machines. HPF does not run as well on non-vector hardware, and the vector hardware has become prohibitively expensive for at least two of the three Japanese supercomputing vendors. The investment in automatic parallelizing and vectorization compiles still seems to have paid off, because it allows the programmer to discover a smaller degree of parallelism at the coarse-grain level.

There was less work in supercomputing software than expected, which is explained by the interest in grid computing, which will be covered in the next chapter.

REFERENCES

PRIMEPOWER Benchmark Achievements – Fujitsu Siemens Computers.

<http://www.fujitsu-siemens.com/products/unix_servers/benchmarks.html> Last accessed February 23, 2005.