for CFA1

ABSTRACT OR SUPPORTING INFORMATION

Presentation/Publication Information:

Paper to be published in Proceedings of the Parallel CFD '99 Conference.
Talk to be given by James R. Taft at Parallel CFD '99,
College of William and Mary, Williamsburg, VA May 24, 1999.

Acknowledgments:

This work was supported under NASA contract 99-MRJ-7503-303-002].

Abstract:

Recent developments at the NASA AMES Research Center's NAS Division have demonstrated that the new generation of NUMA based Symmetric Multi-Processing systems (SMPs), such as the Silicon Graphics Origin 2000, can successfully execute legacy vector oriented CFD production codes at sustained rates far exceeding processing rates possible on dedicated 16 CPU Cray C90 systems.

This high level of performance is achieved via shared memory based Multi-Level Parallelism (MLP). This programming approach, developed at NAS and outlined below, is distinct from the message passing paradigm of MPI. It offers parallelism at both the fine and coarse grained level, with communication latencies that are approximately 50-100 times lower than typical MPI implementations on the same platform. Such latency reductions offer the promise of performance scaling to very large CPU counts. The method draws on, but is also distinct from, the newly defined OpenMP specification, which uses compiler directives to support a limited subset of multi-level parallel operations. The NAS MLP method is general, and applicable to a large class of NASA CFD codes.

MLP - A Parallel Programming Alternative to MPI for New Shared Memory Parallel Systems

James R. Taft
Sierra Software, Inc.
NASA AMES Research Center
M/S 258-5
Moffett Field, CA 94035
jtaft@nas.nasa.gov
(650) 604-0704

Abstract

Recent developments at the NASA AMES Research Center's NAS Division have demonstrated that the new generation of NUMA based Symmetric Multi-Processing systems (SMPs), such as the Silicon Graphics Origin 2000, can successfully execute legacy vector oriented CFD production codes at sustained rates far exceeding processing rates possible on dedicated 16 CPU Cray C90 systems.

This high level of performance is achieved via shared memory based Multi-Level Parallelism (MLP). This programming approach, developed at NAS and outlined below, is distinct from the message passing paradigm of MPI. It offers parallelism at both the fine and coarse grained level, with communication latencies that are approximately 50-100 times lower than typical MPI implementations on the same platform. Such latency reductions offer the promise of performance scaling to very large CPU counts. The method draws on, but is also distinct from, the newly defined OpenMP specification, which uses compiler directives to support a limited subset of multi-level parallel operations. The NAS MLP method is general, and applicable to a large class of NASA CFD codes.

1.0 Background

High Performance Computing (HPC) platforms are continually evolving toward systems with larger and larger CPU counts. For the past several years these systems have almost universally utilized standard off-the shelf microprocessors at the heart of their design. Virtually all hardware vendors have adopted this design approach as it dramatically reduces costs for building large systems. Unfortunately, systems built from commodity parts usually force researchers to embark upon large code conversion efforts in order to take advantage of any potentially high levels of performance. NAS was desirous of breaking this labor intensive barrier to performance. MLP was the result.

The MLP development focused on two codes, ARC3D and OVERFLOW. ARC3D was chosen because it utilized solvers commonly found in NASA CFD codes, and generally performed poorly on RISC based systems. ARC3D is a single zone code, and in essence performs a subset of the typical work done by OVERFLOW on many zones. It was essential that fine grained loop level compiler parallelism work well on at least 32 CPUs for ARC3D in order for MLP to be successful as a technique in OVERFLOW. This work exceeded expectations and resulted in excellent scaling to 128 CPUs. Figure 1 shows the results for a single zone 16M point problem.

The success of the ARC3D work was encouraging and led to the multi-level parallelization effort on OVERFLOW. OVERFLOW was chosen because it was a full production multi-zonal code that was in heavy use at NASA, and ran well on the 16 CPU Cray C90 system, sustaining about 4.6 GFLOPS on large problems. OVEFLOW was indicative of a "toughest" case at NASA.

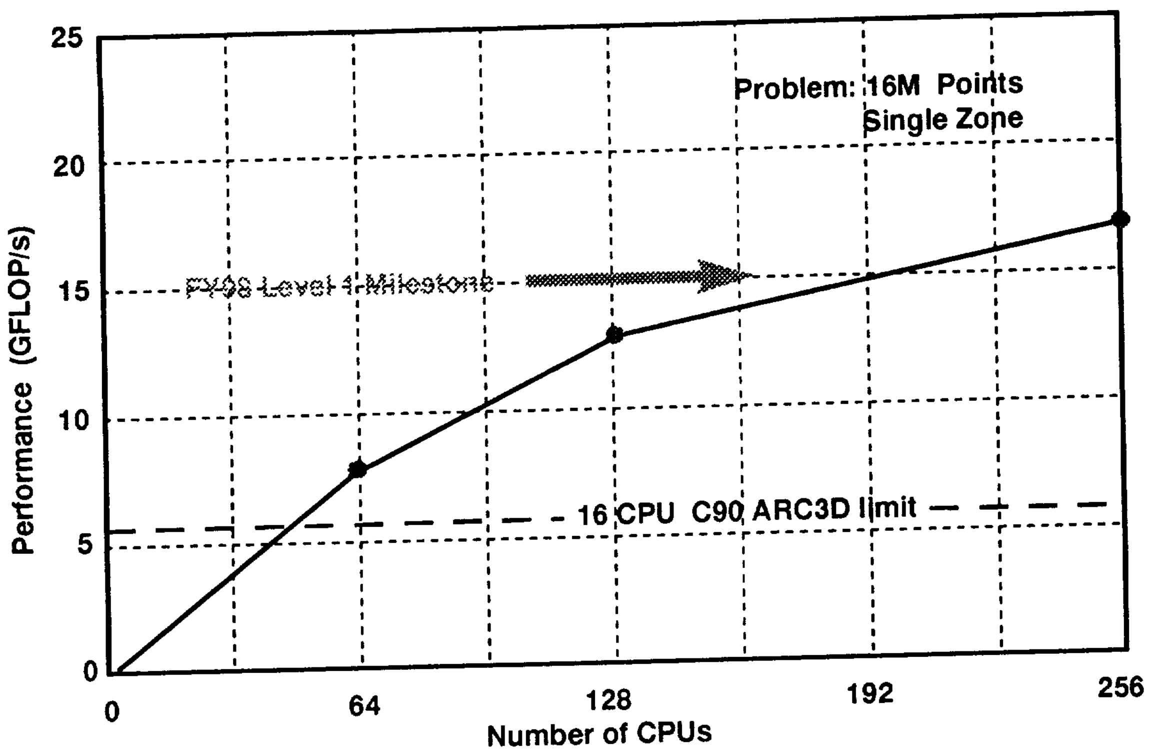

Figure 1 - ARC3D Performance versus CPU Count for Origin 2000

2.0 What is Multi-Level Parallelism (MLP)

Simple fine grained automatic parallel decomposition of application codes is not new. It has been utilized extensively for about two decades. It began with the introduction of the Cray Research XMP line of supercomputers. Much of the parallelism achieved on this machine was transparently provided by the compiler at the loop level, in which different iterations of the computational loops were executed in parallel on different CPUs in the system.

In the 80's computer budgets began to shrink and many researchers turned to an alternative model of parallel computation based on simultaneously executing many communicating independent parallel processes. This true "coarse grained" approach was ideally suited to executions on networks of inexpensive workstations. Performance however, was often elusive.

Coarse grained parallelism seriously began to be accepted in the community with the introduction of the platform independent Parallel Virtual Machine (PVM) message passing library from the Oak Ridge National Laboratory [1]. This was the standard for many years. Today, the most popular method of implementing this level of parallelism is via the PVM successor, the Message Passing Interface (MPI) library, from the Argonne National Laboratory [2].

Historically, codes decomposed with these message passing libraries were most often destined for execution on networks of single CPU workstations, or their topological equivalents, such as the Intel Paragon, Thinking Machines CM5, or IBM SP2. Applications developers spent substantial amounts of time attempting to decompose the problems so that communication between each CPU on the interconnect was at an absolute minimum. No thought was given to multiple levels of parallelism as the architectures simply did not support it.

With the advent of inexpensive moderately parallel RISC based SMPs from HP, DEC, Sun, and SGI, expanding to a second level of parallelism was possible. Users could decompose the problem at the coarsest level with MPI across SMPs, and use the compiler to provide fine grain parallelism at the loop level within an SMP via directives such as those within OpenMP [3]. In general however, clusters of SMPs were still treated as a series of discrete single processor entities, and MPI messages were still exchanged between CPUs even within a single SMP.

While the MPI/OpenMP hybrid approach is potentially better at scaling than the pure MPI solution, the approach has the major drawback that it is still subject to the relatively high MPI latencies whenever messages are used. More importantly, it requires a major rewrite of the code to fully decompose the problem for coarse grained parallel execution.

3.0 What is Shared Memory MLP

Very recently, manufacturers have adopted a new architectural design philosophy resulting in a hierarchical SMP that supports very large CPU counts (>100), albeit with non-uniform memory access (NUMA). The Origin 2000 system from SGI is such a system. For many applications 100 CPUs is more than enough computational power to solve the problem in a reasonable timeframe, and the need to traverse multiple SMPs to achieve the desired level of sustained performance is not necessary. This opens the door to some interesting possibilities. In particular, the high latency HiPPI connections between SMPs can be neglected as there is only one SMP involved. MPI can also be dropped as there is no need to spawn processes on other SMPs, and there are much simpler ways of spawning them on a single SMP.

Given a true SMP architecture and a problem that fits within it, one can define a new way of performing multi-level parallel executions. To distinguish it from past approaches, we define it as Shared Memory MLP. It differs from the MPI/OpenMP approach in a fundamental way in that it does not use messaging at all. All data communication at the coarsest and finest levels is accomplished via direct memory referencing instructions. Furthermore, shared memory MLP is different from just OpenMP (when used in its limited multi-level mode) in that it makes extensive use of independent UNIX processes and shared memory arenas to accomplish its goals. These features are not supported by OpenMP. Both of these features allow shared memory MLP to provide superior performance to the alternatives. More importantly, they provide a simpler mechanism for converting legacy code than either OpenMP or MPI.

For shared memory MLP, the coarsest level of parallelism is not supplied by spawning MPI processes, but rather by the spawning of independent processes via the standard UNIX fork, a system call available on all UNIX systems. This is a much simpler method in that the user simply makes fork calls at any time in the execution of his program to create another process. The user may spawn as many such processes as desired, and each of the processes can execute on one or more CPUs via compiler generated parallelism. The advantage of the fork over the MPI procedure is that the forks can be inserted well after all of the initialization phase of a typical CFD code. Thus, the user does not need to dramatically alter and decompose the initialization sections of major production codes, a daunting task at best.

Once the forks take place, all communication of data between the forked processes is accomplished by allocating all globally shared data to a UNIX shared memory arena, another system call available on all UNIX RISC systems. Again this is a simple process and results in a dramatic reduction in communication latencies over MPI. By using the arena approach, all global communication takes place via memory load and store operations requiring just hundreds of nanoseconds, not the tens of microseconds typical of MPI messaging latencies.This dramatic 50-100 fold reduction in data access times provides the support needed for greatly enhanced parallel scaling needed in typical applications.

4.0 OVERFLOW-MLP

The shared memory MLP recipe described above is very apropos for the field of CFD. In particular, it is ideally suited for CFD computations that utilize multi-zonal approaches in which the total computational domain is broken into many smaller sub-domains. Several production CFD codes at NASA utilize this solution approach. OVERFLOW is one of them.

OVERFLOW was chosen as the test bed to examine the performance, ease of use, and robustness of the MLP technique. It is one of the largest consumers of machine resources at NASA sites. OVERFLOW is a 3D RANS code solving steady and unsteady flow problems of interest.The code consists of approximately 100,000 lines of FORTRAN. It is heavily vectorized, and has historically executed well on the C90 systems at NAS. Typical sustained performance levels are around 450 MFLOPS per processor, with sustained parallel processing rates of around 4.5 GFLOPS on dedicated 16 CPU C90 systems. As such it is considered a good vector/parallel code.

Shared memory MLP was inserted into OVERFLOW by constructing a very small library of routines to initiate forks, establish shared memory arenas, and provide synchronization primitives. Calls to these routines were inserted as needed into the C90 version of the code. The initial effort to convert OVERFLOW to MLP required only two man-months and a few hundred lines of code changes. The effort involved slightly modifying the main program, and six other routines out of the nearly 1000 routines in the code.

The main calculational sequence in OVERFLOW is a series of loops over time and grids. The major change for OVERFLOW-MLP is to sub-divide the grid loop in such a way that multiple independent MLP processes each handle a separate subset of the total number of grids., Thus, the grid loop is done in a coarsely parallel fashion. All initialization and wrapup tasks remain unchanged from the C90 code, as do all of the solvers, etc. The MLP processes performing the work only need to communicate boundary data at a few key points in time during the course of the calculation. The remainder of the time is spent doing computations totally independent of each other.

Figure 3 depicts the MLP layout of the data and communication occurring within the Origin 2000 architecture. Each MLP process is assigned a given number of CPUs. The CPU count for each process is determined from a load balance analysis at run time that attempts to keep the number of points solved by each process about the same. Each process solves only those grids assigned to it. The grids for each process are allocated to memories close to the CPUs executing the MLP process assigned to the grids. The boundary data is archived in the shared memory arena by each process as it completes its processing of a grid. Other processes read this data directly from the arena as needed. At the end of a time step all processes are synchronized at a barrier, and the procedure repeats for each time step taken.

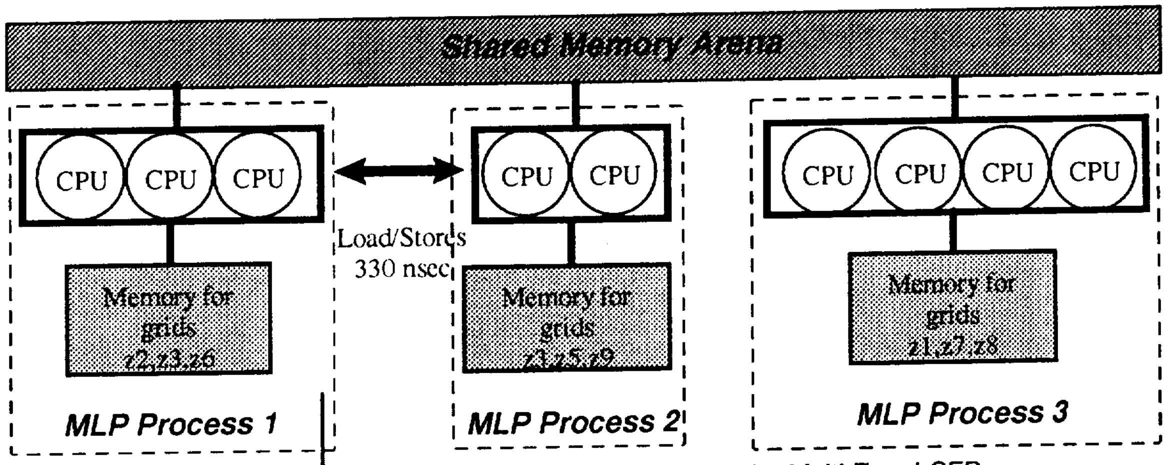

Figure 3 - Shared Memory MLP Organization for Multi-Zonal CFD

Doing the computation of zones in parallel is not new. In fact the MPI version of OVERFLOW already does this. The unique feature of the shared memory MLP approach is that it does so with no message passing and only a few hundred lines of code changes. The MPI implementation requires approximately 10,000 additional lines of code. The end result is that the MLP code is simpler to maintain, continues to execute well on C90 systems, and now executes well on parallel systems at very high sustained levels of performance as seen below.

5.0 OVERFLOW-MLP Performance Results

The major focus during the development of the MLP technique was on obtaining efficient parallel scaling. It is a fact that all of the best RISC based microprocessors rarely achieve in excess of 100 MLFOPS per processor on typical production CFD codes. Memory access is almost always the inhibitor to higher levels of single CPU performance. Thus, unless a large CFD problem can scale to more than a hundred processors, sustained computation in excess of 10 GFLOPs is not likely. At least 10 GFLOPS is needed on the important large problems of today in order to solve them in an acceptable time frame.

It was clear that the MLP technique offered the promise of a tremendous reduction in communication latencies over an MPI implementation. In order to stress test the technique to the fullest, a large real production problem was selected that fully exercised OVERFLOW's typical options for solvers, smoothers, and turbulence models. The problem selected consisted of 35 million points divided among 160 3D zones. The zones varied in size from ~1.5 million points to ~15 thousand points. A total of 10 time steps were executed on various numbers of CPUs. Figure 4 shows the results of this test.

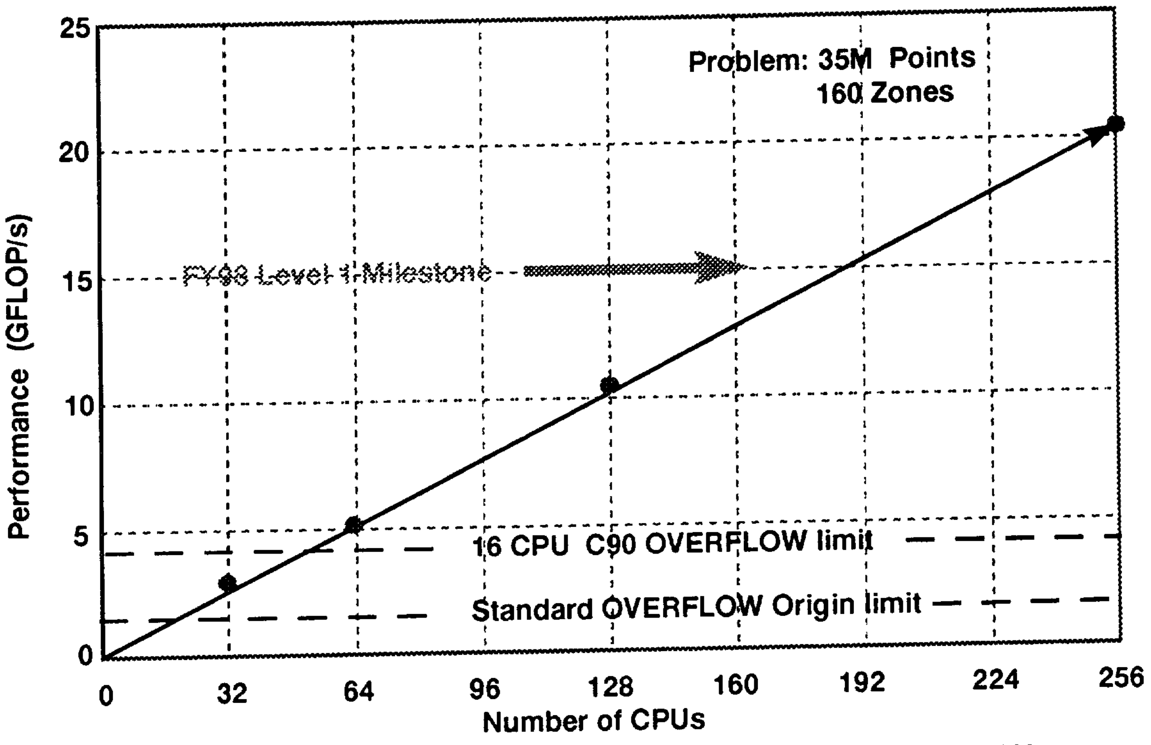

Figure 4 - OVERFLOW-MLP Performance versus CPU Count for Origin 2000

As can be seen the performance scales almost perfectly linearly with increasing processor count. Performance on 64 CPUs is about 5 GFLOPS. Performance on 128 CPUs is about 10 GFLOPS, and performance on 256 CPUs was 20.1 GFLOPS. Performance per CPU remained steady at about 80 MFLOPS.

The fact that OVERFLOW-MLP is a pure vector code and yet executes at sustained performance levels in excess of 20 GFLOPS on RISC systems is remarkable. Essentially this indicates that the new RISC systems will be able to significantly extend the performance envelop for large vector oriented production CFD codes for the first time, a very important feature as we enter the transition period from vector to RISC over the next few years.