Measuring the Performance of a Power PC Cluster *

Emanouil I. Atanassov

Central Laboratory for Parallel Processing, Bulgarian Academy of Sciences, Sofia, Bulgaria
emanouil@copern.bas.bg

Abstract. In this work we present benchmarking results from our Linux cluster of four dual processor Power Macintosh computers with processors G4/450 MHz. These machines are used mainly for solving large scale problems in air pollution and Monte Carlo simulations.
We first present some numbers revealing the maximum performance of an individual machine. The results are from the well known LINPACK benchmark and an optimized Mandelbrot set computation.
The second set of benchmarking results covers the NAS Parallel Benchmark. These tests are written using MPI and Fortran 77 with some Fortran 90 constructs and are close to the real problems that we solve on the cluster. We also tested the performance of a free implementation of Open MP - the Omni Open MP compiler.
The last set of tests demonstrates the efficiency of the platform for parallel quasi-Monte Carlo computations, especially when the vector unit is used for generating the Sobol' sequences.

1 Description of the Software and Hardware Configuration

Our cluster consists of four dual processor Power Macintosh computers, connected with a BayStack 350 Switch. Each node has 512 MB RAM and 30GB hard disk space, and has two processors, Power PC G4 at 450 MHz clock frequency, with AltiVec technology. More details about the AltiVec technology can be found at the website http://www.altivec.org.

We decided to use entirely open source/free source software on these machines. The operating system on the cluster is GNU/Linux, kernel version 2.4.8, with SMP and AltiVec support enabled. We used the Yellowdog distribution version 2.0, but we compiled the kernel with the SMP and AltiVec support ourselves. The cluster was upgraded later to version 2.1, but this didn't result in any significant changes in the benchmarking results, shown here. Since the last upgrade the cluster had not been restarted.

* Supported by a project of the European Commission - BIS 21 under contract ICA1-CT-2000-70016 and by the Ministry of Education and Science of Bulgaria under contract NSF I-811/98.

P.M.A. Sloot et al. (Eds.): ICCS 2002, LNCS 2330, pp. 628–634, 2002.

We use in our work the GNU Compiler Collection (GCC), which has compilers for C, C++ and Fortran 77. When one wishes to use the AltiVec unit of the G4 processor, some special compiler support is needed. We use the Motorola patched gcc compiler, which introduces some non-standard extensions to the C language. In our experience use of this compiler and the AltiVec unit pays off only when a small part of the code is critical for the overall performance.

For our parallel programs we mainly use MPI, so we installed two versions of MPI - LAM MPI and MPICH. In the benchmarks and in our daily work we discovered that the LAM version had better results (version 6.5.4 was installed). It had better latency and throughput in almost all cases. That is why only results from this version are shown here. LAM has been compiled with support for OpenSSH, for better security, and with support for System V shared memory, since significant improvement was discovered in the communication between two processor from the same node when shared memory is used instead of TCP/IP.

We tested also the Omni Open MP compiler. More information on it can be found at `http://pdplab.trc.rwcp.or.jp/pdperf/Omni/`. For the Power PC architecture however, this compiler could be used only to generate parallel programs for the two processors on the same node, while for X86 processors and clusters the programs can operate on the whole cluster.

The cluster is used actively by researchers from the Bulgarian Academy of Sciences, as well as students from Sofia University.

2 Peak Performance Numbers

Power PC G4 is a RISC CPU, which means it has a large set of registers. 32 of them are vector registers, meaning they can store 4 integer or floating point 32 bit values. The CPU can perform one basic operation (like **load, store, xor, ...**) on these registers at once. A significant drawback of the AltiVec unit is that it can perform floating point operations only in single precision, so it can not be used in code that requires double precision floating point operations. In one operation the AltiVec unit deals with 128 bytes of data, so it can perform 4 Flops in 1 processor cycle. It can in some cases perform even more than 4 Flops per cycle, since it has a multiply-and-add operation.

A well known benchmarking test for processor speed is computing a Mandelbrot set. Consider the set coming from the relation $z_{n+1} \rightarrow z_n^4 + c$. Viktor Decyk, Dean Dauger, and Pieter Kokelaar from UCLA developed an AltiVec optimized algorithm, reaching almost 4 Flops per processor cycle. These computations are actively promoted as a benchmark of Power Macintosh computers performance by Apple. Since the source code of the AltiVec version of this code was not freely available, the author developed one such implementation. The code performs approximately 2 additions and 2 multiplications in one clock cycle, leading to an average of about 1533 MFlops.

This result suggests that the Linux OS and the SMP kernel do not impede the performance of the CPU in any way.

The next set of results is connected with the LINPACK benchmark ([6]), which measures essentially maximum speed of the LU algorithm. The author's experience is that the "best" options for compilation are "-O3 -funroll-all-loops", since the G4 CPU has a large set of registers, and all tests shown in the paper were done with these options. When one compiles the reference implementation of the test, he or she gets around 100 MFlops in double precision. However, using the ATLAS library ([1]), one can achieve more than 490MFlops for matrix multiplication and more than 330 MFlops for LU factorization in double precision.

Using the High Performance Linpack (see [5]), and trying various choices for the options at random, we easily found options which allowed us to achieve 300 MFlops rate per processor for the whole cluster, when all 8 processors were used (for matrices of 5000×5000 elements).

3 NAS Parallel Benchmark Results

This benchmark suite was developed by NASA with the aim to measure the parallel performance of given parallel architecture. The suite consists of eight tests, coming in four different sizes - W, A, B and C. For a detailed description of these tests see [2].

The test programs implement some important algorithms, and are usually refered by two letter abbreviations. We will write what these abbreviations mean: BT - Block Tridiagonal, CG - Conjugate Gradient, EP - Embarrassingly Parallel, FT - Fourier Transform, IS - Integer Sort, LU - LU Decomposition, MG - Multigrid, SP - Scalar Pentadiagonal.

The benchmark uses MPI and Fortran 77 with some Fortran 90 constructs and is widely accepted as a benchmark in the scientific community and by hardware and software vendors. See for instance [16],[14],[8],[10].

We remind that the rules for benchmarking do not allow changes in the source code, one can only look for the best compiler options. We used the GNU Fortran Compiler - g77, with options "-O3 -funroll-all-loops". We had problems with some tests, since the code uses some Fortran 90 constructs. For this reason the FT test didn't compile. The other tests produced correct results, when they were able to fit in memory.

In some cases the problems in their default sizes were too large to fit into the memory of a single machine, but can be solved if divided in two or four. All the results are given in Table 1,2,3, and 4. For some of the algorithms by Mops the actual MFlops rate achieved by the program is shown, while for the integer algorithms Mops means "millions of basic operations per second". The efficiency shown in the tables is obtained by dividing the Mops rate obtained by the Mops rate of 1 processor and the number of processors. Efficiency is not shown for size C problems, because the respective problems couldn't fit in RAM when only one processor was used.

We also give in Table 5 results from the Open MP version of the NAS Parallel Benchmark, obtained by using the free Omni Open MP compiler. The Mops rates

achieved using two processors on the same node are shown, and the efficiency is calculated by dividing the achieved Mops rate by the Mops rate of one processor from Tables 1,2 and 3. These results are not so good, compared with the results from the MPI version. In our institution we mostly use MPI, so we were not very concerned with the Open MP performance of the cluster.

Table 1. Results from the NAS Parallel benchmark, size W

Np	BT		CG		EP		IS		LU		MG		SP	
	Mops	Eff.	Mops	Eff.	Mops	Eff.	Mops	Eff.	Mops	Eff.	Mops	Eff.	Mops	Eff.
1	84.3	1	31.1	1	1.9	1	6.3	1	112.3	1	76.3	1	47.2	1
2			27.7	0.88	1.9	1	4.8	0.77	106.5	0.94	63.8	0.84		
4	84.0	0.99	18.9	0.60	1.9	1	2.3	0.37	100.7	0.89	42.4	0.55		
8			12.9	0.41	1.8	0.97	0.9	0.14	84.6	0.75	28.5	0.37	53.0	1.12

Table 2. Results from the NAS Parallel benchmark, size A

Np	BT		CG		EP		IS		LU		MG		SP	
	Mops	Eff.	Mops	Eff.	Mops	Eff.	Mops	Eff.	Mops	Eff.	Mops	Eff.	Mops	Eff.
1	80.1	1	31.4	1	1.9	1	6.2	1	95.4	1	64.1	1	40.4	1
2			29.0	0.92	1.9	0.99	4.6	0.74	97.6	1.02	58.3	0.91		
4	73.9	0.92	22.0	0.70	1.9	0.99	2.5	0.4	95.9	1.00	44.4	0.69		
8			16.1	0.51	1.9	0.99	1.4	0.22	87.2	0.91	37.6	0.59	42.3	1.05

Table 3. Results from the NAS Parallel benchmark, size B

Np	CG		EP		IS		LU		MG		SP	
	Mops	Eff.	Mops	Eff.	Mops	Eff.	Mops	Eff.	Mops	Eff.	Mops	Eff.
1	20.8	1	1.9	1			88.5	1	68.3	1	45.1	1
2	21.2	1.01	1.9	1			87.7	0.99	62.6	0.92		
4			1.9	1	1.9		87.8	0.99	57.9	0.85	46.2	1.02
8			1.9	0.99					12.5	0.18		

Looking at the numbers, we observe that in most cases the cluster shows acceptable performance. Perhaps only the IS (IS means integer sort) algorithm shows definitely poor performance. The performance drop seen in most algorithms when going from 4 to 8 processors is not unexpected, since the two processors on the same node share the same memory bus and the same network interface. Even when using both processors of the same node instead of two processors on different nodes bears a performance penalty of about 5-20%, when using LAM MPI. Comparing these numbers with the peak performance

Table 4. Results from the NAS Parallel benchmark, size C

Np	CG	EP	IS	LU	MG	SP
	Mops	Mops	Mops	Mops	Mops	Mops
2		1.9		85.4		
4	16.1	1.9		85.9		47.5
8	14.1		1.2	65.3		

Table 5. Results from the Open MP version of the NAS Parallel benchmark

Np	BT		EP		LU		MG		SP		FT	
	Mops	Eff.	Mops	Eff.	Mops	Eff.	Mops	Eff.	Mops	Eff.	Mops	Eff.
W	63.8	0.76	1.7	0.89	91.7	0.27	68.2	0.89	39.1	0.83	67.4	
A	58.0	0.72	1.7	0.89	68.2	0.71	50.8	0.79	35.3	0.83	45.7	
B			1.7	0.89	61.0	0.69	56.4	0.83	35.1	0.78		

numbers from the previous section, one can conclude that some of the algorithms presented here (for instance LU) are not very well optimized with respect to the cache of the particular machine. On the other hand, in real programs one rarely has time to introduce additional parameters and experiment with them, in order to fit more data in cache, so the results shown here can be considered as more "realistic", i.e. close to the performance one would expect for his own programs.

4 Parallel Quasi-Monte Carlo Algorithms

The results shown so far are mostly from standard benchmarks. They can be considered as a prediction for what one could expect from his own code. I was impressed by the huge gain in performance, when the AltiVec unit is used, and decided to develop a vector version of my algorithm for generating the Sobol' sequences.

These sequences were introduced by Sobol in [12] (see also [13]). They are widely used in quasi-Monte Carlo computations, because of the uniformity of their distribution. In Financial Mathematics they are used for computing special kind of multi-dimensional integrals, with the dimension reaching often 360 or more. The reader might look at the paper of Paskov and Traub [15] for more details .

Our algorithm for generation of the Sobol sequences, which will be described elsewhere, satisfies the "guidelines" for using the vector unit - a small part of the code takes most of the CPU time. When one needs less than 2^{22} points, there is no loss of precision if only single precision numbers are generated. Generating more than 2^{22} points, while still gaining from the use of the AltiVec unit, is also possible.

In Table 6 one can see timing results from generating 1 000 000 points in 360 dimensions. These results are compared with the results from a basic version of the algorithm, using ordinary operations. In the programs GB and GV we only made sure the generation of the terms of the sequence actually happens, and

is not optimized out by the compiler. In GB we used regular operations and in GV some AltiVec operations were used. In programs SB and SV we show timing results from generating and summing all the terms of the sequence. In SB only regular instructions are used, and in GV the AltiVec instructions are used for both generation and summing the terms, with the summing being done with just the most simple vector algorithm.

Table 6. Timing results for generating and summing 1 million terms of the Sobol' sequences in 360 dimensions

	1			2			4			8		
	Time	Mnps	Eff.	Time	Mnps	Eff.	Time	Mnps	Eff.	Time	Mnps	Eff.
GB	7.3	49.0	1	3.7	96.3	0.98	1.9	189.8	0.97	1.0	363.1	0.92
GV	2.3	159.4	1	1.2	309.9	0.97	0.6	589.9	0.93	0.3	1034.8	0.81
SB	10.8	33.4	1	5.5	65.8	0.99	2.8	130.5	0.98	1.4	252.8	0.95
SV	4.3	83.3	1	2.2	163.6	0.98	1.1	318.2	0.95	0.6	586.6	0.88

By Mnps we denote the number of millions of elements of the sequence generated in one second. We show the total time and the Mnps rate. For each term of the sequence 360 coordinates are needed, so there are 360×10^6 elements to be generated.

Observe the good speed-up obtained by using all the 8 processors of the cluster, and also the fact that SV takes noticeably more time than GV, so for a "real" quasi-Monte Carlo algorithm the generation of the sequence will be performed in a small fraction of the total time. Another observation is that summing is perhaps the simplest operation on the generated sequence that can be performed. Still the difference in time between GB (generation only) and SB (generation and summation), is much more than the time spent for the AltiVec generation program GV.

While good speed-up is usually associated with the Monte Carlo algorithms, the above results show that quasi-Monte Carlo algorithms can also be very efficiently parallelized and vectorized, when the Sobol' sequences are used.

5 Conclusions

The various benchmarking results shown here demonstrate the viability of the Linux cluster built from Power Macintosh machines as a platform for scientific computing. Its vector capabilities can be utilized via standard libraries, specifically optimized for the AltiVec architecture, or by writing hand-tuned subroutines, when the time permits so.

References

1. Automatically Tuned Linear Algebra Software (ATLAS) - `http://math-atlas.sourceforge.net/`

2. Bailey, D., Barton, J., Lasinski, T., Simon, H., eds.: The NAS Parallel Benchmarks. Technical Report RNR-91-02, NASA Ames Research Center, Moffett Field, CA 94035, January 1991.
3. Bailey, D., Harris, T., Saphir,W., van der Wijngaart, R., Woo, A., Yarrow, M.: The NAS Parallel Benchmarks 2.0 Report NAS-95-020, December, 1995
4. GNU Compiler Collection -http://gcc.gnu.org/
5. High Performance Linpack Benchmark - http://www.netlib.org/benchmark/hpl/
6. Linpack Benchmark - http://www.netlib.org/linpack
7. Linux Kernel Archive - http://www.kernel.org/
8. MPI Performance on Coral - http://www.icase.edu/~josip/MPIonCoral.html
9. NAS Parallel Benchmark - http://www.nas.nasa.gov/NAS/NPB/
10. NAS Parallel Benchmarks on a Scali system - http://www.scali.com/performance/nas.html
11. Omni Open MP Compiler - http://pdplab.trc.rwcp.or.jp/pdperf/Omni/
12. Sobol',I.M.: On the distribution of point in a cube and the approximate evaluation of integrals, USSR Computational Mathematics and Mathematical Physics, 7,86–112, 1967
13. Sobol', I.M.: Quadrature formulae for functions of several variables satisfying general Lipshuz condition, USSR Computational Mathematics and Mathematical Physics, 29,935–941, 1989
14. Origin 2000 on NAS Parallel Benchmark - http://www.nas.nasa.gov/~faulkner/o2k_npb_benchmarks.html
15. Paskov,S. and Traub,J.: Faster Valuation of Financial Derivatives, Journal of Portfolio Management, Vol. 22:1, Fall, 1995, 113–120.
16. Turney, R.D.: Comparison of Origin 2000 and Origin 3000 Using NAS Parallel Benchmarks. NAS Technical Report 01-003
17. Yellowdog Linux - http://www.yellowdog.com/