The Performance Realities of Massively Parallel Processors: A Case Study

Olaf M. Lubeck, Margaret L. Simmons, and Harvey J. Wasserman

Computer Research Group
Los Alamos National Laboratory
Los Alamos, New Mexico 87545

Abstract

We present the results of an architectural comparison of SIMD massive parallelism, as implemented in the Thinking Machines Corp. CM-2, and vector or concurrent-vector processing, as implemented in the Cray Research Inc. Y-MP/8. The comparison is based primarily upon three application codes taken from the LANL CM-2 workload. Tests were run by porting CM Fortran codes to the Y-MP, so that nearly the same level of optimization was obtained on both machines. The results for fully-configured systems, using measured data rather than scaled data from smaller configurations, show that the Y-MP/8 is faster than the 64k CM-2 for all three codes. A simple model that accounts for the relative characteristic computational speeds of the two machines, and reduction in overall CM-2 performance due to communication or SIMD conditional execution, accurately predicts the performance of two of the three codes. Other factors, such as memory bandwidth and compiler effects, are also discussed. Finally, the paper shows the similarity of the CM-2 and Y-MP programming models, and also comments on selected future massively parallel processor designs.

1.0 Introduction

It now seems that massively parallel processor (MPP) computer systems are viable architectures, as their use is becoming commonplace in all sectors of the computing community, including government, industry, and university settings. The peak speeds of these machines are among the highest available today, and many believe that they are the only solution to tomorrow’s supercomputing needs.

The Thinking Machines Corp. (TMC) Connection Machine-2, with 2,048 floating point processors and a theoretical peak speed of 28.6 GFLOPS, was the first commercially successful MPP. The CM-2 is typical of the MPP design philosophy that incorporates a large number of inexpensive processors joined by a sparsely-connected network whose transfer rate is low relative to peak computational speed. This strategy attempts to exploit the data locality exhibited by many supercomputer applications by optimizing interconnection throughput for those communication patterns that are most probable. Obviously, the best use of these machines is in cases where the ratio of computation to interprocessor communication is large, and indeed, published reports of such cases have demonstrated impressive performance [1,2]. Another favorable case, and one that is believed to be quite common in scientific computing, is where the dominant communication pattern is nearest-neighbor. Here, too, machines such as the CM-2 can perform well [3].

Before the debut of MPPs, traditional vector processors such as the Cray Research, Inc. (CRI) CRAY-1 were the most common architectural choice for high-performance computing. These machines are now available in relatively small multiprocessor configurations, and were typified by the CRAY Y-MP when the CM-2 first became available. The Y-MP has eight processors and is capable of achieving 2.6 GFLOPS - a clear mismatch in potential peak speed compared with that of the CM-2. The
design strategy of machines such as the Y-MP is very different from that of the MPPs. Here, a few high-performance processors are fully interconnected resulting in uniform memory latency and high interconnection bandwidth relative to the CPUs’ computational speed.

In this paper we address the performance issues arising from the contrasting design strategies of the CM-2 and Y-MP. Are the MPP advantages overwhelming? Are traditional supercomputers still competitive? What is the performance of the CM-2 compared with the Y-MP on full applications? We will show that the peak computational rate of a supercomputer is of little value in answering these questions. We will also show that despite the differing design strategies, the two machines implement similar computational models. Finally, we point out that these issues are crucial to the performance of the follow-on machines, TMC’s CM-5 and CRI’s Y-MP C90.

2.0 Architectures

Although both the Y-MP [4] and the CM-2 [5-7] have been described in great detail in other publications, we include here a brief comparison of their most salient features as they relate to our work.

CRAY Y-MP. The CRAY Y-MP used in this study is a 6.0-nanosecond general-purpose 64-bit vector computer composed of eight processors with 128 Megawords (1 Gigabyte) of globally-addressable memory. Vectors are processed by pipelining elements through independent floating-point functional units. Each processor has four simultaneously active, 64-bit wide ports to memory: two load ports, one store port, and one port that is used for I/O and instruction buffer loads. The overall eight-CPU load/store bandwidth is 32 Gbytes/s. Memory latency, meaning the interval between issue of a load instruction and arrival of the first word of data, is (assuming no conflicts) 19 clocks for contiguous or strided references and 24 clocks for gather/scatter references. The Y-MP includes specific hardware to aid in multiprocessing. These features include shared registers, used to pass data between the CPUs, and semaphore registers, which are used to synchronize code running on different CPUs.

CM-2. The CM-2 may be viewed as a machine in which 2K Weitek floating-point units, 2K network routers, and 64K single-bit processors are connected through an instruction sequencer to a front-end computer such as a Sun Workstation. User array data are stored in the memories associated with the 64K processors. The router chips, which are interconnected via a 12-dimensional hypercube, also have a set of fast communication paths for nearest-neighbor communications that is called NEWS. The front-end processor broadcasts instructions and addresses to the CM processors which carry out the operations on the array elements. Each floating-point CM-2 processor has 4M bytes of memory associated with it, for a total of 8G bytes. The LANL CM-2 is equipped with 64-bit floating-point units; however, data paths to and from the floating-point units are 32-bits wide, and for this reason, the peak rate for double-precision computation with data stored to and fetched from memory is 14 GFLOPS. Using previously published data [2], we estimate that the aggregate communication rate of the CM-2 for a nearest-neighbor exchange is 7.2 Gbytes/s. The internal memory-to-processor bandwidth is 28 Mbytes/s per floating-point node.

3.0 Discussion of Y-MP and CM-2 Programming Model

The CM-2 and CRAY Y-MP are frequently seen as being very different kinds of computers (one is referred to as “data parallel” and the other is referred to as “vector”) [7-10]. In fact, they employ quite similar computational models, although their memory models are different.

It is well known that the Y-MP is a vector multiprocessor system that can be used for general, asynchronous MIMD calculations. However, the Y-MP can also be used as a “concurrent vector” processor, in which different 64-element vectors from a single vectorized DO loop are distributed to multiple processors and
thus run concurrently. This computational mode can be invoked using the Cray autotasking software environment [11]. In theory, any code that has been written to take advantage of ordinary vector processing can be autotasked for concurrent vector mode. That is, the dependence analysis needed to show that a loop can be run in concurrent vector mode is exactly the same analysis required for vectorization.

The spreading of loop iterations over processors is exactly what the CM-2 does, even if the CM Fortran program omits explicit DO loop constructs and array indices. Thus, "data parallel" on the CM-2 and "vector" (or vector-concurrent) refer to the same underlying model of computation, one that is known as "Single Instruction Multiple Data" (SIMD). Both machines issue a single instruction that initiates many operations of the same kind on elements of a data structure.

What, then, is the difference between the CM-2 and the Y-MP? Primarily, it is a matter of the amount of hardware replication and the raw speeds at which each CPU runs. In the CM-2, SIMD computation is done by taking a vector and spreading it among 2K floating-point processors, each pipelining at the maximum rate of 2 results per 150 ns. In the Y-MP (in vector-concurrent mode), SIMD computation is done by spreading a vector among only eight processors each having a hardware vector length of 64, but pipelining at a maximum rate of 2 results per 6 ns. This leads to the simple view that the CM-2 is a vector processor with 2048 pipes and a hardware vector length of one.

We emphasize that these quantitative differences do not mean that the CM-2 presents a fundamentally new or different kind of computational model. The user of either of these machines employs a high-level computational model that is equivalent on the two machines, no matter how the vector of data is spread. The CM-2 programmer visualizes an environment in which each array element is associated with a single CM-2 virtual processor, thus leading to the concept that the machine acts on all array elements "at once." The Y-MP programmer, on the other hand, thinks of a single instruction that can be simultaneously applied to a block of data. Neither of these notions accurately represents the actual machine implementation. There is functional unit pipelining and multiple processor concurrency in both machines.

Of course, while the computational model of the two machines is equivalent, there may be differences in performance, cost, in flexibility within the SIMD model, or in ease of programming, and our results highlight some of these differences. Note also, that these factors are by no means independent of one another. For example, Zima [12], has observed that machines such as the CM-2 are much harder to program than vector computers, because the programmer must ensure that all data required for an operation are local to the processor carrying out that operation.

For completeness, we must note that our comments above refer to the Y-MP in concurrent-vector mode. The Y-MP also admits the more general MIMD programming model.

4.0 Methodology

In comparing the performance of different computer systems using benchmark codes, it is critical that the level of optimization be comparable on all machines being considered [13]. Although this is particularly important for vector/MPP comparisons, we believe that it has not always been addressed in previous studies [13]. An important reason for this disparity is that typically existing Cray codes are ported to the CM-2 with changes in algorithms or data structures during the adaptation [14-18, 21b]. However, the performance of the newer version of the code is never examined on the Y-MP.

Our hypothesis in this work was that any code optimized for the CM-2 would be nearly optimal for the concurrent-vector mode on the Cray, due to the equivalence of the programming models. The extensive "data parallelism" in a CM-2 code translates to long vector lengths that may be distributed among Y-MP processors with effective ‘amortization’ of concurrent processing overhead.

Therefore, our methodology was to begin with codes written for the CM-2 and port them
back to the Cray Y-MP, changing the CM Fortran to Fortran77. Little analysis was required to port the applications back to the Cray.

In all our tests 64-bit precision was used exclusively on both machines. The version 1.0 CM Fortran compiler in slice-wise mode and version 6102 CMIS were used on the CM-2 with a SUN 4/260 as front end. On the CRAY Y-MP, using CFT77 5.0, compiler directives were needed to achieve concurrent-vectorization of a few loops. The times reported in the tables below used the CM Fortran utility CM_TIMER_READ_ELAPSED. Times on the Cray were measured using the IRTC real-time clock function and the SECOND CPU time function. All CM-2 tests were done during dedicated time on the LANL Advanced Computing Laboratory CM-2. CRAY Y-MP tests were done using dedicated time or single-job stream runs on the Cray Research SN 1024 machine located in Eagan, Minnesota. All tests used Fortran or CM Fortran only with no hand-coded or micro-coded routines other than Fortran intrinsic functions.

4.1 The Benchmark Codes

The three codes used in this study are applications of importance to LANL. The codes are NEUT, PUEBLO and X3D. It should be noted that these applications were ported to or developed on the CM-2 because of the increased memory resources the CM-2 presented at the time. Problem sizes for the benchmark runs were chosen to resemble those typical of a production problem.

NEUT is an implementation of a Monte Carlo neutron transport algorithm. It was written specifically for the CM-2, and it is representative of a large class of computations in use at the Laboratory. NEUT models the transport of neutrons in fissionable materials using a 3-D mesh formed by the intersection of irregular tetrahedra. Two materials are simulated in the problem, with three isotopes per material. The problem size can be scaled with the number of source neutrons. An important characteristic of NEUT is that the mean free path of the neutrons is large relative to a typical cell size. Therefore, much of the time in the code is spent simply updating particle positions in the geometry. Not all particle transport problems have this characteristic.

The cross section data for neutrons are accessed using the CM Fortran utility routines that store the data on each floating-point node [19]. Neutrons in the simulation are gathered into batches and processed when the batch size becomes large. The CM Fortran code was developed primarily by Eldon Linnebur of the LANL Computational Physics Group and was translated for the Cray by us.

PUEBLO is a three-dimensional, finite-difference, time-explicit Lagrangian hydrodynamics code used to model point explosions in space. The CM-2 version represents a translation of part of the Los Alamos CAVEAT production code [20]. Development on the CM-2 was done by Eugene Symbalisty of LANL. The code uses a Gamma law equation of state and a first-order Riemann solver. In the CM Fortran version of the code, the geometry structure is implemented as 3-D subgrids for each floating point processor. The grid points lying on the 2-D subgrid faces are communicated to nearest-neighbor processors. In the Cray version, the three dimensions are merged explicitly into a single one-dimensional data structure, so that loop lengths are on the order of \( n^3 \), where \( n \) is either 64 or 128.

X3D is a three-dimensional Free Lagrange Hydrodynamics code that uses an unstructured mesh formed by intersection of irregular tetrahedra [21]. A tabular equation of state is used. Much of the time in the code is spent maintaining and updating the connectivity matrices associated with the geometry. The benchmark problem models the shock wave that results from implosion of a gas ball using about 9,000 mass points and 50,000 tetrahedral edges. Loop lengths in X3D range from about 8,000 to nearly 40,000. The code was developed by Harold Trease of the LANL Computational Physics Group.

5.0 RESULTS
5.1 Results for NEUT

Timing results for two problem sizes of NEUT on the CM-2 and Y-MP are listed in Table 1. Table 2 shows the significant portion of a routine-level profile of the code from the CM-2 and Y-MP. The single-processor Y-MP is comparable to the single quadrant CM-2 for both problem sizes of NEUT. Comparing the fully-configured systems, the Y-MP/8 is about 3.1 times faster than the 64k processor CM-2 on the smaller problem and is about 2.2 times faster on the larger problem. In the following three sections, we explain why the Y-MP exceeds the CM-2 performance on this code.

5.1.1 Peak CPU Speed Is Not Attainable

Compiler inefficiencies and memory latency/bandwidth are the major reasons that peak computational rates are unattainable on any machine [22]. However, the achievable rate expressed as a percentage of peak is widely different on the CM-2 and Y-MP. We measured this for a computation in NEUT that tracks a particle trajectory. The expression timed was

\[
R = (A-B) \ast (C-D) - (E-D) \ast (F-B),
\]

where R, A, B, C, D, E, and F are arrays, and no communication is involved. The measured CPU rates for this loop using 64-bit arithmetic are 1,560 MFLOPS for eight-processors of the Y-MP and 3,200 MFLOPS for a four-quadrant CM-2. Expressed as percentages of peak, these rates are 58% for the Y-MP and 11% for the CM-2, assuming a 28-GFLOP peak CM-2 rate. For fully-configured machines, the empirical results reveal that the CM-2's computational speed advantage over the Y-MP is about 2.1.

The relatively poor performance of the CM-2 relative to peak is attributed mainly to the lack of memory bandwidth per CM-2 node; this effect has been measured previously [23]. The immaturity of the compiler also leads to significant inefficiencies, although these can be overcome with time.

5.1.2 The Effect of WHERE's:

A Monte Carlo particle transport algorithm such as NEUT has two important characteristics, as reported in previous work [24]. The first is that the code may be viewed as a recursive n-ary decision tree in which a particle's progress is followed throughout a series of tests. The outcome of each test is a
function of material properties, particle parameters and random numbers. Effective SIMD-parallelization of such a simulation involves creating queues of particles that undergo the same process (i.e., particles that arrive at the same node in the binary decision tree), in order to keep vector lengths appreciably long. Our earlier paper [24] compared several methods of creating these queues, one of which was a "logical" method using vector masks. In the CM-2 version of NEUT, the vector mask approach is the only one possible, and the masks are formed using the CM Fortran "WHERE" statement. The Y-MP and CM-2 carry out masked operations such as these in two fundamentally different ways. In the CM-2 a mask array is used to deactivate processors for which the predicate is false, and the remaining processors carry out the operation(s) protected by the WHERE [25]. This results in a processor utilization that is equal to the truth ratio of the WHERE predicate. The time for an arithmetic operation will be independent of how many values are true. The Y-MP, however, can use its compressed-index scatter/gather instructions. The time for the operation will be proportional to the truth ratio, and processor utilization will be independent of the truth ratio.

We can model the effect of the WHEREs on CM-2 and Y-MP CPU times in the following way. The CPU time on the CM-2 is

$$t_{cm} = N / S_{cm},$$

where $N$ is the total number of flops done in NEUT and $S_{cm}$ is the characteristic computation rate of the CM-2. The corresponding time on the Cray is given by

$$t_{cray} = ( (1 - \beta) + \beta \times r ) N / S_{cray},$$

where $\beta$ is the fraction of the total time spent within WHERE constructs, and $r$ is the truth ratio of the WHERE statements averaged over the entire run. The ratio of CPU times is then

$$t_{cray} / t_{cm} = (S_{cm} / S_{cray}) \left[ (1-\beta) + \beta \times r \right],$$

and the bracketed term is a reduction factor by which the CM-2's computational rate is reduced due to conditional execution.

The parameters $\beta$ and $r$ may be obtained from TMC’s PRISM software [26] that allows a basic block examination of where CPU and communication times are spent in a CM-2 run. Using PRISM we have found that about 32% of the CPU time in NEUT is spent in code enclosed by WHERE constructs. The truth ratios for these blocks averaged over the entire benchmark run are small, averaging about 8%. Using these values, the reduction factor is .68 and the CM-2's peak characteristic speed advantage (see previous section) has been reduced to 1.4 (= 0.68 * 2.1).

5.1.3 The Effect of Communication:

A second important characteristic of Monte Carlo is data motion, by which we mean memory or interprocessor communication operations that rearrange elements in memory to gain locality [24]. In NEUT, this occurs principally because the geometry through which the neutrons are traced is general and arbitrary. (Generalized geometry is a characteristic of many important Monte Carlo codes [27].) For example, the location of a particle within one of the tetrahedra in NEUT involves gathering the $x$, $y$, and $z$ coordinates of the four tetrahedra vertices, and such gathering must occur each time the particle moves. On the CM-2 this involves use of the router communication network.

We have timed a gather expression individually on a single-processor Y-MP and single-quadrant CM-2, and have found that data transfer rate is about 30 times faster on the Y-MP. The PRISM data show that 46% of the total CM-2 time is spent in communication. From Amdahl’s Law, this means that the Y-MP gains another factor of two due to communications. Combining the effects from the previous two sections, we predict the overall Y-MP/8 execution time should be about 70% of the CM-2’s. This does not agree well with the observed ratio and further studies are underway to explain the discrepancy.
5.2 Results for PUEBLO

Timing results for two problem sizes of the PUEBLO benchmark are given in Table 3. A single quadrant of the CM-2 is about 40% faster than the single-processor Y-MP for the larger problem and the two machines are about the same on the smaller problem. When times for fully-configured systems are compared, the Y-MP is nearly 30% faster than the CM-2 regardless of problem size.

The sole form of communication in PUEBLO is nearest-neighbor, and PRISM reported that 57% of the CM-2 runtime is spent in NEWS communication. In CM Fortran, nearest-neighbor communication is carried out using either the CSHIFT function, which does a circular shift preserving the endpoints, or the EOSHIFT function, which does an "end-off" shift and zeros the beginning point. In the original code, PUEBLO included the EOSHIFT function. However, we later found that CSHIFT runs 40% faster than EOSHIFT and the code was optimized by replacing all EOSHIFTs with CSHIFTs.

In order to judge the impact of nearest-neighbor communications on the CM-2, we took a sample operation out of the routine LAGVEL, which is an important routine in PUEBLO. We timed this operation, which involved seven floating-point operations and one NEWS communication, first using CSHIFT, then using EOSHIFT, and finally without using any communications. The results are listed in Table 4, using a single CM-2 quadrant. The code timed on the CM-2 was

\[
D = \text{CSHIFT}(B, 1, -1)
\]

\[
T = A * A + B * B + C * C + D * D,
\]

and the code timed on the Y-MP was

\[
T(I) = A(I) * A(I) + B(I) * B(I) + C(I) * C(I) + B(I-1) * B(I-1),
\]

where all variables are arrays of dimension \((64)^3\) on the CM-2 and \((64*65*65)\) on the Y-MP. The data show that the overall rate for a sample computation using CSHIFT, the fastest means of communications tried, is still about a factor of two slower than the corresponding computation with no communication. Thus, even nearest-neighbor communication is costly on the CM-2. For the Y-MP, only one rate is given, because, assuming no bank or section conflicts, memory access rates are uniform across the address space. Scaling these rates to fully configured machines shows that the Y-MP should perform about 20% faster than the CM-2, in good agreement with the data in Table 3.

A NEWS shift of a CM-2 array involves copying all subgrid elements into temporary storage and an inter-processor communication of the boundary elements. Because PUEBLO is partitioned into 3-D subgrids on the CM-2, we have fit the measured CM-2 CSHIFT times as a

| Table 3. Execution Times (seconds) for the PUEBLO Benchmark on the CRAY Y-MP and TMC CM-2. |
|----------------------------------------|---------|---------|---------|
|                                        | 16k proc. | 32k proc. | 64k proc. |
| 64-cubed grid                          | 60.2     | 34.2     | 19.5     |
| 128-cubed grid                         | 385.6    | 206.6    | 109.0    |

| Table 4. Rates (MFLOPS) for a Sample Operation on the CRAY Y-MP (single-processor) and TMC CM-2. (single-quadrant) |
|------------------------------------------------------|-----------------|
| A. CM-2                                              | Rate (MFLOPS)   |
| CSHIFT                                              | 422             |
| EOSHIFT                                             | 184             |
| no shift                                            | 876             |
| B. CRAY Y-MP                                         |
| no shift                                            | 1 processor     |
|                                                     | 255             |
function of linear subgrid size to a cubic polynomial. The coefficient of the \( n^3 \) term represents the amount of time to perform the internal copy, and the \( n^2 \) term represents the amount of time needed to perform the actual interprocessor communication. The analysis shows three results: First, the copy rate is 10 times faster than the actual communication rate. Second, the difference in EOSHIFT times vs. CSHIFT times is solely in the copy term; the EOSHIFT copy rate is 10 times slower. Third, at large problem sizes the copying will dominate the SHIFT times and they will scale as the cube of the linear problem size.

5.3 Results for X3D

Timing results for X3D are given in Table 5. On this code, the Y-MP is consistently faster than the CM-2. The single-processor Y-MP is a factor of 3 faster than the 16k processor CM-2. When we compare results for the fully configured systems, the Y-MP is 7.4 times faster than the CM-2.

An important characteristic of Free-Lagrange Hydrodynamics, similar to the Monte Carlo application NEUT, is the gather of data using irregular interprocessor communications. Arbitrary and dynamic connectivity of the spatial geometry, which has been tessellated into tetrahedra, leads to the communication irregularity. This means that the predominant mode of communication in X3D is done through the general router communications network of the CM-2. Timing data show that X3D spends 93% of its execution time in general communications. Remembering that the one-processor Y-MP is 30 times faster than a quadrant of the CM-2 in irregular gathers, we can calculate (from Amdahl's law) that the Y-MP/8 should execute X3D 7.0 times \([1/(2.1*(1-.93))]\) faster than a full CM-2, in good agreement with the measured ratio of 7.4.

6.0 Summary and Conclusions

Responsible, careful evaluation of high performance machines is a necessary condition for the continued progress and future success of massively parallel architectures. These machines have enormous potential and there is certainly a critical need for orders of magnitude computational power over current supercomputers. However, high sustainable computing rates on real applications is the need, not unattainable peak rates or performance optimized for demonstration kernels using microcode. In our tests, the massively parallel machine is faster on none of our full applications when the fully-configured systems are compared. One of the most surprising results is that even in the case of a code using nearest-neighbor communications, the fully-configured CM-2 does not show a computational rate advantage over the Y-MP/8.

We have divided the reasons for the low efficiency of the CM-2 into four areas: compiler immaturity, lack of local memory bandwidth, inflexibility of computing conditionals in the SIMD model, and low interconnection throughput for arbitrary communications patterns. All of these reasons should be investigated in the context of future architectural strategies and the analysis of engineering parameters in these massively parallel machines.

Some fallacies into which researchers fall are pointed out in this paper. For example, in
none of our results do the times for the fully configured machines scale simply by the ratio of the number of processors. As aptly pointed out by Bailey [13], many papers on this subject measure performance on small configurations and erroneously assume perfect scalability to the fully-configured machine. Our data for NEUT do seem to support constant execution times for constant subgrid sizes.

At the heart of general purpose, easily-programmable supercomputers is the concept of flexible architecture. The keys to flexible architectures are high interconnection bandwidth, low-cost synchronization, and communication latency hiding. Massively parallel machines have not yet found the right engineering balance among these ingredients.

Acknowledgments

Many people read the manuscript and provided excellent comments and suggestions for which we are grateful. Some of these are: Ralph Brickner, Ingrid Bucher, Vance Faber, and Jim Moore, LANL; Maynard Brandt, Jeff Brooks, Ken Lord, and Jim Schartzmeier, CRI; Ann Hayes, Dave Forslund, Bob Malone, Stephen Pope, and Dave Rich, LANL ACL. Special thanks are due to Jeff Brooks for pointing out a critical error in the model and to John Cerutti of LANL for providing generous amounts of CM-2 time. We thank the ACL staff for providing us with several hours of dedicated time on the LANL ACL CM-2, Jeff Brooks and Richard Sandness (CRI) for helping to run the codes on the Y-MP, and numerous TMC personnel for helping to run the codes on the CM-2. Thanks are due to Eldon Linnebur, Harold Trease, and Eugene Symbalisty for many helpful conversations and to Kaki Kelly of LANL in helping to debug X3D on the Y-MP. Any errors in this paper should be attributed only to the co-authors.

References


