Porting Scientific Applications to the NYU Ultracomputer

by

Jerome Chiabaut
Vladimir Fleyshgakker
Anne Greenbaum

Ultracomputer Research Laboratory
Courant Institute of Mathematical Sciences
715 Broadway, 10th Floor
New York, NY 10003

Ultracomputer Note #144
July, 1988

Current Address for J. Chiabaut: B & R, 9x-41, P.O. Box 3511 Station C, Ottowa, Ontario, Canada K144H7

This work was supported under the Applied Mathematical Sciences Program of the US Department of Energy under contract DE-FG02-88ER-25052.

ABSTRACT

We report our experience in porting six different scientific codes to the Ultracomputer prototype and present promising results on speedups that can be achieved on shared memory MIMD architectures.

1. Introduction We ported six scientific programs to the Ultracomputer prototype: SIMPLE, GFHE4, GCMWF, SCALGAM, NT3D, and PESKIN. The following table summarizes the characteristics of each program and the kind of input that was used for the timings.

code	number of source lines	Type of Computation	problem size	Restructuring for Parallelism
SIMPLE	1408	fin. diff. in space; explicit/ADI time diff.	128x128	tridiagonal solve -> cyclic reduction
GCMWF	1664	fin. diff. in space; explicit time diff.; Fourier smoothing.	60x40x9	reorder nested DO loops
SCALGAM	745	Monte Carlo	4000 particles 50x16	none required
GFHE4	1882	Monte Carlo	4000 particles	none required
NT3D	1000	explicit time diff.; independent angles	8-24 angles	none required
PESKIN	2162	fin. diff. in space; ADI time diff.; FFT.	16x16 velocities; 64 positions	tridiagonal solve -> cyclic reduction

SIMPLE is a commonly used benchmark designed at Lawrence Livermore National Laboratory (see [CHR]) for testing high speed architectures. It is a simple example of a Lagrangian hydrodynamics calculation. The Green function program, GFHE4, is a production program developed at New York University that typically runs on Cray computers and solves the Schroedinger equation for the ground state of Helium-4. The three-dimensional weather forecasting code, GCMWF, is a simplified version of a production code from NASA. SCALGAM is a Monte Carlo gamma ray transport benchmark available from Los Alamos National Laboratory that has already been ported to Cray and Alliant parallel computers. NT3D is a 3-D neutron transport code under development at Lawrence Livermore National Laboratory, to be used as a benchmark for parallel computers. It has also been implemented on the Alliant FX-8 at Livermore (see [DGF]). PESKIN (see [PES]) has been developed at New York University for simulation of

incompressible flow of a fluid coupled to an elastic membrane (blood flow through the heart).

Several of the codes -- SIMPLE, GCMWF, and PESKIN -- were previously run under the Washcloth simulator (see [AG], [KR], [LUB]). Although some of the original codes required significant restructuring or algorithmic changes to take advantage of parallelism, these transformations were carried out before performing the simulator runs ([KR], [LUB], [PES]). To demonstrate the flexibility of the Ultracomputer architecture and software, we purposely made no further attempt to restructure the programs in order to achieve better speedups (see [GP] for an example of such a major endeavor). On the contrary, we tried to keep the code as readable and straightforward as possible and we limited ourselves to parallelizing loops and adding a few procedures specific to the parallel environment (for example, to implement a parallel stack, or find the maximum in parallel).

Loops were parallelized using the DOALL construct available through a FORTRAN preprocessor on the Ultracomputer prototype [WB2]. Processes are prespawned at the beginning and assigned work whenever a DOALL loop is encountered. Indices of the loop are then assigned dynamically to the processors. DOALL loops can be nested and can contain subroutine calls. We found this construct to be especially useful for parallelizing codes with a minimum of changes to the original source. The small increase in the size of the source programs and the good speedups attest to the success of this approach.

Timings reported in this paper apply only to the main body of the codes. Many of the programs have a short setup phase which, in realistic problems, contributes little to the total run time. Since some of the test runs used fewer time steps than would be required in practice, however, the setup represented a larger fraction of the total time of the test run. This phase can also be parallelized, sometimes more and sometimes less efficiently than the remainder of the code, but to avoid misleading results, it was excluded from the timings. All times are given in *seconds* and represent elapsed time from the beginning of the main computation to the end. Timings were performed in single-user mode on the Ultracomputer prototype.

2. Hydrodynamics Code (SIMPLE)

SIMPLE is a finite difference code that uses the Alternating Direction Implicit (ADI) method of time differencing. This results in a tridiagonal matrix to be solved at each time step. The version that we implemented used a cyclic reduction method to solve this tridiagonal system. This requires somewhat more operations than the best serial algorithm but is more amenable to parallelization. The following table shows the time results for SIMPLE with a mesh size of 128x128 in double and single precision.

SIMPLE	serial	1pe	2pes	3pes	4pes	5pes	6pes	7pes	8pes
time dble	2582	2691	1349	910	695	560	471	414	377
speedup dble	-	0.96	1.91	2.84	3.72	4.61	5.48	6.24	6.85
time sngl	2210	2294	1157	779	597	480	407	356	324
speedup sngl	-	0.96	1.91	2.84	3.70	4.60	5.43	6.21	6.82

The speedups we observe are quite typical of this kind of application and should be compared to the ones obtained on the EPEX simulator (see[PT]).

3. Green Function Helium 4 (GFHE4)

GFHE4 is a Monte Carlo code which solves for the ground state the Schroedinger equation with **n** equal mass particles in 3-dimensions for helium-4 atoms. The wave function is a Jastrow plus triplet factor. The Green function helium-4 code has been parallelized. The following table shows the results of timings made on the single precision and double precision versions for 50 starting configurations of 16 particles each:

GFHE4	serial	1pe	2pes	3pes	4pes	5pes	6pes	7pes	8pes
time dble	12779	12784	6461	4397	3371	2786	2398	2161	1958
speedup dble	-	0.99	1.97	2.90	3.79	4.58	5.32	5.91	6.52
time sngl	9982	9998	5066	3463	2679	2207	1939	1749	1587
speedup sngl	-	0.99	1.97	2.88	3.72	4.52	5.14	5.70	6.28

The speedups are somewhat low for a program that is fully parallel. This is explained by the fact that the program precomputes tables that are subsequently used in read-only mode. These should naturally be shared cached or private. Due to memory limitations of the current Ultracomputer prototype, the tables must be shared by the different tasks. The attribute cacheable was not supported by the compiler at the time of these runs, and so each reference to the tables forces an access to global memory.

4. Three Dimensional Weather Code (GCMWF)

The 3-d weather code has been ported to the Ultracomputer prototype. The following table shows timing results for the double precision version of this code on a 3-d grid encircling the globe with 60 points in longitude, 40 in latitude and 9 regions of fixed altitude:

GCMWF	serial	1pe	2pes	3pes	4pes	5pes	6pes	7pes	8pes
time	2552	2686	1351	915	692	558	475	416	367
speedup	-	0.95	1.89	2.79	3.69	4.57	5.37	6.13	6.95

The code uses a finite difference scheme to generate solutions to a global circulation model of the atmosphere. The grid is evenly spaced in latitude and longitude, and hence the spatial distance between adjacent grid points becomes very small near the pole. A Fourier smoothing of the solution near the poles is used to eliminate high frequency harmonics generated by a time step too large for the spatial discretization in that region. In this initial port, we used a complex Fourier transform. Better speed could be achieved by using a real Fourier transform.

5. Gamma Ray Transport Code (SCALGAM) SCALGAM transports photons through a carbon cylinder with an iron plug and lead jacket. The code has been parallelized and gives the same results as the original version on the Ultracomputer prototype. On a 4000 particles problem, we observed a speedup of 7.1 with 8 processors. Due to a bug in the original program, the code gave non-reproducible results on a larger and more typical problem (200,000 particles). The problem turned out to be very difficult to track down although a more sophisticated compiler could have flagged the use of a non-initialized variable. Although the problem was not related to parallelism, it affected only the parallel code and indicated the need for better parallel debugging tools!

Following are timing results for the 4000 particles problem:

SCALGAM	serial	1pe	2pes	3pes	4pes	5pes	6pes	7pes	8pes
time	2577	2573	1301	879	667	541	458	401	359
speedup	-	1.00	1.98	2.93	3.86	4.76	5.62	6.42	7.17

Speedup curves for SIMPLE, GFHE4, GCMWF, and SCALGAM are plotted in Fig. 1. As can be seen from the figure, the speedups are all nearly linear with the number of processors.

6. Neutron Transport Code (NT3D)

The 3-D neutronics code NT3D uses a Petrov-Galerkin finite element method to solve the equations of neutron transport. Different angles in the equations are uncoupled during a single time step, so the code can be parallelized with tasks of large granularity. The following tables show speedups obtained on the Ultracomputer prototype using 8 or 24 angles, with 10x10 spatial zones:

8 angles	serial	1pe	2pes	3pes	4pes	5pes	6pes	7pes	8pes
time	123.4	123.4	62.9	47.9	33.0	32.4	32.5	32.6	18.6
speedup	-	1.00	1.96	2.58	3.73	3.81	3.80	3.78	6.65

24 angles	serial	1pe	2pes	3pes	4pes	5pes	6pes	7pes	8pes
time	351.6	351.6	177.5	119.8	91.1	76.2	62.5	62.6	49.0
speedup	-	1.00	1.98	2.93	3.86	4.61	5.62	5.62	7.17

Note that, because of the large granularity of tasks, load balancing becomes an issue when the number of angles is not a multiple of the number of processors. With 8 angles, for instance, there is no gain in going from 4 to 7 processors. Smaller grain parallelism is also possible in this code but was not exploited in this implementation.

This code was also parallelized on the Alliant FX-8 at Lawrence Livermore National Laboratory. The following timings were obtained by M. Dorr [DGF] on the Alliant:

8 angles	Alliant	1pe	2pes	3pes	4pes	5pes	6pes	7pes	8pes
time	-	.402	.203	.154	.106	.106	.105	.105	.071
speedup	-	1.00	1.98	2.61	3.79	3.79	3.83	3.83	5.66

24 angles	Alliant	1pe	2pes	3pes	4pes	5pes	6pes	7pes	8pes
time	-	1.206	.607	.408	.311	.264	.218	.214	.186
speedup	-	1.00	1.99	2.96	3.88	4.57	5.53	5.64	6.48

Although parallel efficiency is somewhat lower on the Alliant than on the Ultracomputer prototype, this is probably due to the fact that, with vectorization, individual processors on the Alliant are much faster than those on the ultracomputer. Thus, for the same size problem, the granularity of tasks on the Alliant is much smaller. The following table shows timings on the Alliant for a 20x20 spatial grid with 24 angles:

24 angles	20x20 zones	1pe	2pes	3pes	4pes	5pes	6pes	7pes	8pes
time	-	8.88	4.45	2.99	2.26	1.90	1.54	1.54	1.20
speedup	-	1.00	1.99	2.97	3.93	4.67	5.77	5.77	7.40

The larger problem size (20x20 zones) had little effect on the speedups obtained on the Ultracomputer prototype. Yet the 20x20 problem with 24 angles gave a speedup factor of 7.4 on the Alliant. This is better parallel performance than that obtained by any of these codes on the Ultracomputer prototype. This would seem to indicate that bus saturation is less of a problem on the Alliant.

It should also be pointed out that parallelization on the Alliant was less straightforward than on ultra. The difficulty was in forming a global sum of fluxes. On ultra, this was accomplished using the

floating point fetch-and-add routine (ffaa). It would be more efficient if this were a hardware instruction, as fixed point fetch-and-add is intended to be on the "real" ultracomputer. While such a routine could also be written for the Alliant, it would not be efficient without hardware support, and so a more complicated strategy, requiring extra storage, was used to form the global sum. The first implementation had processors waiting for the sum variable to be unlocked before adding in their fluxes and continuing with their computation. This was quite inefficient. The speedups given above were achieved by having processors continue with their computations, while waiting for the global sum variable to be unlocked.

Results on the Ultracomputer prototype and the Alliant FX-8 are plotted in Figs. 2a and 2b, respectively.

7. Simulation of Blood Flow through the Heart (PESKIN)

The PESKIN program uses a finite difference approximation in space, with ADI time differencing, and with FFT's for smoothing. The following table shows timing and speedup using 8 processors, for various problem sizes:

velocities	membrane positions	serial time	parallel time (8 pes)	speedup
16x16	32	855	130	6.58
16x16	64	1970	300	6.57
32x32	64	11520	1680	6.86

The following table shows comparisons of the speedup using from 1 to 8 processors on the 16x16 velocities, 64 positions problem with the predicted speedups from the Washcloth simulator (see [LUB]).

PESKIN	1pe	2pes	3pes	4pes	5pes	6pes	7pes	8pes
time	1970	1000	700	520	440	380	350	300
speedup	-	1.97	2.81	3.79	4.48	5.18	5.63	6.57
predicted speedup	-	1.99	2.86	3.95	-	-	6.09	7.77
difference (%)	-	1	1.3	4	-	-	7.5	15.4

While actual and predicted speedups are in close agreement using 1 to 4 processors, they differ appreciably using 7 or 8 processors. The reason for this appears to be bus traffic. The simulator models a paracomputer [AG], while bus traffic on the Ultracomputer prototype always seems to limit speedups to a maximum of about 7.2 using 8 processors.

Actual and predicted speedups for the PESKIN code are plotted in Fig. 3.

8. Conclusions

The applications ported to the NYU Ultracomputer prototype have been shown to parallelize efficiently on a small number of processors. The problem sizes used in these tests were generally small compared to the size problems that scientists actually want to run. It is believed that with larger problem sizes these results will scale to larger numbers of processors and the codes will also exhibit good performance on more highly parallel systems, provided the interconnection network can properly accommodate the additional memory traffic.

9. References

[AG] A.Gottlieb, *WASHCLOTH - The Logical Successor to Soapsuds*, Technical Report #29, Computer Science Department, NYU,December 1980.

[CHR]
W.P.Crowley, C.P.Hendrickson, and T.E.Rudy, *The Simple Code*, Lawrence Livermore Laboratory, February 1978.

[DGF]
M. Dorr, A. Greenbaum, and J. Ferguson, *Solving the Neutron Transport Equation in 3-D with a Multitasked Finite Element Method*, LLNL report, to appear.

[DR] F.Darema-Rogers, D.A.George, A.Norton, G.F.Pfister, *VM/EPEX - A VM Based Environment for Parallel Execution*, IBM Research RC11381, 1985.

[EH] E.M.A.Hussein, *Modeling and Design of a Neutron Scatterometer for Void-Fraction Measurement*, submitted for publication.

[GP] D.Gannon and J.Panetta, *Restructuring Simple for the Chip Architecture*, Parallel Computing 3 (1986), pp 305-326.

[HK] H.Katzan Jr., *FORTRAN 77*, Computer Science Series, Van Nostran Reinhold Company, New York, 1978.

[KR] D.Korn and N.Rushfield, *Washcloth Simulations of Three-Dimensional Weather Forecasting Codes*, Ultracomputer Note #55, May 1983.

[LUB]
B.D.Lubachevsky, *Parallelizing an Algorithm of Charles S. Peskin for Describing Incompressible Flow of a Fluid Coupled to an Elastic Membrane*, Ultracomputer Note #42, May 1982.

[PES]
C.S.Peskin, *Numerical Analysis of Blood Flow in the Heart*, Journ. Comput. Physics, Vol.25, pp 220-252, 1977.

[PT] P.J.Teller, *Application Survey Report #2, SIMPLE*, EPEX report.

[WB1]
W.Berke, *Pdb - An Ultracomputer Debugger for Parallel Programs*, User's Guide, Ultracomputer

Documentation note #3, January 1987.

[WB2]

W.Berke, *A Structured environment for Parallel FORTRAN*, Ultracomputer System Software note, August 1987.