Optimization of LESlie3D for the Cray X1 Architecture

Sam Cable and Thomas Oppe, ERDC MSRC

ABSTRACT: *LESlie3D is a CFD code that solves the fully compressible Filtered Navier-Stokes equations, the energy equation, and the chemical species equations using an explicit finite-volume approach. A version of this code for structured meshes was included in the DoD High Performance Computing Modernization Program's application benchmark suite used for performance evaluation of HPC platforms. Cray applications specialists optimized this version for the X1 architecture. The techniques used for the optimization are presented, and a comparison of the original and optimized versions of LESlie3D is made using hardware counter data obtained from Cray PAT on the X1 and from PAPI on an IBM Power4.*

1. Introduction

ERDC MSRC

The U.S. Army Engineer Research and Development Center (ERDC) Major Shared Resource Center (MSRC) in Vicksburg, MS is one of four centers in the nation operated by the Department of Defense High Performance Computing Modernization Program (HPCMP) to supply HPC resources to scientists and researchers involved in computational projects funded by the DoD. Since its inception, the goal of the HPCMP has been to supply the nation's armed services with the most powerful computational resources available. The center at Vicksburg operates a 1904-pe Cray T3E, a 512-pe HP AlphaServer SC40, a 512-pe HP AlphaServer SC45, three 512-pe Origin 3900 platforms, and a 64-MSP Cray X1. These HPC resources support a number of on-site and off-site researchers in various computational fields designated as Computational Technology Areas (CTA's). The various CTA's are:

CWO	Climate/Weather/Ocean Modeling
EQM	Environmental Quality Modeling
CE	Computational Environment
SIP	Signal/Image Processing
FMS	Forces Modeling and Simulation/C4I
CFD	Computational Fluid Dynamics
CSM	Computational Structural Mechanics
CCM	Computational Chemistry and Materials Science
CEA	Computational Electromagnetics and Acoustics
CEN	Computational Electronics and Nanoelectronics

Technology Insertion

In support of the HPCMP mission, a program for the evaluation of HPC hardware has been instituted that involves the running of application and synthetic benchmark programs. The synthetic benchmarks are designed to stress the various software and hardware subsystems in a parallel computer, such as those involved with the processor, cache, memory, I/O, and communications network. The applications benchmarks, on the other hand, are chosen to be typical of the kinds of computation done in the various CTAs and representative of the workload characterizing each MSRC. For each fiscal year since 2001, the Computational Science & Engineering group at ERDC has been tasked with the responsibilities of assembling the application benchmark suite and checking the HPC vendors' responses for completeness and accuracy. The vendor responses are then analyzed by the HPCMP for use in HPC procurement decisions.

LESlie3D

LESlie3D (for "L"arge-"E"ddy "S"imulation "Li"near "E"ddy Model in "3D") was chosen for the TI-02 and TI-03 application benchmark suites as a typical CFD code. LESlie3D is a three-dimensional, Navier-Stokes solver for turbulent reacting flows. It was originally written by Professor Suresh Menon of Georgia Tech's Computational Combustion Laboratory but has undergone many changes in time. It employs a finite-volume predictor-corrector scheme that is fourth-order accurate in space and second order accurate (explicit) in time. The code solves the conservation of mass, momentum, energy and species equations using a conservative formulation. The code is designed to operate in either a direct numerical simulation (DNS) approach or a large-eddy simulation (LES) approach. In the LES approach (which was used for the benchmark), all scales larger than the grid are resolved using the finite-volume scheme and only scales smaller than the grid are modeled using a subgrid model. In LESlie3D, a one-equation model for the subgrid kinetic energy is used to close the unresolved terms in the LES equations. It is also capable of employing a

localized dynamic evaluation of the coefficients in the subgrid closure.

LESlie3d has been used to solve problems of various complexities, from mixing layers to gas turbine combustors. There exist solvers for both cylindrical and square configurations. The algorithm is suitable for obtaining high-resolution simulations of free shear flows such as those encountered in mixing layers, isotropic turbulence, and flame propagation. Curvilinear coordinate transformations are not performed so a uniform grid is required. The code allows a choice for the order of spatial accuracy (2^{nd} or 4^{th}), the number of chemical species used (none, flame model, or multiple species), the LES turbulence model (standard or k-equation model), and inviscid or viscous flow. The benchmark version of the code was set up for fourth-order spatial accuracy, two chemical species, viscous flow, and the standard LES turbulence model.

The code has been used for various reacting and non-reacting flows such as swirl-stabilized combustion in a gas turbine, spatial compressible mixing layers, combustion instability in ramjet engines, etc. However, the version used for the benchmark solves a simpler flow of a temporal mixing layer formed between two parallel plates that are moving in opposite directions. A uniform grid distribution in all three directions is used to resolve the flow in a cubic volume. Two scalar species are also simulated to mimic fuel and air mixing in the mixing layer. Thus, a total of eight conservation equations are solved in the benchmark version of the code.

LESlie3d is written entirely in Fortran77 and is parallelized using the Message Passing Interface (MPI). For the fourth-order scheme, two layers of ghost cells are required in each processor at every time step.

2. Description of systems - hardware

The HPC platforms used in this study are the 64-MSP Cray X1 named "diamond" at ERDC MSRC in Vicksburg, MS, and the IBM POWER4 (p690) named "marcellus" at NAVO MSRC.

The ERDC X1 has 64 Multi-Stream Processors (MSP), with four reserved for the operating system, and 60 MSP's available for application processing. Each X1 MSP consists of four Single-Stream Processors (SSP's) with each SSP consisting of two vector functional units running at 800 MHz and a single scalar functional unit running at 400 MHz. On diamond, then, an MPI code compiled in MSP mode can use up to 60 MPI processes and, if compiled in SSP mode, can use up to 240 MPI processes.

Addition and multiplication vector instructions can be chained in each vector functional unit, so the peak speed of the MSP is 800*4*2*2 = 12.8 GigaFLOPS in 64-bit precision. The vector functional units can process 32-bit operands at twice the rate of 64-bit operands, so that the peak speed in 32-bit precision is 25.6 GigaFLOPS. Since the "vector speed" is so much greater than the "scalar speed" on the X1, vectorizing a code as much as possible is crucial to maximizing X1 performance. Since "multi-streaming," or the splitting of vector operations between the four SSP's in an MSP, is done in hardware, it is also desirable to multi-stream the loops of a code as much as possible. If multi-streaming is not possible or efficient for a code, the code can be compiled in SSP mode so that vector instructions are executed in a single SSP rather than spread across the four SSP's of an MSP. MPI codes that are compiled in SSP mode can use four times as many processes than if compiled in MSP mode, but more synchronizations will be performed through the MPI software. Thus, due to hardware synchronizations, a code that multi-streams well may run faster in MSP mode than in SSP mode using the same number of vector functional units.

NAVO's POWER4 has 1408 IBM POWER4 RS6000 PEs, 1328 of which are available for application processing. Each PE has a clock speed of 1.3GHz. The two floating-point units on each PE together provide a peak processor speed of 5.2 GigaFLOPS. Each PE has one gigabyte of main memory. The PEs are connected via the fast IBM Federation switches. Most of the PEs are grouped in nodes of eight PEs, but two nodes of 32 PEs are available. The POWER4 PEs do not have any equivalent to the X1's multi-streaming/single-streaming levels. They cannot take as much advantage of vectorization, but neither is vectorizing a code crucial to procuring their peak performance.

3. Description of LESlie3D benchmark

The benchmark problem was posed on a 3-D cube domain discretized uniformly in each dimension using NX grid lines. Upon this domain grid, a 3-D processor grid configuration is imposed, say NPX by NPY by NPZ, where NX must be evenly divisible by NPX, NPY, and NPZ. The number of MPI processes is then the product of the processor grid dimensions.

Parts of the LESlie3D benchmark run do not scale well either in terms of memory use or run time. At the beginning of the run, MPI process 0 reads a large binary file named "mixing.data" and broadcasts it in the form of large arrays to all processes. Each process then discards all values in these global arrays that do not belong to its subdomain. Also, a large file containing the global flow field is assembled using the MPI_REDUCE operation and printed by MPI process 0. During the time-stepping phase, nearest neighbor communications in the three dimensions is done at each time step, and trace files are written by MPI process 0 at every 100^{th} time step. The global data written to the trace files is computed using MPI_REDUCE operations.

Optimization for the Cray X1

Cray application analysts optimized LESlie3D for the X1 platform for the HPCMP TI-03 applications benchmark. The structure of many of the computationally intensive loops in LESlie3D is that of a triply nested loop:

```
      do k = k1,k2
        do j = j1,j2
          do i = i1,i2
            (work)
          end do
        end do
      end do
```

In three routines (*emsi*, *emsj*, and *emsk*), the *k* and *j* loops traverse the entire subroutine and there are many instances of the innermost *i* loop. Without compiler directives, only the innermost loops were being vectorized and multi-streamed. Cray added a Cray Streaming Directive (CSD) before the outermost loop to force that loop to multi-stream:

```
!CSD$ PARALLEL DO PRIVATE( vars )
      do k = k1,k2
        do j = j1,j2
          do i = i1,i2
            (work)
          end do
        end do
      end do
!CSD$ END PARALLEL DO
```

In three other routines (*extrapi*, *extrapj*, and *extrapk*), there were several independent triply nested loops interspersed with quadruply nested loops:

```
     do nn = 1,nspeci
       do k = k1,k2
         do j = j1,j2
           do i = i1,i2
             (work)
           end do
         end do
       end do
     end do
```

Without compiler directives, the outmost loop was often multi-streamed, but a load imbalance resulted since *nspeci* was not a multiple of four. To improve the load balance, an artificial outermost loop of four iterations was inserted to split the range of the *k* loop into four nearly equal pieces. This new outermost loop was then multi-streamed with a CSD directive.

```
!CSD$ PARALLEL DO PRIVATE( vars )
      do kth = 1,4
        k1new = k1 + (k2-k1+4)/4*(kth-1)
        k2new = k1new + (k2-k1+4)/4 - 1
        do nn = 1,nspeci
          do k = k1new,k2new
            do j = j1,j2
              do i = i1,i2
                 (work)
              end do
            end do
          end do
        end do
         (other nested loops)
      end do
!CSD$ END PARALLEL DO
```

An experiment was made to compare the performance of the original and optimized codes on the Cray X1 for two problems: one using a mesh of 192^3, 16 MSP processors, and 500 time steps, and another using a mesh of 264^3, 32 MSP processors, and 4000 time steps. The timing results in Figure 1 show that there was a 20% improvement in run time for both experiments. For the optimized code, the proportion of time spent in computation decreased while the proportion of time spent in communication and statistics calculation increased. The statistics calculation did not involve MPI communications.

		Time	speedup	% comp	% comm	% stats
192^3	Orig	3:29		84.46%	11.06%	4.47%
16 PEs	Opt	2:47	20.10%	78.95%	15.42%	5.63%
264^3	Orig	44:35		83.78%	12.44%	3.78%
32 PEs	Opt	35:56	19.40%	76.26%	18.99%	4.75%

Figure 1: Timing statistics of original and optimized codes

Another experiment was conducted to use Cray PAT to gather hardware counter information during runs of the original and optimized LESlie3D codes when running the 192^3 mesh problem with 16 MSP processors. For selected counters, Figure 2 presents the ratio of the counters for the optimized code to the original code. The results show a dramatic decrease in the number of "Dcache misses" and of "Dcache bypass references." Scalar memory references, scalar integer operations, and scalar floating point operations all show significant reductions. Vector integer operations, vector shifts, and vector TLB misses all show large increases, presumably because more vector operations are being done.

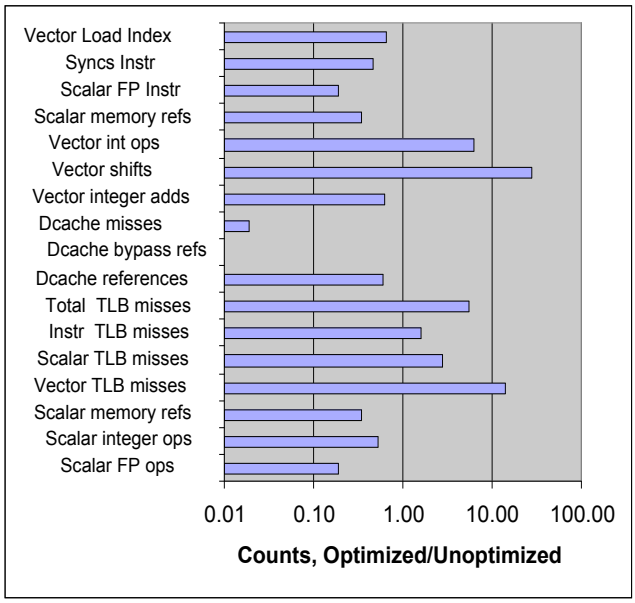

Figure 2: Cray PAT hardware counters before and after optimization

A comparison of the most CPU intensive subroutines was made between the original and optimized codes using Cray PAT to profile the codes. The mesh size was 264^3 and 32 MPI processes were used, configured in a 2 by 4 by 4 processor grid. The results are given in Figure 3.

	Before Optimization			After Optimization			ratio
	Rank	% Inst.	Instruc.	Rank	% Inst.	Instruc.	
emsk	1	19.1%	665814	4	11.9%	283487	0.43
emsi	2	13.5%	470255	8	4.1%	97009	0.21
emsj	3	12.6%	440026	1	14.6%	348132	0.79
mpi_send	4	11.2%	391191	3	12.5%	296980	0.76
extrapi	5	8.3%	287500	7	5.8%	138908	0.48
extrapj	6	8.2%	286391	6	7.2%	170984	0.60
update	7	7.0%	244424	5	11.5%	273378	1.12
extrapk	8	4.8%	167912	2	12.9%	306958	1.83
cxchg	9	3.3%	114326	10	3.4%	81737	0.71
mpi_bcast	10	2.4%	81905				
other		9.6%	334486		16.1%	383324	1.15
Total Inst.			3484230			2380897	0.68

Figure 3: LESlie3D profile before and after optimization

From Figure 3, it can be seen each of the six changed routines had reduced instruction counts except *extrapk*, which almost doubled. The reason for this behavior is unknown, but it can be observed that several unchanged routines also experienced significantly different instruction counts. Thus, run-to-run variability may be a factor.

The profiles of the original and optimized versions of LESlie3D are given in graphical form in Figures 9 and 10 in the Appendix.

Running LESlie3D in MSP and SSP modes

An experiment was performed to compare performance of the optimized LESlie3D code in SSP and MSP modes. Since the optimized code multi-streams well, it was believed that the MSP code, in which some synchronizations are done in hardware, would perform better than the SSP code, in which synchronizations are done through the MPI software. Also, communication costs for the MSP code were expected to be lower than for the SSP code since fewer MPI processes were being used so that fewer messages of larger size were being sent. Figures 4, 5, and 6 show that the optimized LESlie3D code had lower communication and computation costs, and thus lower run times, in MSP mode than in SSP mode for all the mesh sizes and processor counts tested. The raw data for these figures is given in Figures 11-14 in the Appendix.

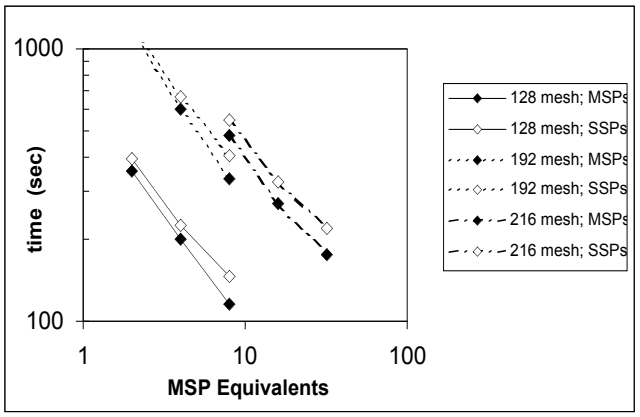

Figure 4: Total run time

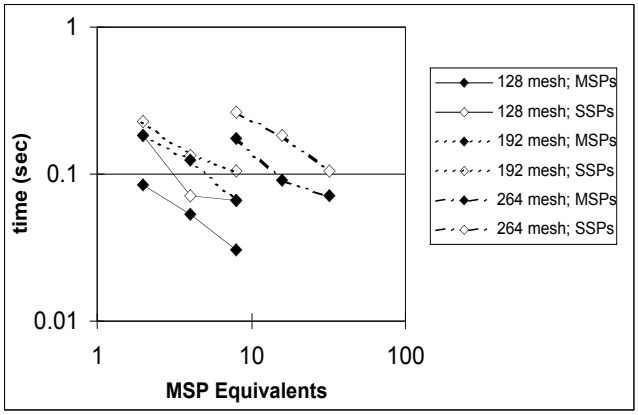

Figure 5: Communication time per time step

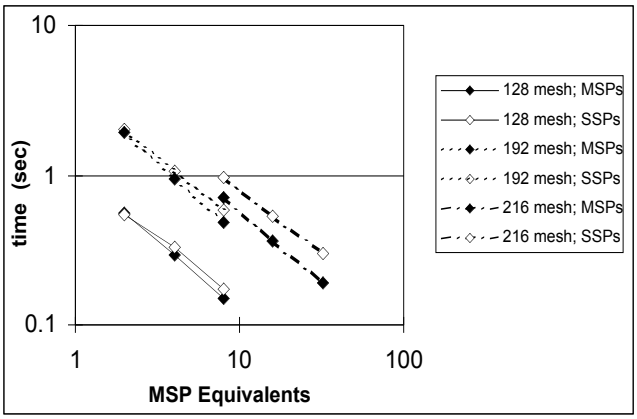

Figure 6: Computation time per time step

The LESlie3D code could be made to run efficiently on the Cray X1 platform with a few minor changes to six routines. The compiler was able to find enough work to multi-stream to effectively use the MSP processors. However, in the latest Programming Environment (PrgEnv 5.2), significant improvements have been made in several MPI routines that are heavily used in LESlie3D, in particular MPI_SEND and MPI_BCAST. With these improvements in the MPI software, the advantage of running in MSP mode over SSP mode may not be as significant.

In the collection of the run data, it was noticed that there was significant variability in the run times. This necessitated making several identical runs for a given mesh size and processor count and using the minimum time for the results obtained in this study.

4. Cray X1 and IBM PWR4 Counter Data

A number of runs were made on the Cray X1 at ERDC and the IBM PWR4 (P690) platform at NAVO to collect timings and hardware counter data. Figure 7 presents a comparison of timings on the Cray X1 and IBM P4 for a problem with a mesh size of $192^3$and using 16 processors running for 500 time steps, where a "processor" for the X1 is defined as an MSP. Even if a processor on the X1 is defined as an SSP, it can be seen that the X1 is over three times faster. For this size problem, it can also be seen that the X1 has a higher proportion of communication time.

	Cray X1	X1 * 4	% total	IBM P4	% total
Total run time	2:47	11:08		36:34	
Per time step					
Time	0.334	1.336	100%	4.392	100%
Calc. time	0.264	1.056	79%	3.43	78%
Comm. time	0.052	0.208	16%	0.471	11%
Stat. time	0.019	0.076	6%	0.49	11%

Figure 7: Cray X1 and IBM PWR4 comparison

Several runs were also made on the two platforms to compare their hardware performance characteristics when running LESlie3D. On the X1, Cray PAT was used to collect the counter data, while PAPI was used to collect similar counter data on the IBM. Because of the radically different architectures of the two platforms as well as the different software used to collect the counter data, making comparisons between specific counters on the two platforms is difficult. It is thought that a rough equivalence exists between the metrics presented in Figure 8. It can be seen that the X1 performs fewer branches, has fewer branch mispredictions, performs fewer integer operations, and has better cache behavior than the P4.

PAT counter	counts/MSP	PAPI counter	counts/PE
Cycles	2.80E+11	PM_CYC	2.83E+12
Branches & Jumps	1.38E+09	PM_BR_ISSUED	1.05E+11
Branches mispredicted	3.55E+07	PM_BR_MPED_CR+ PM_BR_MPRED_TA	2.18E+09
Total FP ops	3.52E+11	PM_FPU_ALL	1.71E+11
Vector int ops + Scalar int ops	1.12E+09	PM_FXU_FIN	3.34E+11
Dcache refs	8.72E+08	PM_LD_REF_L1	3.40E+11

Figure 8: Selected Cray PAT and PAPI counters

Selected PAPI counter events were collected for two runs: a run using a 192^3 mesh, 16 processors, and 500 time steps and another run using a 264^3 mesh, 54 processors, and 4000 time steps. The counter data for these runs is given in Figures 15 and 16 in the Appendix.

Conclusions

The LESlie3D code was optimized for the TI-03 application benchmark suite by making minor changes to six routines. These changes improved the multi-streaming efficiency of the code so that running in MSP mode was more efficient than running in SSP mode. The optimized code ran approximately 20% faster than the original code and performed more vector operations.

The performance of the optimized code on the X1 also compared favorably to its performance on the IBM P4, running 12 times faster at the MSP level and three times faster at the SSP level. The X1 also gets better hardware performance than the P4 in several areas: integer operations, cache behavior, and number of branches.

Appendix

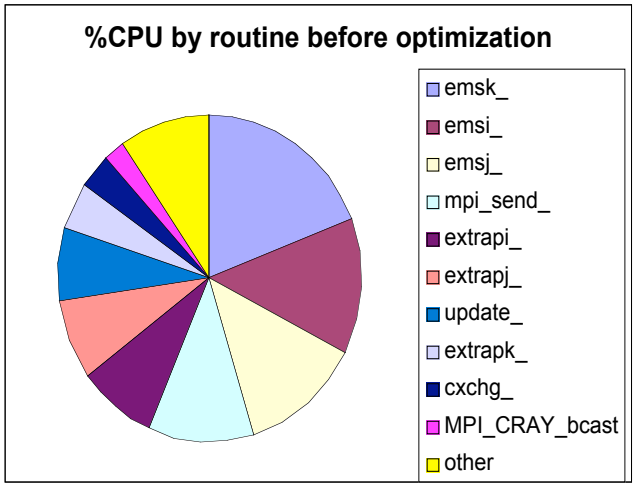

Figure 9: Profile of original code

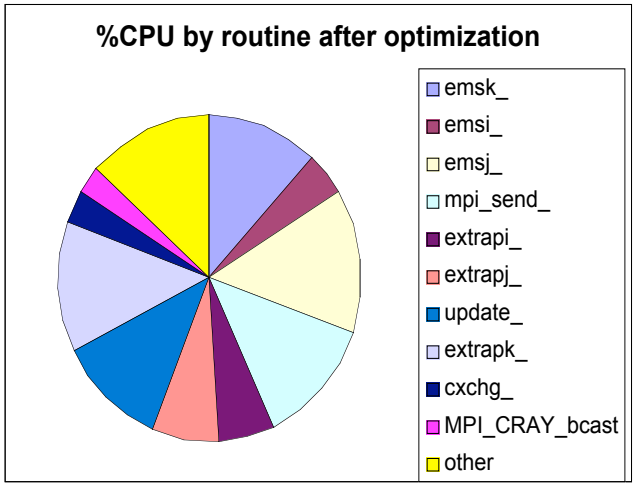

Figure 10: Profile of optimized code

PE's	Total time	time per time step			
		Total	Calc	Comm	Stat
2 MSPs	355.09	0.67856	0.55135	0.08378	0.04327
			81.25%	12.35%	6.38%
8 SSPs	393.64	0.76146	0.54415	0.18176	0.03546
			71.46%	23.87%	4.66%
MSP/SSP	0.9	0.89	1.01	0.46	1.22
4 MSPs	198.91	0.36681	0.29101	0.05275	0.02296
			79.33%	14.38%	6.26%
16 SSPs	226.46	0.42168	0.32827	0.07149	0.02188
			77.85%	16.95%	5.19%
MSP/SSP	0.88	0.87	0.89	0.74	1.05
8 MSPs	115.14	0.19309	0.15107	0.03035	0.01162
			78.24%	15.72%	6.02%
32 SSPs	146.12	0.25158	0.17502	0.06598	0.01055
			69.57%	26.23%	4.19%
MSP/SSP	0.79	0.77	0.86	0.46	1.1

Figure 11: 128^3 mesh, 500 time steps

PE's	Total time	time per time step			
		Total	Calc	Comm	Stat
2 MSPs	1151.09	2.2399	1.91329	0.183	0.14344
			85.42%	8.17%	6.40%
8 SSPs	1204	2.40882	2.04168	0.22827	0.13881
			84.76%	9.48%	5.76%
MSP/SSP	0.96	0.93	0.94	0.8	1.03
4 MSPs	597.2	1.13675	0.93917	0.12309	0.07439
			82.62%	10.83%	6.54%
16 SSPs	668.07	1.27098	1.06322	0.13524	0.07248
			83.65%	10.64%	5.70%
MSP/SSP	0.89	0.89	0.88	0.91	1.03
8 MSPs	333.71	0.5917	0.48827	0.06588	0.03751
			82.52%	11.13%	6.34%
32 SSPs	406.61	0.72734	0.58547	0.10579	0.03606
			80.49%	14.54%	4.96%
MSP/SSP	0.82	0.81	0.83	0.62	1.04

Figure 12: 192^3 mesh, 500 time steps

PE's	Total time	time per time step			
		Total	Calc	Comm	Stat
8 MSPs	478.42	0.84362	0.70287	0.08968	0.05101
			83.32%	10.63%	6.05%
32 SSPs	545.85	0.97378	0.7923	0.13205	0.0494
			81.36%	13.56%	5.07%
MSP/SSP	0.88	0.87	0.89	0.68	1.03
16 MSPs	269.25	0.44476	0.36494	0.05375	0.02605
			82.05%	12.08%	5.86%
64 SSPs	326.36	0.5365	0.42119	0.08996	0.02533
			78.51%	16.77%	4.72%
MSP/SSP	0.83	0.83	0.87	0.6	1.03
32 MSPs	175.12	0.24655	0.19052	0.04297	0.01304
			77.28%	17.43%	5.29%
128 SSPs	218.64	0.30149	0.22283	0.06585	0.01279
			73.91%	21.84%	4.24%
MSP/SSP	0.8	0.82	0.86	0.65	1.02

Figure 13: 216^3 mesh, 500 time steps

PE's	Total time	time per time step			
		Total	Calc	Comm	Stat
8 MSPs	6361.8	1.57096	1.30453	0.17396	0.09241
			83.04%	11.07%	5.88%
32 SSPs	7877.12	1.94647	1.58654	0.2628	0.09707
			81.51%	13.50%	4.99%
MSP/SSP	0.81	0.81	0.82	0.66	0.95
16 MSPs	3292.59	0.80402	0.66588	0.09099	0.04713
			82.82%	11.32%	5.86%
64 SSPs	4888.62	1.19687	0.95864	0.18293	0.05527
			80.10%	15.28%	4.62%
MSP/SSP	0.67	0.67	0.69	0.5	0.85
32 MSPs	1919.13	0.45975	0.3629	0.0716	0.02523
			78.93%	15.57%	5.49%
128 SSPs	2583.51	0.61233	0.48186	0.10482	0.02562
			78.69%	17.12%	4.18%
MSP/SSP	0.74	0.75	0.75	0.68	0.98

Figure 14: 264^3 mesh, 4000 time steps

EVENT	16 pe	54 pe
NUMBER_of_MSGs_DONE	0.00E+00	0.00E+00
PM_BIQ_IDU_FULL_CYC	2.48E+12	1.42E+13
PM_BR_ISSUED	1.05E+11	7.89E+11
PM_BR_MPRED_CR	7.08E+08	9.34E+09
PM_BR_MPRED_TA	1.47E+09	1.66E+10
PM_CYC	2.83E+12	1.71E+13
PM_DTLB_MISS	4.74E+08	3.33E+09
PM_FPU_ALL	1.71E+11	1.06E+12
PM_FPU_DENORM	5.50E+06	8.34E+07
PM_FPU_FDIV	2.57E+10	1.59E+11
PM_FPU_FIN	4.06E+11	2.51E+12
PM_FPU_FMA	7.87E+10	4.86E+11
PM_FPU_FMOV_FEST	7.27E+09	4.50E+10
PM_FPU_FRSP_FCONV	7.90E+06	6.56E+07
PM_FPU_FSQRT	1.42E+09	8.78E+09
PM_FPU_STALL3	1.31E+10	8.38E+10
PM_FPU_STF	1.23E+11	7.58E+11
PM_FPU0_FIN	2.25E+11	1.38E+12
PM_FPU1_FIN	1.82E+11	1.14E+12
PM_FXU_FIN	3.34E+11	2.49E+12
PM_FXU0_FIN	1.14E+11	9.56E+11
PM_FXU1_FIN	2.16E+11	1.56E+12
PM_GRP_DISP_REJECT	1.19E+12	6.78E+12
PM_GRP_DISP_VALID	1.55E+12	9.49E+12
PM_INST_CMPL	8.94E+11	6.40E+12
PM_INST_DISP	4.18E+12	2.52E+13
PM_INST_FETCH_CYC	2.74E+12	1.63E+13
PM_INST_FROM_L1	2.52E+11	2.01E+12
PM_INST_FROM_L2	5.77E+07	6.77E+08
PM_INST_FROM_L25_L275	3.19E+06	2.60E+07
PM_INST_FROM_L3	5.78E+06	4.31E+07
PM_INST_FROM_L35	0.00E+00	0.00E+00
PM_INST_FROM_MEM	8.88E+05	6.01E+06
PM_ITLB_MISS	1.87E+06	1.38E+07
PM_L1_WRITE_CYC	4.42E+08	5.29E+09
PM_LD_MISS_L1	7.24E+09	6.16E+10
PM_LD_REF_L1	3.40E+11	2.54E+12
PM_LSU_LDF	2.66E+11	1.65E+12
PM_ST_MISS_L1	6.03E+10	3.37E+11
PM_ST_REF_L1	1.52E+11	1.11E+12

Figure 15: IBM P4 PAPI events

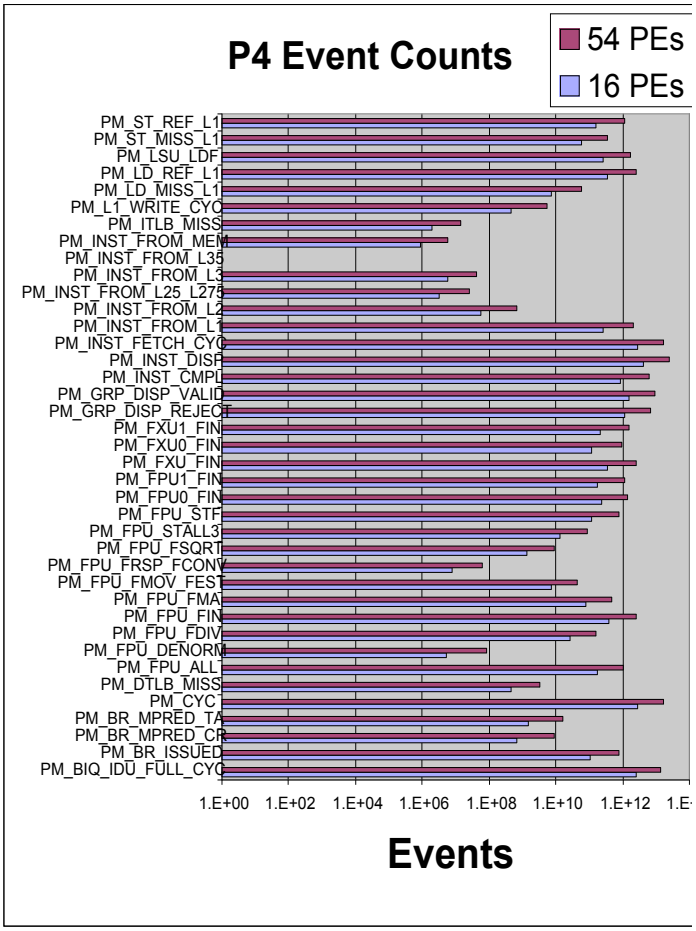

Figure 16: IBM P4 PAPI events

Acknowledgments

The authors would like to thank Dr. Alan Minga of Cray for his help in obtaining the optimized benchmark version of LESlie3D and for his guidance in the use of Cray PAT. We would also like to thank the Cray X1 system administrators and the support staff at the ERDC Information Technology Laboratory for their helpful assistance in using the machine.

About the Authors

Sam Cable and Thomas Oppe are Application Analysts in the Computational Science & Engineering group at the U.S. Army Engineer Research and Development Center Major Shared Resource Center (ERDC MSRC) in Vicksburg, MS. They can be reached at U. S. Army ERDC, ATTN: CEERD-IH, 3909 Halls Ferry Road, Vicksburg, MS 39180-6199, E-mail: Sam.B.Cable@erdc.usace.army.mil or Thomas.C.Oppe@erdc.usace.army.mil