A Study of MPI Performance Analysis Tools on Blue Gene/L

I-Hsin Chung, Robert E. Walkup, Hui-Fang Wen and Hao Yu

IBM Thomas J. Watson Research Center
Yorktown Heights, NY 10598
{ihchung,walkup,hfwen,yuh}@us.ibm.com

Abstract

Applications on todays massively parallel supercomputers rely on performance analysis tools to guide them toward scalable performance on thousands of processors. However, conventional tools for parallel performance analysis have serious problems due to the large data volume that may be required. In this paper, we discuss the scalability issue for MPI performance analysis on Blue Gene/L, the worlds fastest supercomputing platform. We present an experimental study of existing MPI performance tools that were ported to BG/L from other platforms. These tools can be classified into two categories: profiling tools that collect timing summaries, and tracing tools that collect a sequence of time-stamped events. Profiling tools produce small data volumes and can scale well, but tracing tools tend to scale poorly. The experimental study discusses the advantages and disadvantages for the tools in the two categories and will be helpful in the future performance tools design.

1 Introduction

The Blue Gene[1]/L (BG/L) supercomputer is a massively parallel system developed by IBM in partnership with Lawrence Livermore National Laboratory (LLNL). BG/L uses system-on-a-chip technology [16] to integrate powerful torus and collective networks onto a single chip with the processors, and it uses a novel software architecture [14] to support high levels of scalability. The BG/L system installed at LLNL contains 65,536 dual-processor compute nodes. Operating at a clock frequency of 700 MHz, BG/L delivers 180 or 360 Teraflop/s of peak computing power, depending on its mode of operation.

Solving a scientific problem on BG/L typically requires more processors than conventional supercomputers because BG/L has relatively low-power processors, both in terms of Watts consumed and Flops produced. Therefore, to effectively utilize the enormous computation power provided by BG/L systems, it is critical to explore scalability to the maximum extent. It has been shown that a significant number of important real-world scientific applications can be effectively scaled to thousands or tens of thousands BG/L processors [18]. The process of achieving such high level scalability and performance is often difficult and time-consuming, but can be guided with scalable parallel performance analysis tools.

Performance analysis tools produce information collected at run-time to help program developers identify performance bottlenecks and in turn improve application performance. For massively parallel systems like BG/L where scalability is the key, scalable tools for analyzing communication are needed. On BG/L, communication is via the message-passing interface, MPI. The MPI specification includes a "profiling" interface, which provides entry points to intercept and instrument the message-passing calls. One can distinguish between two types of parallel performance tools: (1) profiling tools that generate cumulative timing information, and (2) tracing tools that collect and display a sequence of time-stamped events. It is clear that tracing tools face a fundamental difficulty when there are thousands of processes involved. The volume of trace data can easily get to be unmanageable. As a result, post-mortem data reduction schemes based on filtering the aggregate data may still be untenable. Instead, it is necessary to carefully control trace generation. In contrast, profiling tools are inherently more scalable, because they retain only cumulative data.

In this paper, we discuss the scalability of MPI profiling and tracing tools on BG/L. We discuss existing MPI tracing tools and show that they have serious problems for large-scale BG/L systems. Specifically, we present our study of existing tools that are ported onto BG/L from other platforms.

The rest of the paper is organized as follows. Section 2 gives an overview of MPI on the BG/L. We study the scalability of existing MPI performance analysis tools on BG/L by using several real applications in Section 3 and

[1]Trademark or registered trademark of International Business Machines Corporation.

1-4244-0054-6/06/$20.00 ©2006 IEEE

4. The experiences and lessons learned are described in Section 5. Related work is given in Section 6. Finally, the conclusion is drawn in Section 7.

2 MPI on BG/L

On BG/L, each process can directly access only its local memory, and message-passing is used for communication between processes. The current implementation of MPI on BG/L [17] is based on MPICH2 [5] from Argonne National Laboratory. The BG/L version is MPI-1.2 compliant [15] and supports a subset of the MPI-2 standard. There are parts of MPI-2, such as dynamic process management, that are not supported. Other parts of MPI-2, such as one-sided communication, are under development and testing at the time of this writing.

The MPI implementation on BG/L utilizes three high-speed networks: a three dimensional (3D) torus network for point-to-point communication [13], a collective network for broadcast and reduction operations, and a global interrupt network for fast barrier synchronization. For the torus network, each compute node is connected to its six neighbors through bidirectional links, and the links can be "trained" to wrap around on partitions that are multiples of a midplane (a unit with 512 nodes in an 8x8x8 torus network configuration).

Another important architectural feature of BG/L is that each compute node has two processors. A compute node can operate in one of two modes. In *coprocessor* mode, a single-threaded MPI process occupies each node, and the second processor is used to aid in MPI communication. In *virtual node* mode, there are two single-threaded MPI processes per node, each with access to half of the node memory. The two processors on each chip do not have hardware support for cache coherency, so pthreads or OpenMP are not currently supported on BG/L. As a result, MPI has a very important role for parallel applications on BG/L.

3 MPI Performance Analysis Tools for BG/L

A number of MPI performance analysis tools have been made available for BG/L users by joint efforts of IBM and collaborators. In this section, we first give brief introductions to these tools. Then, in Section 4, we present an experimental evaluation of the scalability, efficiency, and overhead of these tools when applying them on BG/L. Table 1 lists the MPI performance tools that were ported to BG/L and used in this study. These tools can be classified into two categories: (1) profiling tools that provide cumulative data, and (2) tracing tools that provide time-stamped records of MPI events.

Table 1. Existing MPI Performance Analysis Tools

Tool	Function	References
IBM HPCT	profiling and tracing	[3]
Paraver	tracing	[8, 26]
KOJAK	tracing	[4, 25]
TAU	profiling and tracing	[9, 23]
mpiP	profiling	[6]

3.1 IBM High Performance Computing Toolkit

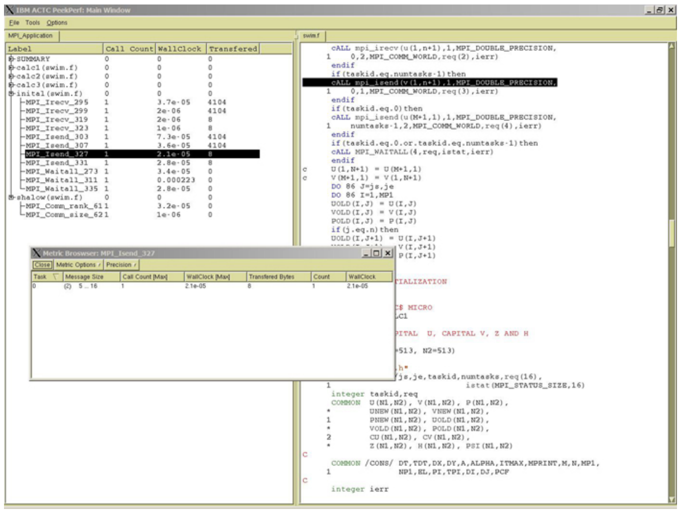
The IBM High Performance Computing Toolkit (IBM HPCT) contains MPI profiler and tracer libraries [3] to collect profiling and trace data for MPI and SHMEM programs. This toolkit was originally developed for AIX Power clusters and now is ported to BG/L. The IBM HPCT generates two types of output files that work with the visualization tools Peekperf and Peekview to identify performance bottlenecks. Peekperf integrates a source-code browser with cumulative performance metrics obtained with the profiler library (Figure 1(a)), a nd Peekview displays the time-stamped MPI trace file obtained via the tracer library (Figure 1(b)).

3.2 Paraver

Paraver [8, 26] is a program visualization and analysis tool that supports both shared-memory and distributed-memory parallel applications. It has three major components: a tracing facility, a trace merge tool, and a visualizer. For MPI tracing, the MPItrace library is used to intercept MPI calls and save individual trace files during application execution. The individual files are then merged, and the merged trace file is displayed using the viewer, which has many display and analysis features. An example of Paraver for MPI trace visualization is shown in Figure 2. Both the trace merge tool and the viewer run on BG/L front-end nodes, while trace generation is done from the BG/L compute nodes. Paraver is best suited for parallel applications at a modest scale by BG/L standards, because at large process counts the trace files become large and hard to work with. This basic difficulty affects all tracing tools to some extent.

3.3 KOJAK

KOJAK (Kit for Objective Judgment and Knowledge-based Detection of Performance Bottlenecks) [4, 25] is a collaborative research project aiming at the development of a generic automatic performance analysis environment for parallel programs. It includes a set of tools performing program analysis, tracing, and visualization. The



IBM ACTC PeekPerf: Main Window

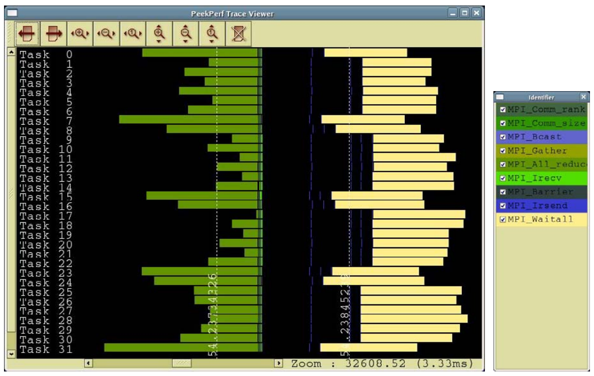
Label	Call Count	WallClock	Transfered
SUMMARY	0	0	0
calc1(swim.f)	0	0	0
calc2(swim.f)	0	0	0
calc3(swim.f)	0	0	0
inital(swim.f)	0	0	0
MPI_Irecv_295	1	3.7e-05	4104
MPI_Irecv_299	1	2e-06	4104
MPI_Irecv_319	1	2e-06	8
MPI_Irecv_323	1	1e-06	8
MPI_Isend_303	1	7.3e-05	4104
MPI_Isend_307	1	3.6e-05	4104
MPI_Isend_327	1	2.1e-05	8
MPI_Isend_331	1	2.8e-05	8
MPI_Waitall_273	1	3.4e-05	0
MPI_Waitall_311	1	0.000223	0
MPI_Waitall_335	1	2.8e-05	0
shalow(swim.f)	0	0	0
MPI_Comm_rank_61	1	3.2e-05	0
MPI_Comm_size_62	1	1e-06	0

Metric Broswser: MPI_Isend_327

Task	Message Size	Call Count [Max]	WallClock [Max]	Transfered Bytes	Count	WallClock
0	(2) 5 ... 16	1	2.1e-05	8	1	2.1e-05

```
      cALL mpi_irecv(u(1,n+1),1,MPI_DOUBLE_PRECISION,
     1     0,2,MPI_COMM_WORLD,req(2),ierr)
      endif
      if(taskid.eq.numtasks-1)then
      cALL mpi_isend(v(1,n+1),1,MPI_DOUBLE_PRECISION,
     1     0,1,MPI_COMM_WORLD,req(3),ierr)
      endif
      if(taskid.eq.0)then
      cALL mpi_isend(u(M+1,1),1,MPI_DOUBLE_PRECISION,
     1     numtasks-1,2,MPI_COMM_WORLD,req(4),ierr)
      endif
      if(taskid.eq.0.or.taskid.eq.numtasks-1)then
      cALL MPI_WAITALL(4,req,istat,ierr)
      endif
c     U(1,N+1) = U(M+1,1)
c     V(M+1,1) = V(1,N+1)
      DO 86 J=js,je
      DO 86 I=1,MP1
      UOLD(I,J) = U(I,J)
      VOLD(I,J) = V(I,J)
      POLD(I,J) = P(I,J)
      if(j.eq.n)then
      UOLD(I,J+1) = U(I,J+1)
                  V(I,J+1)
                  P(I,J+1)

                  TIALIZATION

                  C$ MICRO
                  LC1

                  PITAL  U, CAPITAL V, Z AND H

                  =513, N2=513)

                  ,h*
                  /js,je,taskid,numtasks,req(16),
     1                            istat(MPI_STATUS_SIZE,16)
      integer taskid,req
      COMMON  U(N1,N2), V(N1,N2), P(N1,N2),
     *        UNEW(N1,N2), VNEW(N1,N2),
     1        PNEW(N1,N2), UOLD(N1,N2),
     *        VOLD(N1,N2), POLD(N1,N2),
     2        CU(N1,N2), CV(N1,N2),
     *        Z(N1,N2), H(N1,N2), PSI(N1,N2)
C
      COMMON /CONS/ DT,TDT,DX,DY,A,ALPHA,ITMAX,MPRINT,M,N,MP1,
     1              NP1,EL,PI,TPI,DI,DJ,PCF
C
      integer ierr
```

(a) Source Code Performance Metrics Mapping



(b) Trace Visualization

Figure 1. IBM HPCT Visualization for MPI Performance Analysis

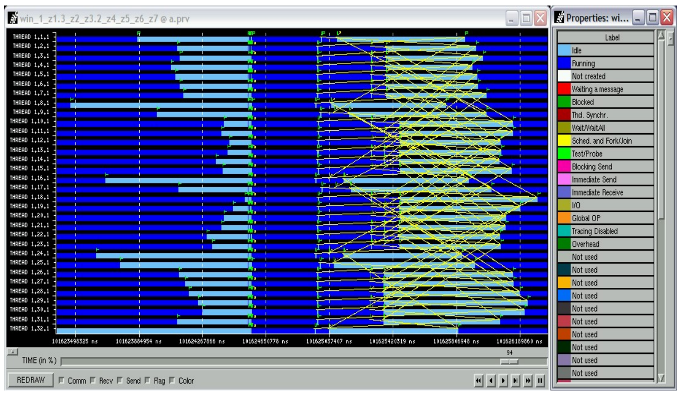
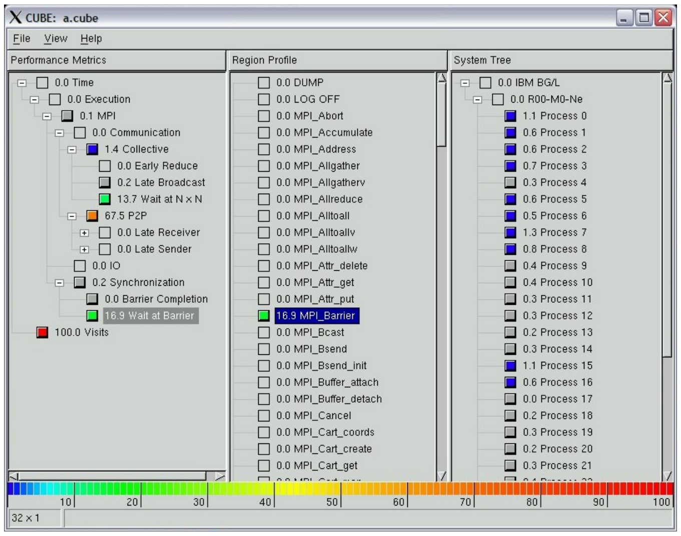


Figure 2. Paraver Visualization for MPI Tracing

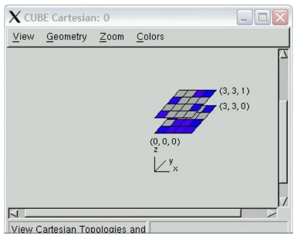


CUBE: a.cube

File View Help

```
Performance Metrics            Region Profile          System Tree
0.0 Time                       0.0 DUMP                0.0 IBM BG/L
  0.0 Execution                0.0 LOG OFF               0.0 R00-M0-Ne
    0.1 MPI                    0.0 MPI_Abort               1.1 Process 0
      0.0 Communication        0.0 MPI_Accumulate          0.6 Process 1
        1.4 Collective         0.0 MPI_Address             0.6 Process 2
          0.0 Early Reduce     0.0 MPI_Allgather           0.7 Process 3
          0.2 Late Broadcast   0.0 MPI_Allgatherv          0.3 Process 4
          13.7 Wait at N x N   0.0 MPI_Allreduce           0.6 Process 5
        67.5 P2P               0.0 MPI_Alltoall            0.5 Process 6
          0.0 Late Receiver    0.0 MPI_Alltoallv           1.3 Process 7
          0.0 Late Sender      0.0 MPI_Alltoallw           0.8 Process 8
      0.0 IO                   0.0 MPI_Attr_delete         0.4 Process 9
      0.2 Synchronization      0.0 MPI_Attr_get            0.4 Process 10
        0.0 Barrier Completion 0.0 MPI_Attr_put            0.3 Process 11
        16.9 Wait at Barrier   16.9 MPI_Barrier            0.3 Process 12
100.0 Visits                   0.0 MPI_Bcast               0.2 Process 13
                               0.0 MPI_Bsend               0.3 Process 14
                               0.0 MPI_Bsend_init          1.1 Process 15
                               0.0 MPI_Buffer_attach       0.6 Process 16
                               0.0 MPI_Buffer_detach       0.0 Process 17
                               0.0 MPI_Cancel              0.2 Process 18
                               0.0 MPI_Cart_coords         0.3 Process 19
                               0.0 MPI_Cart_create         0.2 Process 20
                               0.0 MPI_Cart_get            0.3 Process 21
```

(a) Hotspot Analysis



(b) Topology

Figure 3. KOJAK Visualization for MPI Profiling

instrumentation for MPI is obtained with the PMPI interface, which intercepts calls to MPI functions. KOJAK uses the EPILOG run-time library, which provides mechanisms for buffering and trace-file creation. On BG/L, the resulting trace files can be quite large. In terms of visualization, KOJAK provides several options including tree-style hot spot analysis (Figure 3(a)). The user can identify performance bottlenecks by exploring the tree. There is also a topology view (Figure 3(b)) to help the user map performance metrics back to the compute nodes in terms of the physical layout on the torus network.

3.4 TAU

TAU [9, 23] is a program and performance analysis framework. It includes a suite of static and dynamic tools that form an integrated analysis environment for parallel applications. TAU includes automatic instrumentation to capture data for functions, methods, basic blocks, and program statements. In addition to automatic instrumentation, TAU provides an API for manual instrumentation. TAU can be used for either profiling (collecting cumulative data) or tracing (recording time-stamped events). TAU includes a visualizer, Paraprof, for profile data. For trace data, TAU does not include it's own trace viewer, but it can convert trace files for use with other visualization tools, such as Paraver. TAU has many features and has been ported to a variety of platforms. In our experiments with TAU, we used only the MPI profiling capability.

3.5 mpiP

mpiP [6] is a light-weight profiling library for MPI applications. It collects cumulative information about MPI functions. It uses communication only during report generation, typically at the end of the program execution. Since it collects only cumulative information, the output size is very small compared to MPI tracing tools, and the execution time overhead is normally small. However, the detailed time history of communication events is not available with this tool.

4 Performance Study

In this section, we study the scalability, efficiency, and overheads of the existing MPI performance analysis tools on BG/L.

4.1 Applications

We used a collection of scientific applications on BG/L to examine the strengths and limitations of parallel performance tools. The applications are briefly described below.

SAGE [22] is an Adaptive Grid Eulerian hydrodynamics application from Science Applications International. SAGE uses blocks of cells in a three-dimensional Cartesian framework. SAGE has many features including the ability to do automatic mesh refinement. Two input sets were used: one that does significant computational work but has a static mesh, and another that exercises the automatic mesh refinement capability.

FLASH [2] is a parallel adaptive-mesh multi-physics simulation code designed to solve nuclear astrophysical problems related to exploding stars. The particular test case that we used was a two-dimensional weak scaling problem [29], which includes an expanding shock wave and exercises the adaptive-mesh refinement capability.

SOR is a program for solving the Poisson equation using an iterative red-black SOR method. This code uses a two dimensional process mesh, where communication is mainly boundary exchange on a static grid with east-west and north-south neighbors. This results in a simple repetitive communication pattern, typical of grid-point codes from a number of fields.

POP (Parallel Ocean Program) [12,20,21,28] is a parallel ocean circulation model. We used the input file included in POP version 2.0.1, with a fixed domain size as the test case. The domain is decomposed into blocks, and blocks are distributed among the processors. In contrast to the other codes, POP was used in a strong-scaling mode, where the total problem size is fixed, and the work per processor decreases as the application is scaled up.

sPPM [10] is a simplified version of the Piecewise Parabolic Method (PPM), a hydrodynamics algorithm which is particularly useful when there are discontinuities such as shock waves . The application uses a three-dimensional process mesh, and communication is by exchange with nearest neighbors in +/-x, +/-y, and +/- z directions. This communication pattern fits nicely onto the 3D torus network, and as a result sPPM scales almost perfectly on BG/L.

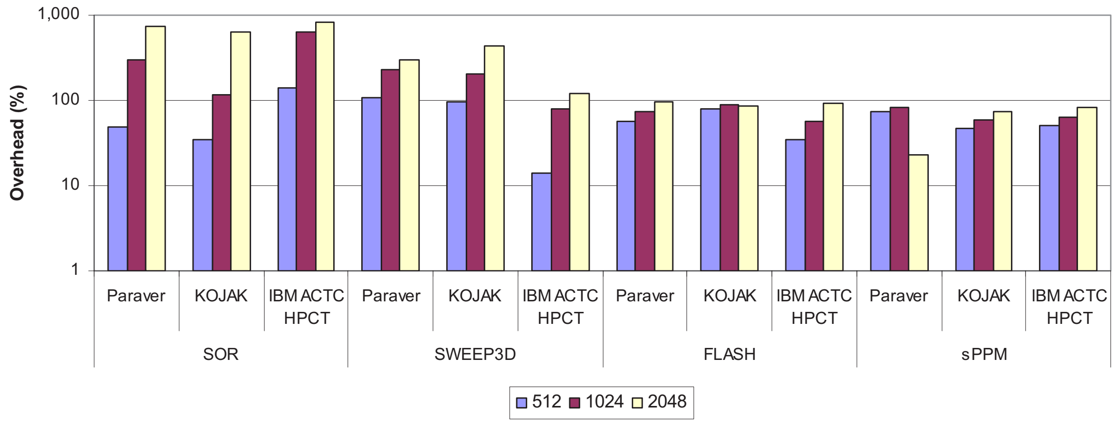
SWEEP3D [11] is a simplified benchmark program that solves a neutron transport problem using a pipelined wave-front method and a two-dimensional process mesh. Input parameters determine problem sizes and blocking factors, allowing for a wide range of message sizes and parallel efficiencies.

4.2 Performance Measurement

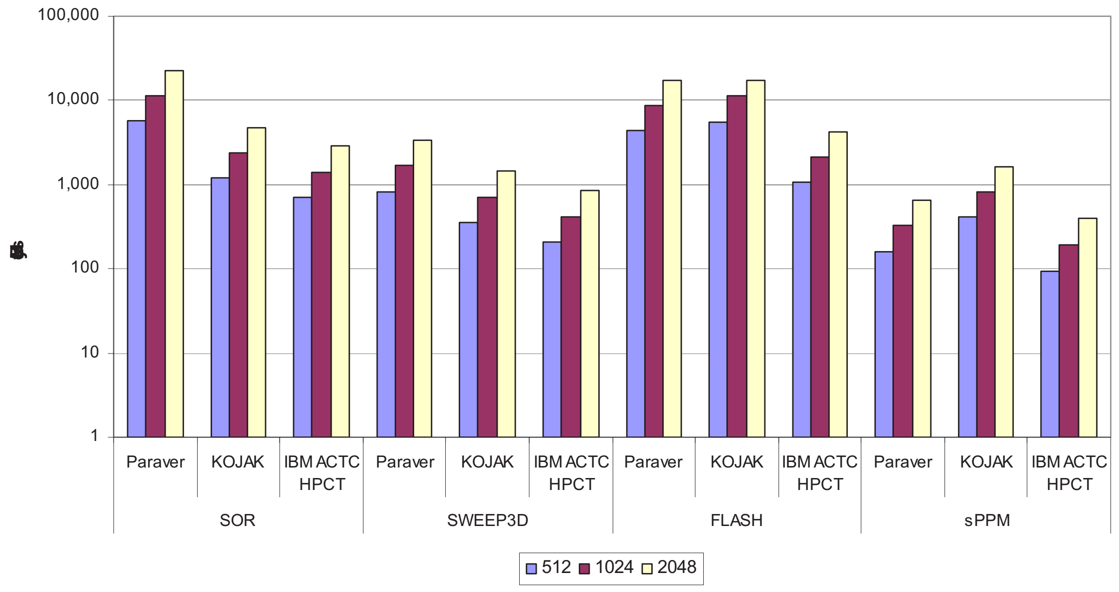
In this section we report measurements using the existing MPI performance tools with the collection of applications listed in Section 4.1. Two metrics were chosen to characterize the tools: (1) execution-time overhead ($(elapsed\_time - reference\_time)/reference\_time$, where the $reference\_time$ is measured without profiling or tracing), and (2) the volume of collected data.

Figures 4(a) and 4(b) show the measurements, using weak scaling (work per processor remains constant) for a set of applications, with 512, 1,024, and 2,048 compute nodes on BG/L in coprocessor mode. The execution-time overhead for the profiling methods provided by TAU and mpiP was less than 3% and is not shown in the figures. In our experience, profiling tools can provide useful cumulative data up to very large systems, including the full 64K-node Blue Gene/L at Livermore, with relatively little overhead. In contrast, the execution-time overhead for the tracing methods used by IBM HPCT, Paraver, and KOJAK grows faster than linearly with the number of MPI processes, and is already of order 100(factor of two slower performance) with 1,024 processes. The overall performance is slowed down significantly with 2,048 or more MPI processes. Also, the volume of trace data collected, shown in Figure 4(b), can quickly grow to of order 100 GBytes, which is too large for efficient analysis or visualization. At very large process counts, it is simply not practical to save all time-stamped records from every MPI rank.

A closer examination reveals that the execution-time overhead is substantially affected by the generation of trace file output. The trace file I/O behavior is not uniform over the course of program execution. For example, the I/O bandwidth obtained when tracing FLASH

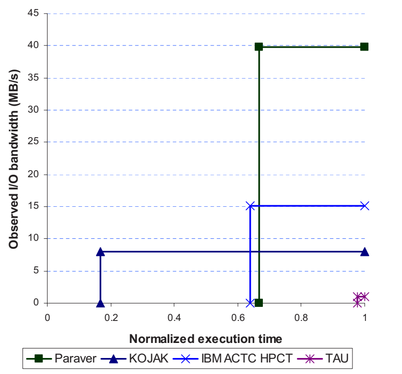


(a) Execution Time Overhead

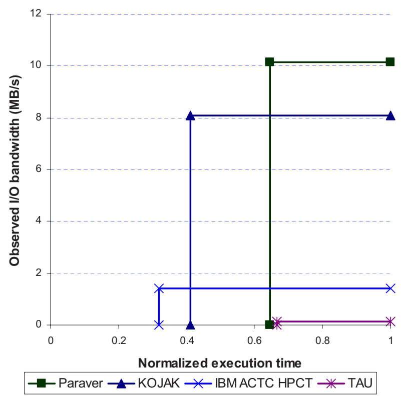


(b) Performance Data Size

Figure 4. MPI Performance Analysis Tool Applications



(a) FLASH, 1,024 Compute Nodes



(b) sPPM, 1,024 Compute Nodes

Figure 5. Performance Analysis Tool I/O Distributions

and sPPM is shown in Figure 5 as a function of time. The tracing tools use buffers in memory to save event records, but when the buffers are full it is necessary to write data to the file-system. In order to minimize the performance impact of tracing on systems like BG/L that only supports blocked I/O, it is advantageous to use large memory buffers, and delay file I/O until near the end of program execution. In our experiments, Paraver follows this approach and achieves somewhat better utilization of I/O bandwidth. The execution-time overhead for tracing could be kept to a minimum by keeping all trace data in memory, but that is not feasible for long-running applications with large numbers of events. The BG/L system that we used for these experiments did not have a parallel file system attached to it. The output files were written to an NFS file system mounted on the nodes. An aggressive parallel file-system could significantly reduce the execution-time overhead caused by trace file generation. However, the problem of managing the vast amounts of trace data would remain.

5 Discussion

Weak scaling and strong scaling are commonly used to study the scalability for a given application. For weak scaling, as the number of processors increases, the workload per processor remains the same. In other words, total workload increases with the number of processors. For strong scaling, total workload remains the same so when the number of processors increases, workload reduces for each processor.

Among the applications we studied in Section 4.2, POP is run with strong scaling configuration. As the number of processors increases, the execution overhead becomes very large. We suspect that when the workload or data for each processor is too small, it requires more communications for each processor to finish its share of the job (i.e. the communication and computation ratio increases). The application POP fails to run with Paraver and KOJAK when using 1,024 and 2,048 compute nodes. One possible reason is that in order to record numerous communication information, the memory usage per process exceeds the 512 MBytes limit on BG/L.

Due to the limitations for each compute node on BG/L (e.g., physical memory size, single thread/single process, performance counter events), many program execution behaviors are difficult to measure. For example, memory usage is only estimated by using heap and stack sizes. We expect in the next generation of Blue Gene system, Blue Gene/P, a lot of such performance measurement obstacles can be removed. For example, with multiple threads on a single compute node, we would be able to measure the I/O activities more precisely.

There are a number of ways to reduce the data volume. For example, if the application does similar work for many time-steps, one could limit tracing to just one or two time-steps, either by inserting calls to start/stop tracing within the application, or by setting run-time parameters to limit the total number of time-steps in the numerical simulation. By selectively tracing just one part of the application, the data volume can be dramatically reduced. Also, the nature of large-scale parallel applications is that many MPI processes are doing very similar work, so one can limit trace generation to a subset of MPI ranks and still obtain useful insight into the time-dependent behavior of the application. Our experience has been that good insight can be obtained by saving data from about 100 MPI processes, which can reduce the data volume by roughly a factor of 1,000 for large BG/L systems. One can imagine a number of ways to select MPI ranks. For example, one could pick a contiguous range of MPI ranks, or one could define a communication neighborhood based on actual communication patterns, or one could choose a contiguous block of processes based on the topology of the network (a three-dimensional torus for BG/L).

6 Related Work

Performance tools like Dyninst [1,19], Paradyn [7,24] provides on general framework for performance tuning for parallel application. Dyninst provides a C++ class library for program instrumentation. Using this library, it is possible to instrument and modify application programs during execution. A unique feature of this library is that it permits machine-independent binary instrumentation programs to be written. Numerous projects including TAU are utilizing this library. Paradyn, based on Dyninst, is a performance measurement tool for parallel and distributed programs. Performance instrumentation is inserted into the application program and modified during execution. The instrumentation is controlled by a Performance Consultant module. The Performance Consultant has a well-defined notion of performance bottlenecks and program structure, so that it can associate bottlenecks with specific causes and specific parts of a program. The instrumentation overhead is controlled by monitoring the cost of its data collection.

Software like pSigma [27] can utilize the tool to minimize human intervention. PSigma is an infrastructure for instrumenting parallel applications. It enables the users to probe into the execution of an application by intercepting its control-flow at selected points.

7 Conclusion

In this paper we examined existing MPI profiling and tracing tools on Blue Gene/L. The profiling tools collect only cumulative timing data, and can scale to very large systems with relatively little overhead. Tracing tools collect and analyze time-stamped events, but the trace files tend to grow to unmanageable sizes when the number of MPI processes is large. To control the volume of trace data, one can limit trace generation to a subset of MPI ranks and/or a selected time window, and still obtain useful insight into the time-dependent behavior of parallel applications. We discuss a possible method to control trace generation. This method will provide flexible control of trace generation, and can be used for more complex tasks, such as automatically detecting and logging a repeated sequence of MPI calls. In the near future we plan to implment and verify our proposed method to have a scalable MPI performance analysis tool.

Acknowledgment

The authors would like to thank Manish Gupta, David Klepacki, and Jose Castanos for their supports and reviews.

References

[1] An Application Program Interface (API) for Runtime Code Generation. [Online]. Available: http://www.dyninst.org

[2] FLASH homepage. [Online]. Available: http://flash.uchicago.edu/website/home/

[3] IBM Advanced Computing Technology Center MPI Tracer/Profiler. [Online]. Available: http://www.research.ibm.com/actc/projects/mpitracer.shtml

[4] Kit for Objective Judgement and Knowledge-based Detection of Performance Bottlenecks. [Online]. Available: http://www.fz-juelich.de/zam/kojak/

[5] MPICH-A Portable Implementation of MPI. [Online]. Available: http://www-unix.mcs.anl.gov/mpi/mpich/

[6] mpiP: Lightweight, Scalable MPI Profiling. [Online]. Available: http://www.llnl.gov/CASC/mpip/

[7] Paradyn:Parallel Performance Tools. [Online]. Available: http://www.paradyn.org

[8] PARAVER homepage. [Online]. Available: http://www.cepba.upc.es/paraver/

[9] TAU: Tuning and Analysis Utilities. [Online]. Available: http://www.cs.uoregon.edu/research/tau/home.php

[10] The ASCI sPPM Benchmark Code. [Online]. Available: http://www.llnl.gov/asci_benchmarks/asci/limited/ppm/asci_sppm.html

[11] The ASCI sweep3d Benchmark Code. [Online]. Available: http://www.llnl.gov/asci_benchmarks/asci/limited/sweep3d/asci_sweep3d.html

[12] The Parallel Ocean Program (POP). [Online]. Available: http://climate.lanl.gov/Models/POP/

[13] N. R. Adiga, M. A. Blumrich, D. Chen, P. Coteus, A. Gara, M. E. Giampapa, P. Heidelberger, S. Singh, B. D. Steinmacher-Burow, T. Takken, M. Tsao, , and P. Vranas, "Blue Gene/L Torus Interconnection Network," *IBM Journal of Research and Development*, vol. 49, no. 2/3, pp. 265–276, 2005.

[14] N. Adiga, G. Almasi, G. Almasi, Y. Aridor, R. Barik, D. Beece, R. Bellofatto, G. Bhanot, R. Bickford, M. Blumrich, A. Bright, J. Brunheroto, C. Cacaval, J. Castaos, W. Chan, L. Ceze, P. Coteus, S. Chatterjee, D. Chen, G. Chiu, T. Cipolla, P. Crumley, K. Desai, A. Deutsch, T. Domany, M. Dombrowa, W. Donath, M. Eleftheriou, C. Erway, J. Esch, B. Fitch, J. Gagliano, A. Gara, R. Garg, R. Germain, M. Giampapa, B. Gopalsamy, J. Gunnels, M. Gupta, F. Gustavson, S. Hall, R. Haring, D. Heidel, P. Heidelberger, L. Herger, D. Hoenicke, R. Jackson, T. Jamal-Eddine, G. Kopcsay, E. Krevat, M. Kurhekar, A. Lanzetta, D. Lieber, L. Liu, M. Lu, M. Mendell, A. Misra, Y. Moatti, L. Mok, J. Moreira, B. Nathanson, M. Newton, M. Ohmacht, A. Oliner, V. Pandit, R. Pudota, R. Rand, R. Regan, B. Rubin, A. Ruehli, S. Rus, R. Sahoo, A. Sanomiya, E. Schenfeld, M. Sharma, E. Shmueli, S. Singh, P. Song, V. Srinivasan, B. Steinmacher-Burow, K. Strauss, C. Surovic, R. Swetz, T. Takken, R. Tremaine, M. Tsao, A. Umamaheshwaran, P. Verma, P. Vranas, T. Ward, M. Wazlowski, W. Barrett, C. Engel, B. Drehmel, B. Hilgart, D. Hill, F. Kasemkhani, D. Krolak, C. Li, T. Liebsch, J. Marcella, A. Muff, A. Okomo, M. Rouse, A. Schram, M. Tubbs, G. Ulsh, C. Wait, J. Wittrup, M. Bae, K. Dockser, L. Kissel, M. Seager, J. Vetter, and K. Yates, "An Overview of the Blue Gene/L Supercomputer," in *Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing.* IEEE Computer Society Press, 2002, pp. 1–22.

[15] G. Almasi, C. Archer, J. G. Castanos, J. A. Gunnels, C. C. Erway, P. Heidelberger, X. Martorell, J. E. Moreira, K. Pinnow, J. Ratterman, B. D. Steinmacher-Burow, W. Gropp, and B. Toonen, "Design and implementation of message-passing services for the Blue Gene/L supercomputer," *IBM Journal of Research and Development*, vol. 49, no. 2/3, p. 393, 2005.

[16] G. S. Almasi, D. Beece, R. Bellofatto, G. Bhanot, R. Bickford, M. Blumrich, A. A. Bright, J. Brunheroto, C. Cascaval, J. Castanos, L. Ceze, P. Coteus, S. Chatterjee, D. Chen, G. Chiu, T. M. Cipolla, P. Crumley, A. Deutsch, M. B. Dombrowa, W. Donath, M. Eleftheriou, B. Fitch, J. Gagliano, A. Gara, R. Germain, M. E. Giampapa, M. Gupta, F. Gustavson, S. Hall, R. A. Haring, D. Heidel, P. Heidelberger, L. M. Herger, D. Hoenicke, R. D. Jackson, T. Jamal-Eddine, G. V. Kopcsay, A. P. Lanzetta, D. Lieber, M. Lu, M. Mendell, L. Mok, J. Moreira, B. J. Nathanson, M. Newton, M. Ohmacht, R. Rand, R. Regan, R. Sahoo, A. Sanomiya, E. Schenfeld, S. Singh, P. Song, B. D. Steinmacher-Burow, K. Strauss, R. Swetz, T. Takken, P. Vranas, T. J. C. Ward, J. Brown, T. Liebsch, A. Schram, , and G. Ulsh, "Cellular Supercomputing with System-on-a-Chip," in *IEEE International Solid-State Circuits Conference (ISSCC)*, 2002, pp. 152–153.

[17] G. Almasi, C. Archer, J. G. Castanos, M. Gupta, X. Martorell, J. E. Moreira, W. Gropp, S. Rus, and B. Toonen, "MPI on Blue Gene/L: Designing an Efficient General Purpose Messaging Solution for a Large Cellular System," in *Proceedings of the 10th European PVM/MPI Users Group Meeting*, 2003, pp. 252–261.

[18] G. Almasi, G. Bhanot, A. Gara, M. Gupta, J. Sexton, B. Walkup, V. V. Bulatov, A. W. Cook, B. R. de Supinski, J. N. Glosli, J. A. Greenough, F. Gygi, A. Kubota, S. Louis, T. E. Spelce, F. H. Streitz, P. L. Williams, R. K. Yates, C. Archer, J. Moreira, and C. Rendleman, "Scaling physics and material science applications on a massively parallel blue gene/l system," in *ICS '05: Proceedings of the 19th annual international conference on Supercomputing.* New York, NY, USA: ACM Press, 2005, pp. 246–252.

[19] B. R. Buck and J. K. Hollingsworth, "An API for Runtime Code Patching," *Journal of High Performance Computing Applications*, vol. 14, pp. 317–329, 2000.

[20] J. K. Dukowicz and R. D. Smith, "Implicit free-surface method for the Bryan-Cox-Semtner ocean model," *Journal of Geophysics Research*, vol. 99, pp. 7991–8014, Apr. 1994.

[21] J. K. Dukowicz, R. D. Smith, and R. C. Malone, "A Reformulation and Implementation of the Bryan-Cox-Semtner Ocean Model on the Connection Machine," *Journal of Atmospheric and Oceanic Technology*, vol. 10, no. 2, pp. 195–208, Apr. 1993.

[22] D. J. Kerbyson, H. J. Alme, A. Hoisie, F. Petrini, H. J. Wasserman, and M. Gittings, "Predictive performance and scalability modeling of a large-scale application," in *Supercomputing '01: Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM).* ACM Press, 2001, pp. 37–37.

[23] A. D. Malony, S. Shende, R. Bell, K. Li, L. Li, and N. Trebon, "Advances in the tau performance system," pp. 129–144, 2004.

[24] B. P. Miller, M. D. Callaghan, J. M. Cargille, J. K. Hollingsworth, R. B. Irvin, K. L. Karavanic, K. Kunchithapadam, and T. Newhall, "The Paradyn Parallel Performance Measurement Tool," *IEEE Computer*, vol. 28, pp. 37–46, Nov. 1995.

[25] B. Mohr and F. Wolf, "KOJAK - A Tool Set for Automatic Performance Analysis of Parallel Applications ," in *Euro-Par 2003: Proceedings of the International Conference on Parallel and Distributed Computing*, Sept. 2003.

[26] V. Pillet, J. Labarta, T. Cortes, and S. Girona, "PARAVER: A tool to visualise and analyze parallel code," in *Proceedings of WoTUG-18: Transputer and occam Developments*, vol. 44. Amsterdam: IOS Press, 1995, pp. 17–31.

[27] S. Sbaraglia, K. Ekanadham, S. Crea, and S. Seelam, "pSigma: An Infrastructure for Parallel Application Performance Analysis using Symbolic Specifications," in *The sixth European Workshop on OpenMP - EWOMP'04*, Oct. 2004.

[28] R. D. Smith, J. K. Dukowicz, and R. C. Malone, "Parallel ocean general circulation modeling," in *Proceedings of the eleventh annual international conference of the Center for Nonlinear Studies on Experimental mathematics : computational issues in nonlinear science.* Elsevier North-Holland, Inc., 1992, pp. 38–61.

[29] G. A. Sod, "A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws," *Journal of Computational Physics*, vol. 27, pp. 1–31, Apr. 1978.