Simulation of Compressible Flow on a Massively Parallel Architecture

DAN WILLIAMS AND LUC BAUWENS

Department of Mechanical Engineering, The University of Calgary, 2500 University Drive N.W., Calgary, Alberta T2N 1N4, Canada; e-mail: {dnwillia, bauwens}@acs.ucalgary.ca

ABSTRACT

This article describes the porting and optimization of an explicit, time-dependent, computational fluid dynamics code on an 8,192-node MasPar MP-1. The MasPar is a very fine-grained, single instruction, multiple data parallel computer. The code uses the flux-corrected transport algorithm. We describe the techniques used to port and optimize the code, and the behavior of a test problem. The test problem used to benchmark the flux-corrected transport code on the MasPar was a two-dimensional exploding shock with periodic boundary conditions. We discuss the performance that our code achieved on the MasPar, and compare its performance on the MasPar with its performance on other architectures. The comparisons show that the performance of the code on the MasPar is slightly better than on a CRAY Y-MP for a functionally equivalent, optimized two-dimensional code.

1 INTRODUCTION

We have ported a computational fluid dynamics application to the MasPar MP-1. The application uses the flux-corrected transport (FCT) algorithm [1], and consists of a set of Fortran subroutines that are collectively referred to by the name of the main subroutine, LCPFCT. In addition to the main subroutine, several auxiliary subroutines can be used to define the geometry, source terms, and boundary conditions. The code solves the coupled sets of multidimensional nonlinear conservation laws that describe reactive and nonreactive gas dynamics.

LCPFCT itself can handle Cartesian, cylindrical, spherical, or user-defined coordinate systems. Alternate sets of boundary conditions can be selected by making the appropriate choice of arguments to the subroutine calls. Besides inflow, outflow, or reflecting wall conditions in any coordinate system, LCPFCT can also handle periodic boundary conditions. Multidimensional problems are solved using the method of fractional steps [5]. The computational grid can be nonuniform, and can move during a time step, allowing the user to perform Lagrangian or sliding rezone calculations.

This article describes our porting and optimization of FCT on the MasPar. First we give a general description of the flow model and the FCT algorithm. Second, we give a brief overview of the MasPar architecture. Third, we briefly describe the two-dimensional blastwave problem with periodic or solid-wall boundary conditions which we use as a benchmark. Fourth, we discuss the hurdles involved in adapting the basic LCPFCT compressible fluid dynamics module, to a form compatible with and efficient on the MasPar. Finally, we summarize the performance of our code on the

Received May 1994
Revised December 1994

Scientific Programming, Vol. 4, pp. 193–201 (1995)
CCC 1058-9244/95/030193-09

MasPar. We compare the results of running our two-dimensional code on the MasPar to equivalent benchmarks on other architectures including the CRAY Y-MP and the Connection Machine CM-2.

2 THE FLOW MODEL AND ALGORITHM

Our flow model is based on the inviscid, time-dependent, multidimensional Euler equations of gas dynamics. This includes three conservation laws, for mass, momentum, and energy. In the inviscid model, shocks appear as mathematical discontinuities. The solution that we seek is the weak solution to the differential equation that satisfies the integral form of the equations. From a physical standpoint, in the presence of shocks, the integral form is the proper form of the conservation statement. The FCT algorithm is used to solve each of the coupled conservation equations defining the gas dynamic system.

FCT is an explicit, nonlinear, monotone method designed to enforce positivity and causality on the numerically computed solution. These constraints ensure that the numerical solution approximates the solution of the conservation laws in integral form. Positivity and causality would be lost in the numerical approximation if one did not ensure that the finite-difference scheme is conservative. FCT satisfies the monotonicity requirement: it implements profile-dependent nonlinear corrections to the truncation-error terms, which ensures that no new numerically produced maxima or minima occur near shocks or contact discontinuities. The FCT algorithm ensures fourth-order phase accuracy in smooth regions of the flow and guarantees conservation, monotonicity, and positivity in regions with steep gradients. An extensive discussion and analysis of FCT has been published by Boris and Book [1].

FCT is second order accurate in space. Second order accuracy in time is achieved by splitting the time step. First a half time step computation is executed and then the intermediate, time-centered values of the physical variables are used to evaluate the source terms for the full time step. To ensure second order accuracy, the time step must be small enough so that the cell averaged values of the physical variables do not change appreciably during the time step. Direction splitting [6] allows for multidimensional FCT calculations. In a two-dimensional problem this is accomplished by separating each gas dynamic equation into its respective x and y parts. First each y-direction column in the grid is integrated, and then the x-direction rows are integrated. Direction splitting creates a bias that will eventually break the symmetry of a solution, depending on which direction is integrated first. This can be eliminated by performing two calculations, x-y then y-x, and averaging the results.

FCT is a "uniform" algorithm in which, at each time step every cell undergoes the same numerical operations regardless of the values of the physical variables in that cell. It is therefore ideally suited for massively parallel processing on a single instruction, multiple data (SIMD) architecture such as the MasPar.

FCT is also well suited for simulating high-speed compressible flow. We are interested in performing simulations that include combustion. Other numerical schemes such as PPM piecewise parabolic method (PPM) [2] or monotonic upstream-centered scheme for conservation laws (MUSCL) [11] would be appropriate for combustion simulations, but we have chosen FCT because of its simplicity and its good track record in combustion. It has been used successfully for combustion simulation by other researchers such as Oran et al. [6, 7, 8], Thibault et al. [10], and Zhang et al. [13].

3 THE MASPAR

The MasPar system architecture includes a processor element array, an array control unit, and a UNIX workstation as a front end. The front end manages program execution and user interface. When there is a need for parallel execution, the front end sends the program for execution to the processor array. The 8,192 processors are organized in a two-dimensional array topology (128 × 64), in a SIMD architecture. In a SIMD architecture, all the processors simultaneously perform the same operation on different data as one single stream of instructions is broadcast to all processors by the array control unit. The MasPar MP-1's theoretical peak performance is 650 Mflop/s single precision and 290 Mflop/s double precision.

There are three types of communication on the MasPar. First, there are communications from the array control unit to the processor array where the array control unit broadcasts data or instructions to all processors in the array simultaneously. Second, nearest neighbor data communications are carried out by the X-net. The X-net is an eight-

way, two-dimensional toroidal mesh that allows a processor to communicate with its nearest neighbors. Finally, communication between arbitrary processors is carried out by a hierarchical crossbar called the global router.

Interprocessor communication is required when a processor requires data that are not resident in its local memory. X-net communication will be used if the data reside on a neighboring processor. In this case, the performance penalty is low because X-net communication is fast. If the data do not reside on a neighboring processor, global router communication will be used. While router communication is efficient, it is much slower than X-net communication—the X-net has approximately 16 times the bandwidth of the router.

Another more costly type of communication is array sloshing between the front end and the processor array. Array sloshing has a profound effect on performance and occurs in several circumstances. First, if an array that has been allocated on the processor array is accessed in a serial (Fortran 77) manner, it will be sloshed to the front end. To avoid this situation, serial access on the processor array should be avoided. Likewise, when an array has been allocated on the front end, and a subroutine is called that uses this array in a parallel context, the array will be sloshed from the front end to the processor array and back again when the routine exits. To avoid this, compiler directives can be used to force allocation on the processor array at declaration time. Even more costly than the latter two cases is the sloshing of a COMMON block of arrays. This can be avoided in the same manner as for individual arrays.

4 TWO-DIMENSIONAL BLASTWAVE COMPUTATION

The problem used to benchmark FCT on the MasPar is a two-dimensional blastwave computation with periodic boundary conditions. This computation involves both supersonic and subsonic flows with interacting shocks and a high degree of symmetry. The blastwave problem may not be a significant real-world problem, but it is a good benchmark for computational fluid dynamic (CFD) codes. It is a good benchmark because the solution is very symmetric, it maintains this symmetry for a long time, and a good CFD code should resolve the shockwaves within a few cells while maintaining the symmetry of the solution. Also, this problem was previously used by Oran et al. [5], and using the same problem allowed us to compare our results.

The computation is initialized with a high-density, high-pressure square of fluid that is 32 cells on each side. The square is situated in a doubly periodic mesh 128 cells on each side. The square region in the center of the domain begins with a density 15 times the background density and a pressure 30 times the background pressure. The contact surface that defines the interface between the initial high pressure material and the low pressure material is tracked by using an additional species variable in the computations. This computation is a good benchmark for a CFD model because it should retain its symmetry for a long time, and because periodic or solid wall boundary conditions should give the same solution as long as the symmetry is maintained.

Figures 1 and 2 show contours of pressure and location of the contact surface for several time steps during the simulation. During the simulation, the up–down symmetry is eventually broken by round-off error arising from the limited precision of the floating-point calculations. Symmetry across the 45° degree diagonals is eventually broken by the truncation errors in the time step splitting. These errors appear to be larger than the round-off errors.

When the unconfined high-pressure gas is released, a shock forms that races out from the edges of the initial square. The contact surface closely follows behind the shock. Figure 1 shows the development of the pressure contours for six time steps. By step 500 the initially square shock has progressed through a circular phase and continues to change shape. By step 1,500 the shock has reflected from the ends of the computational domain and has begun to recompress the material inside the contact surface. As time proceeds to step 10,000 the shocks become progressively weaker, and become oriented parallel with the sides of the domain.

Whereas the shock patterns become simpler as time progresses, the vorticity caused by the shock interactions with the contact surface warps the interface into increasingly complex patterns. The vorticity is generated by the baroclinic source term in the vorticity equation. This term is nonzero when the gradients of density and pressure are not aligned. The misalignment is created by the shock reflecting from the corners of the domain, which

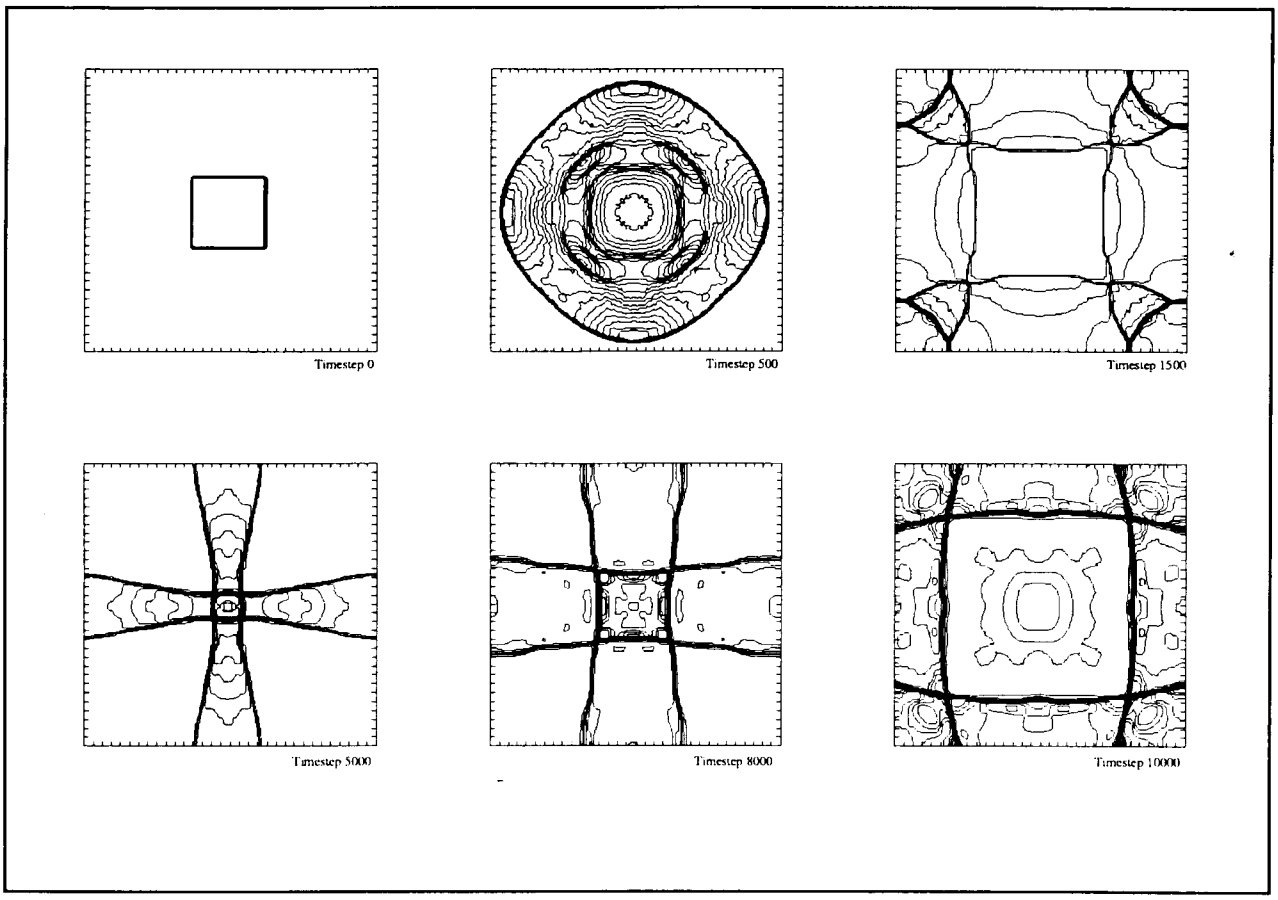

FIGURE 1 Pressure contours at six time steps for the square blastwave problem.

weakens the shock. The shocks reflect back through the interface creating more vorticity.

5 PORTING AND OPTIMIZING FCT

The MasPar can be programmed in either MPL, a parallel extension to C, or MPF, an implementation of a subset of Fortran 90. We used version 3.012 of MasPar's high-performance Fortran compiler "mpfortran" or MPF. In MPF, parallel operations are expressed with the Fortran 90 array extensions. Arrays are treated as unitary objects rather than requiring them to be iterated through one element at a time, as in standard Fortran 77. MPF generates code for both the front end and the processor array, effectively making the details of the architecture transparent to the programmer. However, programming style will directly affect performance.

Optimization of FCT on the MasPar required two main issues to be addressed: making effective use of the processor array and minimization of communication cost. In SIMD architectures, operations performed on a subset of the processor array, such as single lines or columns, or boundary nodes, cost as much in cycles as operations on the whole array. Thus, constructs detaling, for instance, with the boundaries separately from the main arrays can easily double the computation time, and they are advantageously concatenated with the main operations. In addition, unwanted interprocessor communication can occur if arrays are not correctly allocated on the processor array. It was critical to ensure that all arrays were properly aligned on the processor array in order to minimize communication overhead.

Porting the code involved getting the code to run on the processor array, and optimizing the code on the processor array. First, since the subroutines were originally written in Fortran 77, they were converted to Fortran 90 using the MasPar version of Pacific-Sierra's VAST-2.4.01L (DPVAST) translator. The translator searches the

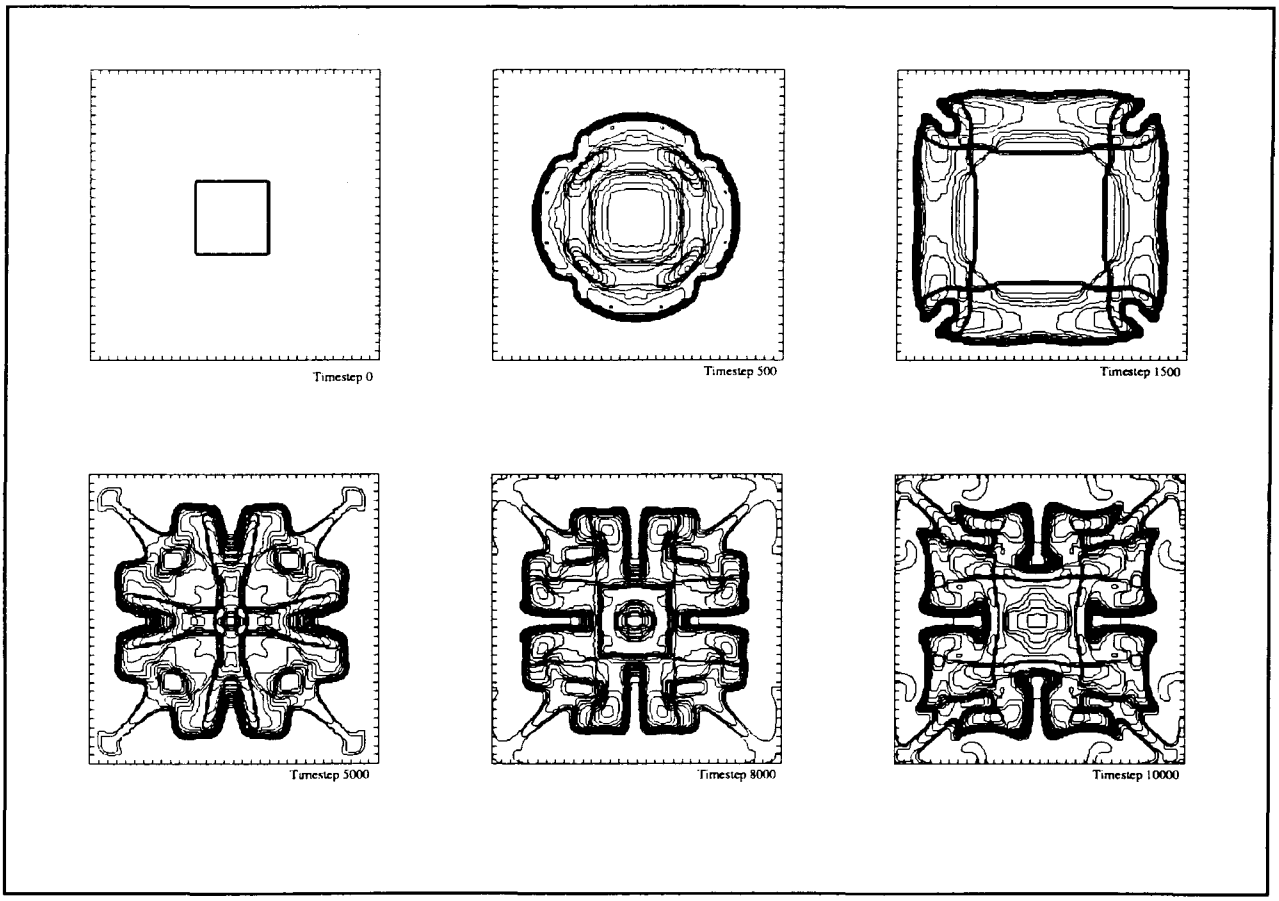

FIGURE 2 Contact surface locations at six time steps for the square blastwave problem.

Fortran 77 code for scalar "do" loops and converts them to the Fortran 90 array notation, which is understood by the processor array. The code as translated by DPVAST ran very poorly—the temporal performance was much less than one time step per second.* Optimization of the code was carried out in several stages described below, and the results of optimization are summarized in Table 1.

Stage one dealt with COMMON block allocation. The original blocks contained both scalar and array data, and as a result, the compiler allocated them on the front end. Since the arrays in the blocks were used in Fortran 90 array constructs, the whole block had to be sloshed to the processor array. By removing all COMMON blocks, and passing the block elements as parameters to the subroutines, we improved the temporal performance by a factor of 2. This was still unsatisfactory.

Stage two consisted of altering the implementation of direction splitting. The original LCPFCT subroutines were one dimensional. Two-dimensional problems were handled by calling the one-dimensional subroutines row by row, and then column by column. This minimizes scratch memory—a desirable feature on a vector machine, but leads to poor performance on a SIMD architecture. For example, for a 128 × 128 grid, a row or column array will contain only 128 elements. When the one-dimensional subroutines are called with a 128-element array only 128 processors will be used at a time, resulting in a performance that is less than 1.0% of the peak perfor-

* Hockney [4] defines temporal performance as the inverse of the execution time [$R_T = T^{-1}(\mathbf{N};\mathbf{p})$], where **N** is the problem size and **p** is the number of processors. Temporal performance is measured in solutions per second (sol/s) or time steps per second (tstep/s). It is a good metric for comparing different algorithms when solving a certain benchmark problem because it tells you which algorithm solves the problem the fastest.

Table 1. Effect of the Optimization (Resolution: 128 × 128)

Stage	Temporal Performance (tstep/s)	Benchmark Performance (Mflop/s-64 bit)	Time to Perform Optimization (hr)
DPVAST	0.03	0.3	2.0
1	0.06	0.7	3.0
2	3.03	35	15.0
3	4.55	46	3.0
4	5.56	63	3.0
5	5.88	68	2.0
6	7.14	81	1.0

mance of the machine. The code was rewritten such that all the rows were dealt with in a single operation and then all the columns. This can be done because adjacent rows or columns are independent of one another during this computation. This stage improved the temporal performance to approximately 3 tstep/s.

The profiler was used in stage three to find out which parts of the code used the most CPU time. The profiles showed that approximately 70% of the time was spent in the LCPFCT subroutine, and a large percentage of the time was spent in calculating the boundary conditions. The DPVAST version of the code used two one-dimensional arrays to store the results of the boundary calculations. These arrays could not be properly aligned with the main two-dimensional arrays. To eliminate the communication penalty that results from this situation, the code that dealt with the boundary calculations was rewritten, and the results of the boundary calculations were stored in the main two-dimensional arrays. In addition, in the original implementation, periodic boundary conditions were controlled by setting the value of a double precision flag to either zero or one. The code was modified to use a logical flag and an IF-THEN-ELSE block instead. As a result of the modifications performed in this stage the temporal performance improved to 4.55 tstep/s.

Stage four consisted of replacing some of the subroutine array parameters with COMMON blocks. As a result of eliminating the COMMON blocks in stage two of the optimization, a significant amount of overhead had been created from passing a large number of array parameters to the subroutines. The original COMMON blocks contained both array and scalar data. COMMON blocks that contain only arrays, and no scalar data will be properly allocated on the processor array. Replacing some of the subroutine array arguments with COMMON blocks reduced the overhead. This stage further improved the temporal performance to 5.56 tstep/s.

In stage five, analysis of the code profiles indicated that some overhead was being incurred due to router communication, because some of the subroutine calls included arguments that were parts of arrays. This situation is handled by copying into a temporary array through use of the router, which is time consuming. The code was modified to eliminate the router operations and temporal performance increased to 5.88 tstep/s.

Further improvement was obtained in stage six by aligning all array allocations. The current compiler implementation maps arrays directly on the processor array based upon declaration indices, without any optimization attempt. Some of our arrays needed to include boundary nodes, while some others did not, and we had declared them with their minimal size, which caused unnecessary shifting of whole arrays to occur, which is costly and can be avoided by proper array declaration. By changing the declaration indices we aligned the array allocations and improved the temporal performance to about 7 tstep/s.

6 RESULTS AND DISCUSSION

Table 1 presents a summary of the effects that each stage of optimization had upon the performance of FCT. The benchmark performance was calculated as suggested by Hockney [4].† The

† Hockney [4] defines benchmark performance as the ratio of the benchmark floating-point operation (flop) count $F_b(N)$ and the execution time $T(N;p)$ [$R_b = F_b(N)/T(N;p)$]. The benchmark performance generally has units of million floating-point operations per second (Mflop/s). The benchmark flop count for our code was obtained by counting operations also as suggested by Hockney [4] (+, −, × = 1 flop; ÷, √ = 4 flop; exp, sin, cos, etc... = 8 flop; if (x .rel. y) = 1 flop).

Table 2. FCT Benchmark Performance for Different Grid Sizes

Grid Size	Single Precision		Double Precision	
	R_B (Mflop/s)	% peak	R_B (Mflop/s)	% peak
128 × 64	146	24	90	31
128 × 128	108	18	81	28
256 × 128	126	21	87	30
256 × 256	138	23	93	32
512 × 512	174	29	Insufficient memory	

largest improvement in performance was obtained in stage two, which required the code to be completely rewritten. The other stages resulted in modest improvements, but were much less time consuming than stage two. Overall, optimization of FCT on the MasPar was by far the most time-consuming task, while the translation process (DPVAST stage) was relatively straightforward. Table 1 effectively demonstrates the significance of optimization. By modifying the way in which the code was written we improved the performance by a factor of approximately 240 over the DPVAST version. At this point however, further improvement would require rewriting a fully two-dimensional version of FCT for the MasPar, not using direction splitting.

Table 2 shows the benckmark performance achieved with the optimized version of the code. The first row of Table 2 (128 × 64) corresponds to a problem with an array size that can be mapped optimally onto the processor array. In this case there is a minimum in communication overhead because none of the arrays have to be mapped into multiple layers of processor memory. For the 128 × 128 case a drop of about 25% in single precision, and about 10% in double precision performance is observed. This would be due to the communication overhead created by one extra layer of memory having to be allocated. Table 2 also shows the percentage of the peak performance attained by FCT. Since we have implemented two-dimensional FCT computations using direction splitting, the peak performance that our code could attain is reduced by at least a factor of two. This is because the rows and columns of the grid are dealt with separately. With a fully two-dimensional version of FCT this problem could be fixed, but would require substantial rewriting of our original code. Taking this into consideration our results are quite good.

Figure 3 shows the dependence of benchmark performance on the problem size. After the initial

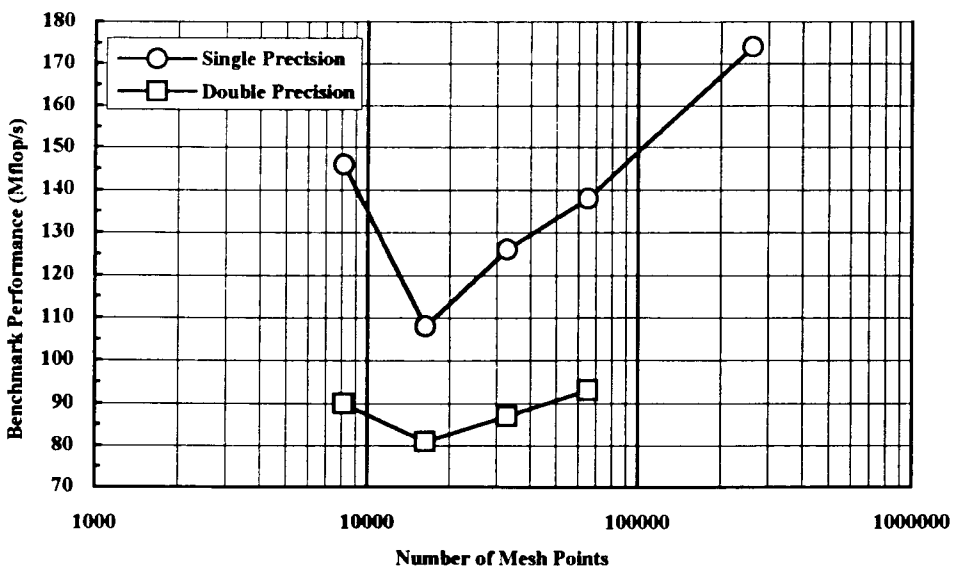

FIGURE 3 Dependence of benchmark performance on the problem size.

drop in benchmark performance, when changing from an 128 × 64 to a 128 × 128 grid, the performance increases at an approximately logarithmic rate as the problem size increases. This is because the ratio of floating-point operations to communication overhead is increasing. The incremental changes in benchmark performance seem to be much more dramatic for the single precision than double precision, because communication overhead is independent of the precision.

Timings and floating-point performance for FCT on various architectures are presented in Table 3, which includes the current figures and data from Oran et al. [5]. Timings are shown for two typical vector supercomputers, the CRAY Y-MP and the Fujitsu VPX240. The timing for the MasPar as compared to the Cray is competitive, with the MasPar approximately 7% faster. The Fujitsu is a much faster machine than either the Cray or the MasPar, with a theoretical peak performance of 2.5 Gflop/s. The timings shown in Table 3 were obtained without any attempt to optimize the code on the Fujitsu, and should be susceptible to improvement relatively easily.

Another parallel architecture was also included for comparison. The Connection Machine CM-2 has a SIMD architecture similar to the MasPar. The MasPar timing is 43% faster than the Connection Machine. FCT on the Connection Machine was written in a parallel implementation of C called C* (C star) by Oran et al. [5], while the code was written in Fortran 90 on the MasPar. The final two platforms presented for comparison are both single processor RISC workstations.

Further optimization of our code could be obtained by writing a fully two-dimensional version of LCPFCT, but this would require starting over from scratch. Due to our limited time on the MasPar we did not perform this step. Now that we have a fully optimized version of LCPFCT that uses direction splitting there are several possibilities for using the code to perform real-world simulations. We now have the ability to perform large scale direct simulations of both nonreacting and reacting flows. A combustion model is currently being added to the code so that we can simulate detonations and reacting flows in combustors. Our results only considered rectangular domains. To avoid the drastic performance degradation involved with simulating flow through more complex domains, techniques such as domain decomposition, and mapping of rectangular subdomains on the processor array topology would be required. In any event, finite-difference algorithms are arguably not the most suitable for complex geometries.

7 CONCLUSIONS

The MasPar's SIMD architecture is well suited for explicit finite-difference Euler solvers because the problem can be optimally mapped onto the machine topology, and the algorithm requires that the same operations be performed at all cells at all time steps. Optimization required that we rewrite the code to calculate the direction-split rows or columns simultaneously (in parallel), which can be done since adjacent rows or columns are independent of one another during such a computation. Apart from syntax modifications, further optimization was carried out by modifying the boundary condition calculations so that they were aligned with the main arrays. We have improved performance by a factor of 240 through the described optimizations.

In general, Fortran 90 parallel code is easier to write and work with, and shorter than the corresponding scalar code. The performance of FCT on the MasPar is slightly better than on CRAY Y-MP (1 CPU), and is also faster than on the Connection Machine CM-2. The MasPar has been relatively user friendly and easy to program, and the

Table 3. Benchmarks for the Two-Dimensional Blast Problem (Resolution: 128 × 128)

Computer Type	Temporal Performance (tstep/s)	Benchmark Performance (Mflop/s–64 bit)
MasPar MP-1 (8,192 K)	7.14	81
CM-2 (8,192K) [5]	5.00	57
CRAY Y-MP (1 CPU) [5]	6.67	76
Fujitsu VPX240	14.3	170
IBM RS6000/950	0.96	11
HP 9000/710	0.55	6.2

profiling and optimizing tools are effective. The results show that the MasPar is a suitable computer on which to carry out multidimensional FCT computations.

REFERENCES

[1] J. P. Boris and D. L. Book. "Solution of the continuity equation by the method of flux-corrected transport." *Methods Comput. Phys.* vol. 16. pp. 85–129. 1976.

[2] P. Collela and P. R. Woodward. "The piecewise parabolic method (PPM) for gas-dynamical simulations." *J. Comput. Phys.*, vol. 54. pp. 174–201. 1984.

[3] Digital Equipment Corporation. *DECmpp Parallel Fortran Reference Manual.* Maynard. MA: Digital Equipment Corporation. 1992.

[4] R. W. Hockney. "A framework for benchmark performance analysis." *Comput. Benchmarks*, pp. 65–76. 1993.

[5] E. S. Oran. J. P. Boris. and R. O. Whaley. "Exploring fluid dynamics on a connection machine." *Supercomput. Rev.*, 1990.

[6] E. S. Oran and J. P. Boris. *Numerical Simulation of Reactive Flow.* New York: Elsevier. 1987.

[7] E. S. Oran. J. P. Boris. T. R. Young. and J. M. Picone. "Numerical Simulation of Detonations in Hydrogen-Air and Methane-Air Mixtures." *Proceedings of the 18th Symposium (International) on Combustion.* Pittsburgh: The Combustion Institute, 1981. pp. 1641–1649.

[8] E. S. Oran, T. R. Young. and J. P. Boris, "Application of Time-Dependent Numerical Methods to the Description of Reactive Shocks," *Proceedings of the 17th Symposium (International) on Combustion.* Pittsburgh: The Combustion Institute, 1979. pp. 43–54.

[9] D. F. Snelling. "A philosophical perspective on performance measurement," *Comput. Benchmarks*, pp. 97–103. 1993.

[10] P. A. Thibault. F. Zhang. J. Penrose. and A. Sulmistras. "Numerical Modeling of Detonation Driven Hollow Projectiles." *Proceedings of the Second Annual Conference of the CFD Society of Canada.* Toronto: University of Toronto Press. 1994. pp. 395–402.

[11] B. van Leer. "Towards the ultimate conservative difference scheme. V. A second-order sequel to Godunov's method." *J. Comput. Phys.*, vol. 32. pp. 101–136. 1979.

[12] D. Williams. K. Grewal. C. Schuh. and L. Bauwens. "A Finite Difference CFD Code on a SIMD Architecture." *Proceedings SS'93 High Performance Computing: New Horizons*, 1993. pp. 531–536.

[13] F. Zhang. D. Tran. J. Penrose. C. Yee. and P. A. Thibault. "Numerical Studies of Detonation Propagation in Mixtures of Combustible Gases and Inert Dust." *Proceedings of the Second Annual Conference of the CFD Society of Canada. Toronto: University of Toronto Press, 1994, pp. 261–268.*