Performance Analysis of PHASTA on NCSA Intel IA-64 Linux Cluster

Wai Yip Kwok

National Center for Supercomputing Applications, Champaign, IL 61820, USA
kwok@ncsa.uiuc.edu,
http://www.ncsa.uiuc.edu/∼kwok

Abstract. Performance of a computation-intensive multi-purpose CFD code PHASTA is analyzed on the NCSA Intel IA-64 Linux cluster. The capabilities of current-generation, open-source performance analysis tools available on this terascale system are demonstrated. Code profiling and hardware-performance counting tools are used to measure single-processor performance. Results pinpoint dominant but inefficient subroutines when level-3 optimization is used. Performance of these subroutines improves by compiling with level-2 optimization instead, due to reduction in total instructions. Flop rates of individual subroutines are estimated to guide further tuning. Parallel performance is addressed with performance visualization of inter-processor communication. Results reveal sporadic communication overhead in the function MPI_Waitall. This overhead constitutes about 18% of total simulation time.

1 Introduction

Cluster computers, which are commodity workstations connected by high-speed network, have emerged as a major source of terascale computing systems. New microprocessors such as the Intel IA-64 processors have much power to perform computing-intensive tasks. However, their cutting-edge technology also poses challenges to performance analysis and compiling.

In this study, we report success in using a variety of performance analysis tools to study the single-processor performance and multiple-processor communication of a computational fluid dynamics (CFD) code on the NCSA Intel IA-64 Linux cluster. The analysis provides vital information to code tuning. Section 2 describes the application code, NCSA Linux cluster, and performance analysis tools. Single-processor performance analysis is presented in Section 3. Multi-processor communication is discussed in Section 4. A summary is given in Section 5.

2 Preliminaries

This section provides basic information about the multi-purpose CFD code, the computer architecture, and the tools used to analyse performance of the code.

P.M.A. Sloot et al. (Eds.): ICCS 2003, LNCS 2660, pp. 43–52, 2003.

2.1 Application Code

Parallel Hierarchic Adaptive Stabilized Transient Analysis, or PHASTA software is developed and supported by the Scientific Computation Research Center at the Rensselaer Polytechnic Institute [5]. It can model three-dimensional, compressible or incompressible, laminar or turbulent, steady or unsteady flows, using unstructured grids. A stabilized finite-element formulation for fluid dynamics using mesh-entity hierarchical basis functions is used in the software.

PHASTA consists of more than 150 codes. Most of the codes are written in Fortran 90. These Fortran 90 subroutines carry out all the computation. A small number of codes are written in C and C++. Message passing interface (MPI) is used for data communication when multiple processors are used.

2.2 NCSA Intel IA-64 Linux Cluster

The NCSA Intel IA-64 Linux cluster is a distributed-memory system [3]. It is based on IBM IntelliStation Z Pro 6894 workstation. Each workstation (node) has two 800 MHz Intel Itanium I processors. There are more than 120 nodes. The cluster runs Red Hat Linux version 7.1 and kernel 2.4.16, and uses both Myrinet interconnect and Gigabit Ethernet network.

2.3 Performance Analysis Tools

A variety of tools are used to analyze the performance of PHASTA. GNU gprof and VProf are used for code profiling. GNU gprof is written by Jay Fenlason [2]. It can generate flat profiles, which show how much time is spent in each function, and how many times that function is called. VProf is developed by Curtis Janssen in Sandia National Laboratories [6]. It carries out statistical profiling for hardware-performance events. On the NCSA Itanium Linux cluster, VProf makes use of the Performance Application Programming Interface (PAPI) [4] to access the hardware-performance monitors. PAPI is developed by the Innovative Computing Laboratory at the University of Tennessee. VProf requires Linux kernel version 2.4.19 and PAPI version 2.3.1 to function properly. A node on the NCSA IA-64 cluster is upgraded to this kernel to enable the VProf analysis.

Overall code statistics are gathered with a tool named psrun. The tool psrun [8] is a command-line utility used to gather hardware-performance information on an unmodified executable. It achieves counting of multiple hardware-performance events (multiplexing) through PAPI. It is developed by Rick Kufrin at NCSA.

MPI communication among multiple processors are monitored and visualized with MultiProcessing Environment (MPE) libraries and Upshot [7] developed by Argonne National Laboratory. MPE and Upshot provide users with profiling and visualization tools for their MPI programs.

3 Single-Processor Performance Analysis and Tuning

After the code is ported to the IA-64 Linux cluster and completes some test problems correctly, its single-processor performance is analyzed. To validate the accuracy of profiling tools on the new IA- 64 architecture, both GNU gprof and VProf are used to ensure consistency. PHASTA, when compiled with level-3 optimization, produces profiles shown in Table 1. Profiles measured with GNU gprof and VProf are consistent. The functions e3ls, e3conv and e3wmlt occupy a large portion ($> 60\%$) of the total time consumed. This finding directs optimization efforts to these dominant subroutines.

Table 1. Time-profiles of PHASTA before tuning. All numbers are in percentage

Subroutine name	Percentage of time spent, measured with GNU gprof	Percentage of time spent, measured with VProf
e3ls	33.25	36.10
e3conv	17.43	16.50
e3wmlt	11.99	13.00
fillsparsec	5.61	5.90
e3ivar	4.03	3.00
e3massl	4.02	3.60
sparseap	3.63	3.20
e3mtrx	3.15	2.60
e3visc	1.96	1.30
e3	1.81	1.80
Others	13.12	13.00

When compilation of functions e3ls and e3conv is reviewed, the following warning message is noticed: 'Space exceeded in Data Dependence Test in e3ls (or e3conv). Subdivide routine into smaller ones to avoid optimization loss.' This warning appears regardless of size of grid, block, dataset, etc, as all the above information is fed into the code at the execution rather than the compilation stage. The compilation process of the whole code is then examined and it is found out that the following files induce the above warning: 1. asithf.f, 2. bardmc.f, 3. e3bdg_nd.f, 4. e3bdg.f, 5. e3bvar.f, 6. e3conv.f, 7. e3dc.f, 8. e3ls.f, 9. e3mtrx.f, 10. e3tau.f.

Various compiler flags and loop adjustment are explored to remove the cumbrance in the above codes. It is found at the end that the same subroutines, compiled with level-2 optimization, surprisingly run faster than their level-3 optimized counterparts. The main difference between level-3 and level-2 optimizations is its additional, more aggressive optimization. These optimization procedures, also named high-level language optimizations, include prefetching, scalar replacement, loop transformation, floating-point division optimization and

more data-dependency analysis [1]. The additional optimization affects memory access, instruction parallelism, predication, and software-pipelining. The resulting effects can be beneficial or harmful, depending on the particular code or loop. In the current situation, additional optimization ends up harmful.

As a result, the subroutines that fail the data-dependence test are compiled with level-2 optimization, while the rest stays with level-3 optimization. It should be noted that in subsequent paragraphs level-2 optimization refers to this mixed level-2 and level-3 optimization.

After the change in optimization level, wall-clock time spent in e3ls and e3conv reduces by 48% and 34%, respectively (Figure 1). In terms of percentage of time spent in individual functions, e3ls and e3conv consumes 22% and 15%, respectively, after tuning, down from 33% and 17% before tuning (Table 2).

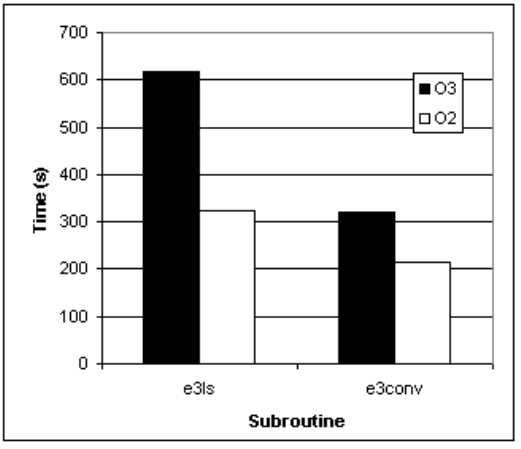

Fig. 1. Wall-clock time spent in subroutines e3ls and e3conv, compiled with level-3 (denoted O3) and level-2 (denoted O2), respectively. Wall-clock time is measured with GNU gprof

Profiles of hardware-performance events provide more information to understand the performance improvement. VProf collects profiling data on various hardware-performance events down to the line level. In this analysis, profile information is collected for the following events:

1. Cycles waiting for memory access (PAPI_MEM_SCY)
2. L3 cache misses (PAPI_L3_TCM)
3. L2 cache misses (PAPI_L2_TCM)
4. Total instructions (PAPI_TOT_INS)
5. Load/store instructions (PAPI_LST_INS)
6. Floating-point instructions (PAPI_FP_INS)
7. Total cycles (PAPI_TOT_CYC)

Table 2. Time-profiles of PHASTA using level-3 and level-2 optimizations, measured with GNU gprof. All numbers are in percentage

Subroutine	Level-3 optimized code	Level-2 optimized code
e3ls	33.25	22.30
e3conv	17.43	15.03
e3wmlt	11.99	15.27
fillsparsec	5.61	8.86
e3ivar	4.03	4.63
e3massl	4.02	4.89
sparseap	3.63	4.74
e3mtrx	3.15	3.36
e3visc	1.96	2.30
e3	1.81	2.30
Others	13.12	16.32

8. Time profile (PROF).

Table 3 shows the profiles for level-3 optimized code. In addition to the percentage profile, more interesting is the actual event counts in various subroutines. Consider a hardware-performance event in a subroutine s_i, the event count N_{s_i} can be obtained by multiplying the event count of the whole code (N_T) with the percentage of event count spent in that subroutine p_{s_i}, as indicated in Equation (1).

$$N_{s_i} = N_T \times p_{s_i}. \tag{1}$$

The total counts of hardware-performance events (N_T's) are measured with psrun, while the distribution of hardware-performance events among subroutines (p_{s_i}) is recorded by VProf (Table 3 for example). Table 4, which shows the event counts in individual subroutines for the level-3 optimized code, is thus derived from Table 3 and the results from psrun. Table 5 lists the same quantities for the level-2 optimized code.

Tables 4 and 5 reveal the distribution of instruction types. Significant portions of instructions are floating-point and load/store. A considerable percentage of instructions cannot be measured with PAPI in the case of level-3 optimization. Floating-point and load/store instructions carried out by various subroutines change slightly by switching optimization level. However, there is a drastic reduction in unclassified instructions carried out by subroutines e3ls and e3conv after tuning. This drop contributes substantially to the performance improvement. Performance tool pfmon pinpoints those unclassified instructions as instructions with no operations (NOPS_RETIRED). L3 cache misses in subroutines e3ls and e3conv also decrease slightly.

Flop rates of the dominant subroutines are shown in Figure 2 for the level-2 optimized code. It reveals that subroutines e3ls and e3conv behave above average. Further tuning should focus on subroutines e3wmlt and fillsparsec, which occupy 18% and 8% of the time spent and perform relatively poorly.

Table 3. Profile of hardware-performance events in level-3 optimized PHASTA, measured with VProf. All numbers are in percentage

Subroutine	MEM_ SCY	L3_ TCM	L2_ TCM	TOT_ INS	LST_ INS	FP_ INS	TOT_ CYC	PROF
e3ls	30.9	22.4	30.2	25.6	33.3	41.8	41.7	36.1
e3conv	15.4	15.0	14.9	13.6	12.7	12.3	18.1	16.5
e3wmlt	19.7	11.1	26.1	21.5	25.6	24.4	12.8	13.0
fillsparsec	3.1	2.6	0.8	2.8	1.9	0.9	2.3	5.9
e3massl	3.3	2.2	5.0	8.1	8.2	1.7	4.1	3.6
sparseap	3.4	14.4	4.4	1.9	2.0	0.9	0.8	3.2
e3ivar	2.4	4.9	2.6	3.5	1.5	2.7	3.5	3.0
e3mtrx	3.3	4.2	4.8	2.7	2.0	1.9	2.6	2.6
e3	2.6	1.7	1.1	1.2	2.0	0.0	0.9	1.8
e3metric	1.3	2.4	1.0	2.5	2.9	2.7	1.6	1.5
others	14.6	19.1	9.1	16.6	7.9	10.7	11.6	12.8

Table 4. Counts of occurrence of hardware performance events in level-3 optimized PHASTA, estimated using psrun and VProf. All numbers are in 10^9, except for FLOP rate

Subroutine	MEM_ SCY	L3_ TCM	L2_ TCM	TOT_ INS	LST_ INS	FP_ INS	TOT_ CYC	FLOP rate (mflops)
e3ls	207	1.26	22.53	403	170	133	703	150
e3conv	103	0.84	11.12	214	65	39	305	102
e3wmlt	132	0.62	19.47	338	131	78	216	286
fillsparsec	21	0.15	0.60	44	10	3	39	59
e3massl	22	0.12	3.73	127	42	5	69	62
sparseap	23	0.81	3.28	30	10	3	13	169
e3ivar	16	0.28	1.94	55	8	9	59	116
e3mtrx	22	0.24	3.58	42	10	6	44	110
e3	17	0.10	0.82	19	10	0	15	0
e3metric	9	0.13	0.75	39	15	9	27	253
others	98	1.07	6.79	261	40	34	196	138
Total	671	5.619	74.601	1574	512	318	1687	150

Table 5. Counts of occurrence of hardware performance events in level-2 optimized PHASTA, estimated using psrun and VProf. All numbers are in 10^9, except for FLOP rate

Subroutine	MEM_ SCY	L3_ TCM	L2_ TCM	TOT_ INS	LST_ INS	FP_ INS	TOT_ CYC	FLOP rate (mflops)
e3ls	187	0.99	21.06	246	153	136	245	440
e3conv	81	0.58	11.63	79	49	48	91	415
e3wmlt	162	0.76	19.39	325	126	78	281	220
fillsparsec	27	0.18	0.76	42	10	3	53	45
e3massl	28	0.11	4.11	122	44	6	92	49
sparseap	30	1.03	3.19	29	11	3	27	88
e3ivar	19	0.31	1.75	53	9	8	72	89
e3mtrx	28	0.19	6.23	43	13	7	53	100
e3	22	0.11	0.76	17	11	0	22	0
e3metric	12	0.15	0.68	37	13	9	38	180
others	124	1.20	6.46	247	44	37	258	113
Total	720	4.51	76.03	1240	484	334	1233	215

Regarding overall performance, PHASTA runs at 220 Mflops (CPU cycles) and 170 Mflops (effective). Compared with the NCSA SGI Origin2000 supercomputer, PHASTA performs 2.4 times faster on the Itanium cluster on a single processor.

4 Multi-processor Communication Analysis

Message Passing Interface (MPI) is implemented in PHASTA for parallel computing. Communication among processors is monitored with Multi-Processing Environment (MPE) libraries. Several multi-processor simulations are run to evaluate the inter-processor communication. Each simulation uses 4 processors and contains 20 iterations. The log files are viewed using the logfile viewer Upshot, and the visualization is shown in Figure 3. Heavy communication is indicated by bundles of black arrows. In each simulation, twenty bundles can be counted, corresponding to the twenty iterations. The time required per iteration is not even. The simulation alternates between long and short time-steps. A long and short time-step takes about 8 and 2 seconds, respectively. The reason for the variation is the re-use of matrix every other step.

It is noteworthy that there are several 'extra-long' time-steps that are marked by rough MPI communication and idleness in processors due to 'MPI_Waitall'. This communication overhead appears at different iterations in simulations carrying out the same calculation, suggesting that the code is not responsible for the peculiar delay. One likely cause is congestion in message communication through the Myrinet switches. Due to synchronization of the code, a large wait-all time on one processor affects all the other processors.

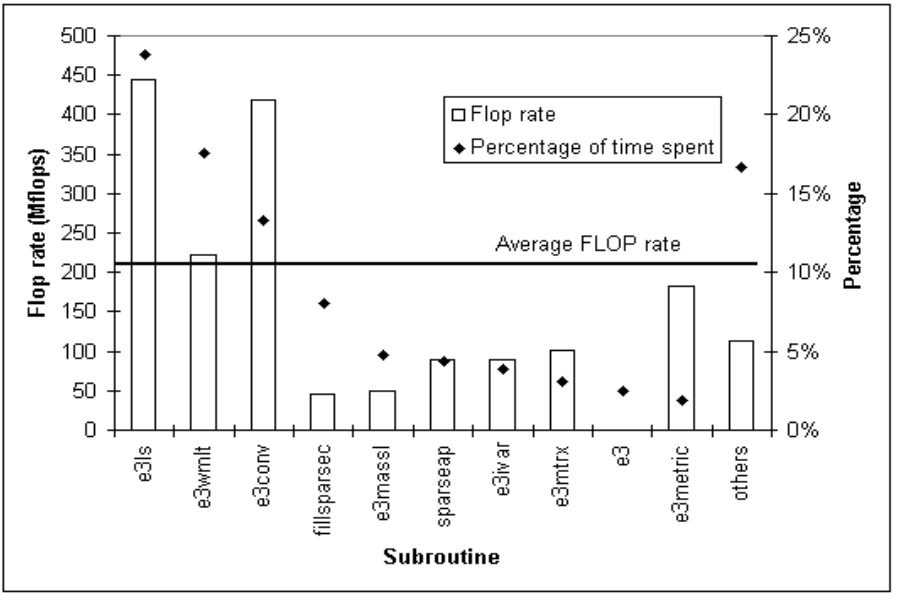

Fig. 2. Flop rate and time profile of dominant subroutines in level-2 optimized PHASTA, estimated using VProf and psrun

An attempt is made to estimate the adverse effect of the congestion in message communication quantitatively. The communication overhead depends on two factors: 1. frequency at which the congestion occurs; 2. amount of time delay the congestion induces. Five tests with 100 iterations each are run to gather relevant statistics. Time-steps are divided into three categories: short time-steps with duration less than 5 seconds, long time-steps with duration between 5 and 10 seconds, and extra-long time-steps that last more than 10 seconds due to communication overhead. The statistics of all five simulations is shown in Table 6. The percentage of time spent in communication overhead s_{oh} is estimated with Equation (2).

$$\begin{aligned} s_{oh} &= N_{el} \times \frac{\tau_{el} - \tau_l}{N_s \times \tau_s + N_l \times \tau_l + N_{el} \times \tau_{el}} \\ &= 83 \times \frac{13.9 - 7.68}{244 \times 176 + 173 \times 7.68 + 83 \times 13.9} \\ &= 18\%. \end{aligned} \tag{2}$$

As a result, a speed-up of about 3.3 instead of an ideal 4 is obtained when 4 processors are used.

A nice feature of Upshot is that it allows one to change the resolution to a much finer level for deeper understanding of data communication. This study is

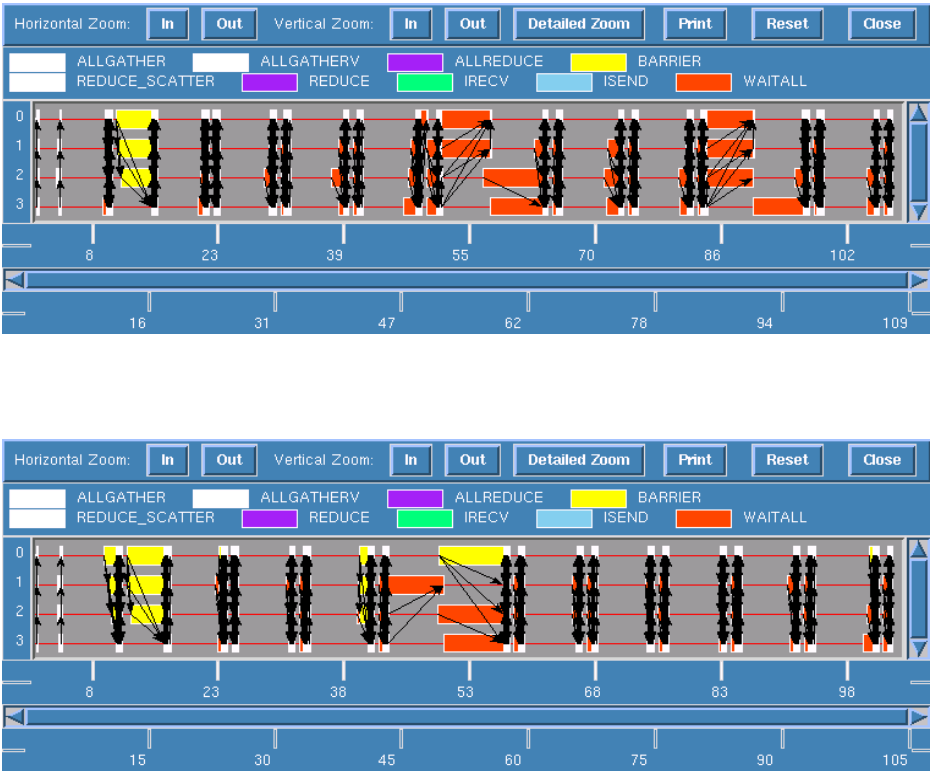

Fig. 3. MPI communication among 4 processors in 2 test simulations, visualized with Upshot. Each simulation contains 20 iterations. Time from 0 to about 100 seconds is represented in the x-axis. The numbers from 0 to 3 on the y-axis are the processor indices

Table 6. Duration of time-steps of five PHASTA benchmark simulations

Simulation	1	2	3	4	5	Total
Number of short time-steps, N_s	49	49	49	49	48	244
Number of long time-steps, N_l	33	30	34	42	34	173
Number of extra-long time-steps, N_{el}	18	21	17	9	18	83
Average length of a short time-step, τ_s (seconds)				1.66		
Average length of a long time-step, τ_l (seconds)				7.68		
Average length of an extra-long time-step, τ_{el} (seconds)				13.9		

conducted on some sample time-steps but not covered in this paper due to limit in space.

A direction of further investigation is an analysis of PHASTA on a larger machine configuration. This analysis requires PHASTA researchers to custom-build the mesh and input conditions, and is currently being pursued.

5 Summary

Single-processor performance and multi-processor communication of a CFD code PHASTA on the NCSA Intel IA-64 Linux cluster are studied. Code profiling and study of compilation process suggest that level-3 optimization hinders performance of some dominant subroutines. Changing optimization to level-2 for the affected subroutines improves performance of PHASTA. Hardware-performance events such as various instructions and cache misses are counted at the subroutine level. Results reveal that the change in optimization level leads to a drop in instructions with no operations (NOPS_RETIRED). Inter-processor communication is visualized with MPE libraries and Upshot. Results show that communication overhead occurs sporadically with the subroutine MPI_Waitall. This delay will be further investigated.

Acknowledgements.
The author would like to thank Kenneth Jansen and Anil Karanam at Rensselaer Polytechnic Institute for explaining the functions of PHASTA. Rick Kufrin and Greg Bauer at NCSA provided vital help in the use of performance analysis tools. The author also had constructive discussions with Faisal Saied on various code optimization issues.

References

1. Intel Fortran compiler user's guide. 2002.
2. GNU gprof homepage. http://www.gnu.org/manual/gprof-2.9.1/gprof.html. 2003.
3. NCSA homepage. http://www.ncsa.uiuc.edu. 2003.
4. PAPI homepage. http://icl.cs.utk.edu/projects/papi.
5. Scientific Computation Research Center homepage. http://www.scorec.rpi.edu. 2003.
6. VProf homepage. http://aros.ca.sandia.gov/~cljanss/perf/vprof.
7. MPE libraries and Upshot homepage. http://www-unix.mcs.anl.gov/perfvis/software/viewers.
8. psrun homepage. http://perfsuite.ncsa.uiuc.edu/psrun.