PPMLIB, A Scalable Library for Computational Fluid Dynamics

Grant No. DE-FG02-94ER25207

5/15/1994 – 5/14/1999

Final Report

Paul R. Woodward

12/14/2002

This grant has supported the construction of PPMLIB, a library of compiled code modules for use in fluid dynamical calculations. This library, along with documentation, is available to the scientific computing community on the LCSE Web site at www.lcse.umn.edu/PPMLib. A password protected companion Web site, www.lcse.umn.edu/PPMLIB, holds all the source code to the compiled modules. The library is intended for use in large-scale parallel computation. These two sites have been combined into a single site, www.lcse.umn.edu/ppmlib, which will make the source code available with GNU licenses and also displays links to other relevant Web sites.

When the original proposal for this grant was written, my team was making extensive use of MPP architectures such as the Connection Machine CM-5. For these architectures we developed a style of Fortran programming, called Fortran-P, that allowed automatic translation into parallel code in the SIMD computing model. However, shortly after the project was funded Thinking Machines went bankrupt and the SIMD architecture seemed endangered. Despite the production of the Cray T3D/T3E series, I felt that SIMD was too limiting and began to develop a more flexible style of parallel computation based upon a shared memory programming model. This model, which we now call SHMOD (for SHared Memory On Disk), is the base strategy of parallel computation behind the PPMLIB library. This code structure is ideally suited to the SMP cluster architecture, which was just emerging as the DoE support for this project began. SHMOD, especially when implemented using a component software architecture, now seems ideally suited to the Grid. This feature results from the latency tolerant design that was introduced to support use on SMP clusters. Development of SHMOD and tests of its applicability beyond the PPM codes are now supported by the NSF through its PACI program.

During the first two years of this project we settled upon the parallel programming paradigm that the library would support. This is SHMOD. A special feature of this paradigm is that code modules that embody the physics of the simulation are written for single CPU execution; the parallel aspect of the code is evident only at a higher level. This code construction allows a very practical separation of skills between people who create physics modules and those who create the outer parallel code that binds them together. PPMLIB includes a rich set of hydrodynamics modules that embody the latest advances in PPM numerical algorithm technology. Our group continually updates these modules, although at fairly wide intervals, as we improve our methods and as we add new ones to the set.

The parallel outer code that binds PPMLIB modules is made available in source. This strategy follows the "don't call me, I'll call you" rule. Code which must call modules in the library as well, perhaps, as any additional modules added by a code developer must be available in source format. This code constitutes what people now refer to as a framework, as opposed to a library. This outer parallel code comes in two hierarchical levels. First is the thread-parallel code that coordinates the CPUs of a single SMP. This code assumes a shared memory, but makes extremely sparing use of it. It is highly efficient. This outer template code is included in our team's sPPM benchmark code, used extensively by the DoE's ASCI program. It is probably best known to the community in this instantiation, because of the visibility imparted by the large ASCI procurements. Earlier illustrative templates placed on our Web site involved 2-D gas dynamics, which from its simpler nature makes the parallel computation techniques more easily understood.

DOE Patent Clearance Granted

Mark P. Dvorscak

Mark P. Dvorscak

(630) 252-2393

E-mail: mark.dvorscak@ch.doe.gov

Office of Intellectual Property Law

DOE Chicago Operations Office

2.6.03

Date

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

A second layer of outer parallel code coordinates the execution of the code on a number of SMPs that are interconnected on a network. The simplest version of this outer, outer code is the version in the sPPM code template. This code involves simultaneous communications not overlapped by computation. The ASCI program specifically requested that this sort of synchronous communication be used in sPPM so that this benchmark would be sensitive to the latency, bandwidth, and topology of the interconnection network. The SHMOD version of this parallel code, however, is tolerant of the expected high latencies and low bandwidths between SMPs on the network. This tolerance is achieved by the design principles given below.

The SHMOD implementation of PPM in PPMLIB has restructured the gas dynamics computation so that a small set of disk files contain all the data, save global constants such as the value of the time step, needed to update for a time step a rectangular solid grid subdomain. For a 2-D domain decomposition (of a potentially 3-D domain), there are 9 such disk files – one for the domain interior, 4 for the domain edges (or faces in 3-D), and 4 for the domain corners (or edges in 3-D). For a 3-D domain decomposition, there are 27 such disk files – one for the domain interior, 6 for the domain faces, 12 for the domain edges, and 8 for the domain corners. It is important to understand that these grid domains are intended to have a reasonable size, so that the computation can be efficient. We typically use grid bricks of 128 cells on a side. The difference stencil of the PPM algorithm then dictates that to update such a grid brick for a single time step the thickness of the face, edge, and corner blocks of “ghost” cells used to provide boundary conditions for this update must be 10 cells. Because the computation works its way through these ghost cells, vectorizable loop lengths progressively decrease from an initial value of $128 + 2 \cdot 10 = 148$ cells to a final value of just 128 cells. The result is an efficiency of 82%, defined as the ratio of non-redundant to total flops. In this case, 18% of the work done to update the grid brick would not have to be done if only a single, very large grid domain were updated by some single SMP with the necessarily immense memory. In practical terms, this efficiency measure is an underestimate, since modern design of CPUs with cache memory does not permit high performance updating of extremely long vector data. In fact, for present CPUs such as the Intel Pentium-4, peak vector performance is obtained at vector lengths around 128 for algorithms like PPM. Thus the grid brick updates used by PPM in the SHMOD framework are in fact highly efficient.

The restructuring of the PPM computation so that the grid domain is updated in a series of grid subdomain updates, each operating on a separate set of 9 or 27 disk files, allows a latency tolerant execution. Once all the data in the disk files for a subdomain has been prepared, that subdomain update is independent of any others (except for its dependence on a small set of global constants, such as the time step value). The data in these disk files is prepared by the subdomain update processes from the previous round of updates corresponding to the previous time step. Thus a given grid brick cannot be updated for this time step until the updates for the previous time step have been completed for it and for all 8 or 26 of its neighbors. To prevent pauses in execution it is therefore advisable to have substantially more grid bricks than participating SMP network nodes. We prefer to use roughly 4 times as many grid bricks as there are SMPs. For large problems this is not a limitation. For example, a grid of 2048^3 cells would be divided into 4096 grid bricks of 128 cells on a side. This could easily keep 1024 SMPs constantly busy. Smaller problems should be run on smaller numbers of SMP nodes if they are to run with equal efficiency. This feature of SHMOD has not proved to be an important limitation.

With the ratio of the number of grid brick subdomains to the number of SMPs on the network as large as 4 it is possible to insert a barrier at the end of each round of brick updates without a significant impact on performance. We have done this in the past for convenience, although it is not necessary, and our latest implementations do not have this feature. If the barrier is omitted, we must use a speculative value of the time step for each new round of updates, a procedure which rarely results in overestimates large enough to force a round of updates to be restarted (which of course is possible because the data for these updates is nonvolatile in its disk file format). The time step value is the result of a global reduction operation over the subdomains. It is not performed synchronously, and therefore a speculative value

based upon partial knowledge of the latest round of updates and global knowledge of the previous round is possible. This procedure eliminates the need for synchronous global communication and increases latency tolerance dramatically.

The communication of the 9 or 27 disk files to an SMP node on the network for its update task is overlapped with the update by that SMP of its previous grid brick task. Also, the communication of the 9 or 27 product files resulting from a grid brick update is overlapped with the SMP's next brick update task. Perhaps surprisingly, this communication does not require enormous network bandwidth to be completely overlapped with computation, and its requirements for network communication latency are so slight that they need hardly be considered at all. For example, on the dual processor Intel Itanium SMPs that our team now uses, 50 MB/sec link bandwidth is more than sufficient, and this can even be achieved using switched Gigabit Ethernet. Disk subsystem bandwidth at a node can easily match this level, since modern single disks are this fast. It is the near irrelevance of communication latency, traditionally defined, that makes the implementation with disk files (the D in SHMOD) practical. Nevertheless, it is important to note that modern operating systems automatically cache disk files. Hence, for problems small enough so that all the subdomain data resident at a node can be cached in SMP memory, communication latency and bandwidth delivered to a SHMOD application automatically become the latency and bandwidth of memory to memory communications. This is a very useful feature, because it is the smaller problems that can benefit from memory to memory communication latencies and bandwidths, and these problems are the ones that can fit their complete data contexts into SMP memory. Interestingly, this benefit for smaller problems comes automatically through the operating system's use of disk caches; the programmer need do nothing special to invoke it.

Both DoE and NSF supercomputer centers are beginning to implement cluster architectures that involve special storage nodes attached to storage area networks (SANs). Our team at the LCSE in Minnesota did pioneering work using SANs in the early days of Fibre Channel networks (under NSF and DoE ASCI support), and a small company, Sistina Software, split off from our lab to commercialize the shared file systems that SANs make possible. SHMOD is a natural parallel computing style on such cluster architectures, although it is even more natural when each SMP on the network has fast attached disks. The aggregate bandwidth of both the SMP interconnection network and of the SAN must be high in order to support complete overlapping of communication with computation in this new sort of arrangement, since more data transmissions become concentrated at the storage nodes. We are working with Karl-Heinz Winkler's team at Los Alamos and with Rob Pennington's team at NCSA to explore implementation of SHMOD in these new parallel computing environments. (We performed a SHMOD demonstration in the NCSA exhibit booth at Supercomputing 2002 using a scaled down system with this cluster architecture.) A benefit in this cluster architecture is a reduction in the aggregate disk subsystem cost over that of a SAN connected to all cluster nodes. RAID is commonly used as well, which increases reliability. For SHMOD applications, this cluster architecture allows storage of the task data contexts on a SAN that does not go down when a node fails. This permits a natural implementation of fault tolerance. With minor revision to the code's task scheduler, tasks that are not completed in a timely fashion can be cancelled and reassigned to another, more responsive SMP. Thus the computation can go right on in the face of a single or even a multiple node failure, and, in principle, it can migrate automatically to nodes that prove most responsive. These are new features that we are only now building into SHMOD, with NSF PACI support.

Our earliest test of SHMOD parallel computation on an SMP cluster used four 32-processor Origin SMPs at Los Alamos, in collaboration with Karl-Heinz Winkler's group and with support from the ASCI program. However, the relative inaccessibility of that computing system led us to move our main effort to NSF platforms. In 1997 and 1998 we experimented at NCSA with two 128-processor Origin SMPs interconnected by a disk subsystem which we provided and that was simultaneously attached to each machine via 4 Fibre Channel arbitrated loops. We implemented a raw file system on these disks, since no shared file system software was then available. We did a demonstration of this setup at the Super-

computing conference in the NCSA booth. Our later test implementations of SHMOD exploited a prototype shared file system, which we called the Global File System, on our lab's cluster of 4-processor SGI VisPC SMPs. Use of the shared file system made SHMOD much easier to implement, but it became clear to us that a shared file system could not be a requirement for SHMOD if it were to be broadly adopted. We have therefore developed a SHMOD communications library. It is very simple, and its main component is a remote I/O server that runs on each network node and that can either read and forward disk files at its node to another node or that can perform the reverse process, a remote file write. We have been testing and refining this remote I/O server software on NCSA clusters (and Myrinet interconnects), with NSF PACI support, over the last 2 years. We are now working with Karl-Heinz Winkler's team to implement it on his group's test clusters at Los Alamos. Our remote I/O server creates for a SHMOD application the particular functionality of a shared file system that it needs. It does this at the cost of making the application's task management software become aware of the network node locations of the disk files that hold its continually updated brick task contexts. This is not a huge task, because the domain decompositions we use are highly structured and simple. This simplicity is possible, because in SHMOD any network node can, in principle, update any task data context regardless of the location on the network of its disk files. The sequence of grid brick subdomain updates is thus automatically load balanced dynamically. Consequently, a simple domain decomposition is possible even if it might result in load imbalances from one brick to another.

It should be clear from the above discussion that our work that began with the PPMLIB project, funded through this grant from the Office of Science, has continued well beyond the time span of that grant. The reason for this is the extremely dynamic nature of supercomputing platform architectures. At the time the original PPMLIB proposal was written, we were computing on a Thinking Machines CM-5 with 2048 processors grouped into 512 nodes of a proprietary network. At present, we are using clusters of 2-processor nodes on third-party networks like Myrinet and Gigabit Ethernet. This has been a dramatic change that has greatly affected our work. The move of our community to embrace Grid computing is putting still further stress on teams like ours who are trying to maintain very highly efficient parallel code performance under unifying assumptions, expressed in fundamental application design, about the architecture of the high performance computing platform. It should therefore not be surprising that the code modules in PPMLIB that encapsulate the hydrodynamics functionality, and even those that express parallelism at the level of a single SMP, have been relatively stable, while a very dynamic development process has gone on involving the most outer code that coordinates the efforts of network nodes. PPMLIB was proposed as a research project, and it is the nature of research to be dynamic.

One of the research issues involved in the PPMLIB project was the question of whether a library like PPMLIB of compiled code modules made available without source code could be adopted by code development teams with the ease with which, for example, graphics libraries like OpenGL are adopted without source code. It is not clear to me whether this issue is technical, sociological, or cultural. Viewed as a technical issue, which is how I originally considered it, I believe there should be no large barrier to adoption. However, the experience of this project has convinced me that this issue is not entirely technical. Our team has tried without success to convince collaborating code development teams to use the PPMLIB software as compiled modules, but in every case we have had to make the source code available in order to proceed. The work had much of its intended benefit, since the software has indeed been used, and it did greatly ease the work of our collaborators. These collaborating teams were, most notably, Karl-Heinz Winkler's team at Los Alamos and the team of Bill Dannevik, the ASCI turbulence team (which provided some support to the project), at Livermore. Another team, named in the original proposal as desiring PPM software, that of Jim Glimm at Stony Brook, chose to write their own rather than experiment with compiled modules. Admittedly, Glimm's team had specialized needs due to their elaborate interface tracking software framework, but it is unfortunate that they were unable to benefit from PPMLIB. From this experience, I have come to the conclusion that the PPMLIB software should all be made available in source code with GNU licensing and am proceeding to do this.

Although it is perhaps not technically necessary, hydrodynamics tends to form the underlying framework of codes that incorporate with it other physics, such as combustion or radiative transfer. As a result, code developers believe that it is hydrodynamics code that must call other modules, rather than the reverse. This attitude is perhaps more based in the community's history and culture than in actual technical requirements, but the attitude persists regardless. Thus code developers feel that they must fiddle with our PPMLIB code modules, whether or not this is actually true. Consequently, to overcome this barrier, I have decided to make the source code available under GNU license in Open Source format.

Despite the difficulty that we found in promoting our library of compiled rather than source code modules for gas dynamics, we have had no difficulty in promoting PPM technology. These successes are appropriate to list here, since they relate to the robustness and value of our continued efforts along this line, even though those efforts are no longer funded through this project. The influence of our sPPM benchmark code, which provided the community with a template for efficient SMP cluster application design, should not be underestimated. In the early years of SMP cluster deployments, this template code was continually used as proof that this new computing architecture could give very high performance, and it is clear from my many conversations with people working with these computing systems that they had examined this template code and were familiar with its structure. Our demonstrations of parallel computation over the years with our PPM codes, using the PPMLIB code modules and later the SHMOD outer code framework, have, I believe, inspired a number of efforts. Perhaps most notable is the construction of the ASCI FLASH Center's code framework at the University of Chicago on a PPM foundation. The turbulence team at Livermore, working in collaboration with us, actively used the PPMLIB code modules to perform numerous large-scale simulations of Rayleigh-Taylor and Richtmyer Meshkov turbulent fluid mixing. Our demonstrations also convinced the RAGE code group at Los Alamos to engage with us beginning this year to incorporate into RAGE our PPM algorithm technology as well as to develop additional technology with us for the treatment of multifluid boundaries. An initial import of the contact discontinuity steepener from the PPM algorithm into RAGE was reported at the recent DoE code developer's conference. The PPM MHD algorithm is also being used in the NASA high performance computing program and is, I am told, being incorporated into Mike Norman's new astrophysics community code. Galen Gisler and Rob Coker at Los Alamos are also trying to incorporate the PPM MHD algorithm into the RAGE code. PPM is also widely used in the astrophysics community, not only through its use in the FLASH code but also through its use by Dave Arnett's and John Blondin's teams at the University of Arizona and at North Carolina State University.

Our demonstrations of PPM parallel computing under the SHMOD framework have sparked considerable interest in the NSF supercomputing program. One reason for this is the clear adaptability of this style of parallel computation to both loosely interconnected clusters and to the emerging Grid. Through the personal interest of Larry Smarr, the former NCSA Director, our team was led to demonstrate the effectiveness of the SHMOD outer code framework for applications other than PPM hydrodynamics. This activity, supported under the NSF PACI program, led to a collaboration with the meteorology team of Bob Wilhelmson, at the University of Illinois. We demonstrated an implementation of his COMMAS mesoscale meteorology code in the SHMOD framework in the NCSA exhibit booth at Supercomputing 2002. This implementation relied on a fundamental assumption underlying our initial PPMLIB proposal, namely that the program that should be used to update a grid subdomain and that which should update the entire problem are essentially identical. This is a feature that we called self-similarity of code design. We took the COMMAS code without modification and embedded it within our SHMOD framework. By enhancing slightly the thickness of the ghost cell regions around each grid subdomain, we were able to let the COMMAS code perform its boundary condition operations at the grid brick edges of interior subdomains without affecting the updates of those grid brick interiors. At the actual problem domain boundaries, we of course let COMMAS perform the boundary conditions in its normal way. The slight thickening of the ghost cell regions makes the overall parallel computation less efficient, but this efficiency can be improved, if desired, if the COMMAS developers rewrite the way in which their code treats its boundaries. Alternatively, the efficiency can be improved by making the

problem larger. This implementation strategy resulted in a quick and straightforward port of COMMAS to SHMOD, and immediate high performance and flexible parallel computation for this code on the NCSA Itanium cluster. Under our support from the NSF PACI program, we will experiment with this strategy using the much more complex WRF community meteorology code from NCAR. A primary benefit of this work will not only be an efficient parallel implementation but also the flexible and fault tolerant code execution features of SHMOD that were described above.

One additional aspect of the PPMLIB project is an effort to make available to the scientific computing community our team's data visualization software. This software can be used independently of PPM software or the SHMOD framework, although the parallel implementation of our visualization software uses SHMOD. Our initial effort in this area was to isolate and make available a set of PPM I/O routines within PPMLIB. These routines can be inserted into other codes as modules that take data in Fortran scalars and arrays and that write this data to dump files in the PPM compressed data format. These dump files can then be processed with utilities that we provide that can build files that we call bricks of bytes (or, in their latest version, hv-files). These files can be operated upon by our volume rendering utility, HVR (Hierarchical Volume Renderer). This visualization software is popular. Its earliest available version, BoB (for Brick of Bytes), was distributed through the University of Minnesota's Army High Performance Computing Research Center Web site. A later version, AnimaBob, was distributed through our LCSE Web site, and the present version, HVR, is available on this Web site as well. The combination of this visualization software together with the PPMLIB I/O routines makes it relatively easy for others to solve the data format problem. If they have data in their own preferred format, they can take a program that reads the data in, and then write it back out in our PPM format using the PPMLIB I/O routines. We have found this approach useful in working with the meteorologists at the University of Illinois, who use community data formats like HDF and netCDF. We have helped them to build data format translators, and it has been a simple matter. They have become fans of volume rendering as a result of using our HVR software. It should be noted that this latest version of our visualization software, HVR, available on our Web site, has been constructed under DoE ASCI VIEWS program support as well as NSF PACI support.

The persistent products of the PPMLIB research project are of course the software and documentation available on our Web site at www.lcse.umn.edu/PPMLib, at www.lcse.umn.edu/PPMLIB, at www.lcse.umn.edu/HVR, and, in a new integrated and updated version, at www.lcse.umn.edu/ppmlib. We have promoted this software in the community through demonstrations, talks at conferences, and direct collaborations. Although the results of the project did not turn out precisely as originally planned, this is the nature of research, and it is my judgment that the work reported here is and has been a success. It has already had a significant impact on our community, and, with the new Open Source distribution of the software coupled with the role that PPM has assumed over this time interval in community codes like FLASH, that impact should increase.