Preprint
UCRL-JC-142829

An Integrated Performance Visualizer for MPI/OpenMP Programs

J. Hoeflinger, B. Kuhn, P. Petersen, H. Rajic, S. Shah, J. Vetter, M. Voss, R. Woo

This article was submitted to
Workshop on OpenMP Applications and Tools, West Lafayette, IN, July 30-31, 2001

February 25, 2001

U.S. Department of Energy
Lawrence Livermore National Laboratory

This work was performed under the auspices of the United States Department of Energy by the University of California, Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

An Integrated Performance Visualizer for MPI/OpenMP Programs*

Jay Hoeflinger † Bob Kuhn † Paul Petersen † Hrabri Rajic †
Sanjiv Shah † Jeff Vetter ‡ Michael Voss † Renee Woo †

† KAI Software
Intel Americas, Inc.
Champaign, Illinois 61820

‡ Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, California 94551

Abstract

1 Introduction

Cluster computing has emerged as a defacto standard in parallel computing over the last decade. Now, researchers have begun to use clustered, shared-memory multiprocessors (SMPs) to attack some of the largest and most complex scientific calculations in the world today [2, 1], running them on the world's largest machines including the US DOE ASCI platforms: Red, Blue Mountain, Blue Pacific, and White.

MPI has been the predominant programming model for clusters [3]; however, as users move to "wider" SMPs, the combination of MPI and threads has a "natural fit" to the underlying system design: use MPI for managing parallelism between SMPs and threads for parallelism within one SMP.

OpenMP is emerging as a leading contender for managing parallelism within an SMP. OpenMP and MPI offer their users very different characteristics. Developed for different memory models, they fill diametrically opposed needs for parallel programming. OpenMP was made for shared memory systems, while MPI was made for distributed memory systems. OpenMP was designed for explicit parallelism and implicit data movement, while MPI was designed for explicit data movement and implicit parallelism. This difference in focus gives the two parallel programming frameworks very different usage characteristics. But these complementary usage characteristics make the two frameworks perfect for handling the two different parallel environments presented by cluster computing: shared memory within a box and distributed memory between the boxes.

Unfortunately, simply writing OpenMP and MPI code does not guarantee efficient use of the underlying cluster hardware. What is more, existing tools only provide performance information about either MPI or OpenMP, but not both. This lack of integration prevents users from understanding the critical path for performance in their application. This integration also helps users adjust their expectations of performance for their application's software design. Once the user decides to investigate their application's performance, they need detailed information about the expense of operations in their application. Most likely, message passing activity and OpenMP regions are related to these expensive operations. Viewed in this light, the user needs a performance analyzer to understand the complex interactions of MPI and OpenMP. For message passing codes, several performance analysis tools exist: Vampir, TimeScan, Paragraph, and others [make citations]. For OpenMP codes there is GuideView and a few other proprietary tools from other vendors [make citations]. However, in practice, there is little production quality support for the combination of MPI and OpenMP.

As a result, KAI Software has partnered with the Department of Energy through an ASCI Pathforward contract to develop a tool called Vampir/GuideView, or VGV. This tool combines the capabilities of Vampir and GuideView, into

*This work was performed under the auspices of the U.S. Dept. of Energy by University of California LLNL under contract W-7405-Eng-48. Authors may be contacted at: {jay.p.hoeflinger, bob.kuhn, paul.m.petersen, hrabri.rajic, sanjiv.shah, michael.voss, renee.woo}@intel.com and vetter@llnl.gov.

one tightly-integrated performance analysis tool. From the outset, its design targets performance analysis on systems with thousands of processors.

The purpose of this paper is to describe this tool, how it may be used, and how it can help pin-point the source of performance problems in MPI/OpenMP programs.

2 Goals of the Project

The main goals of the VGV project are to create an integrated MPI/OpenMP performance analysis tool that is easy to use and that scales well to even the largest systems currently available. This new tool is largely based on the existing Vampir and GuideView tools.

2.1 Scalability

A performance analysis tool faces new problems when it is used for systems with thousands of processors. If the tool is not careful, the amount of information gathered about the performance of a program can get very large, filling disks or causing large data transfer times. The amount of information displayed on-screen can overwhelm the user if it is not displayed appropriately, and on-screen display space is limited, anyway. The aggressive goal of the VGV project is to quadruple the number of processors that can be analyzed every year for the next two years. This year VGV can handle 1000 processors.

2.2 Integration

To perform effective performance analysis with VGV, there must be an integration of information from Vampir and GuideView. This not only avoids the work of manually coordinating output from the two tools, but also provides a platform for synthesizing an overall performance report. The performance data of both tools should also be integrated with source code information.

2.3 Effective Data Presentation

VGV should present an interface which makes the experience of using it for solving performance problems on large machines not materially different from solving such problems on small systems. The tool should also be able to draw the user's attention to potential performance problems, and help the user locate the source of those problems in the program.

3 Using MPI with OpenMP

Before describing how VGV intends to meet its goals, we will briefly mention some key issues that must be addressed when using MPI with OpenMP. MPI may be used with OpenMP, but they have no knowledge of each other, so a few basic rules must be followed to ensure that they do not interfere with each other.

In general, MPI implementations are not thread-safe, so MPI functions can not be safely used when more than one OpenMP thread is active. Therefore, calls to MPI functions should be done either outside OpenMP parallel regions, as shown in Figure 1, or inside a region in which only one thread is active, such as a `MASTER` region or a `SINGLE` region, as shown in Figure 2.

Reviewers note: In the full paper, we will address other MPI/OpenMP issues.

4 Structure of the Tool

The flow of the integrated tool follows 4 steps:

1. instrumenting the program at compile time,
2. generating an integrated MPI/OpenMP trace file at runtime,

```
      CALL MPI_SEND(A(1), N, MPI_REAL, 1, tag, comm, ierr)
      CALL MPI_RECV(A(1), N, MPI_REAL, 0, tag, comm, status, ierr)
!$omp parallel do shared(A, B, N)
      DO I=1,N
         B(I) = F(A(I))
      END DO
```

Figure 1: Example of using MPI to exchange data outside an OpenMP parallel region.

```
!$omp parallel shared(A, B, N)
!$omp do
      DO I=1,N
         B(I) = F(A(I))
      END DO
!$omp master
      CALL MPI_ALLREDUCE(A, RA, N, MPI_REAL, MPI_SUM, 0, comm)
!$omp end master
!$omp barrier !  to insure consistent memory
!$omp do
      ...
!$omp end parallel
```

Figure 2: Example of using MPI to do a reduction operation inside an OpenMP parallel region.

3. post-run performance analysis for MPI with Vampir

4. analyzing OpenMP performance with GuideView.

This design integrates Vampirtrace and Vampir with the OpenMP components, Guide, the Guide Runtime Library, and GuideView.

Like most MPI performance analysis tools, Vampirtrace uses the MPI library wrapper interface for instrumentation. As each MPI call is performed, an event is written to a trace file. Vampir is the post-run trace file analysis tool.

Guide is a portable OpenMP compiler for Fortran and C++ that restructures source code and inserts calls to the Guide Runtime Library. The Guide Runtime Library layers on top of threads to implement OpenMP functions. The program is instrumented to call clock timers at all the significant OpenMP events. At the end of a run, the information gathered from these timers is written into a statistics file.

The heart of the MPI and OpenMP integration occurs at runtime. The instrumentation of OpenMP and MPI requires coordination. This is achieved by adding OpenMP events to the Vampirtrace API. The Guide Runtime Library is modified to instrument interesting OpenMP events. For each interesting OpenMP event, the timers are put into a data structure that is time stamped and sent to the log file via the new API.

5 Usage of the Tool

Once an integrated MPI/OpenMP trace file has been created during the application run, it can be viewed by an integrated user interface. Vampir shows the tracefile events ordered by time in the timeline display. When an MPI process executes an OpenMP region, a wiggle or glyph appears at the top of that process' time line. The user can select that glyph to view that OpenMP region or can select a set of MPI processes or a time line section for OpenMP analysis.

OpenMP analysis aggregates the OpenMP data structures from all the tracefile events in the selection. Then the aggregated data is written to a file where a GuideView server process reads the file.

GuideView displays the OpenMP regions for each MPI process as a separate set of OpenMP data. In this way, the user can use GuideView tools to select a subset of the hundreds of MPI processes that may be running and sort by any OpenMP performance measure. Examples of OpenMP performance measures for sorting are: scheduling imbalance,

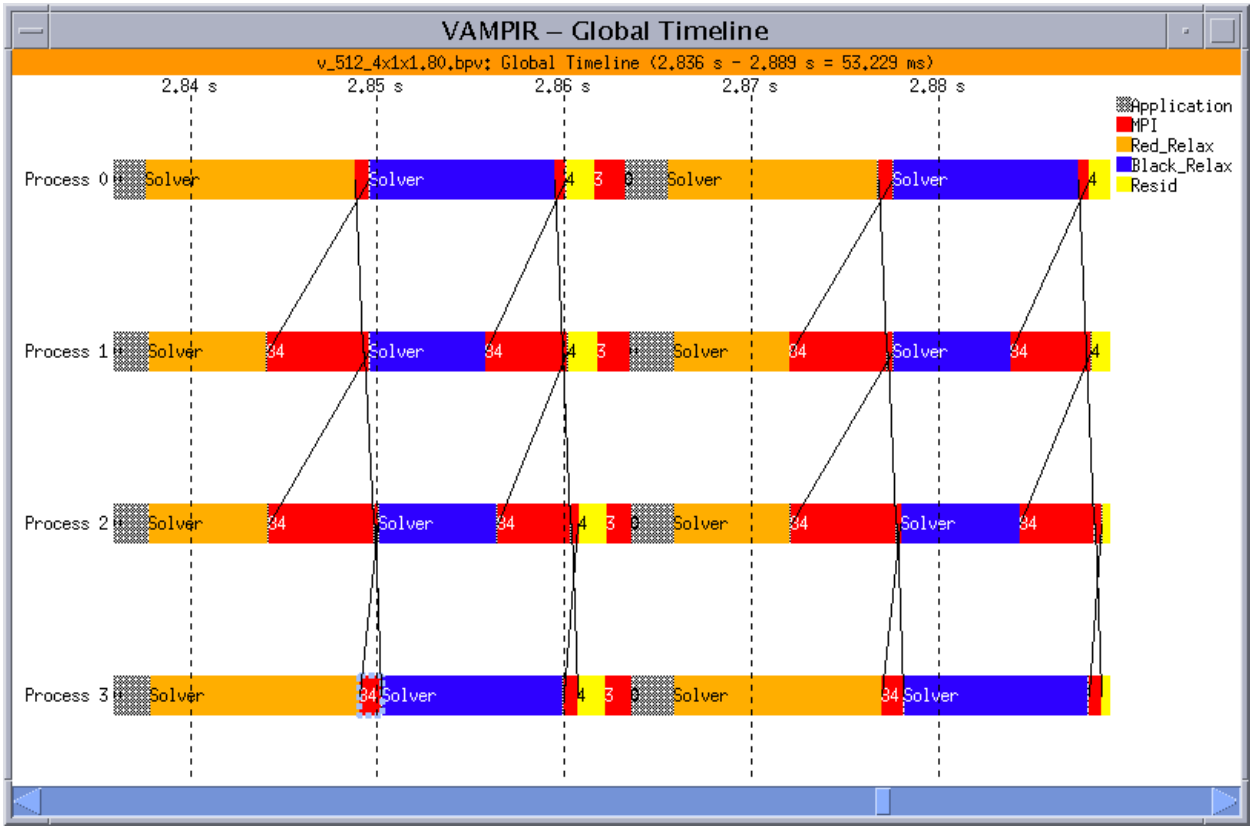

Figure 3: Vampir display of MPI-only code, showing load imbalance.

lock time, time spent in a locked region, and overhead. The user can specify that GuideView show the top or bottom n, where the user specifies n. This mechanism allows a user to compose compound performance queries by sorting on one criteria, filtering the top responders, and then sorting by another criteria.

The user can also view the subroutine profile for one or a selection of MPI processes within GuideView. This can be viewed as inclusive to allow the user to understand the call tree structure, or exclusive to understand which subroutines consume the most time.

One of the important uses of the tool is to locate regions of the program where some processors spend much time waiting while others are doing useful work. This is referred to as a load-imbalance. From the color-coded display, the user can determine how much time each processor spends waiting.

A key use of VGV is source code browsing. The source code associated with any part of the performance data may be brought up in a browsing window by clicking the mouse on the data display.

6 Finding Performance Problems with VGV

Reviewers note: in the final paper this section will be significantly expanded.

In Figure 3, the Vampir display for an MPI-only version of a code is shown. Vampir clearly shows a load imbalance between the processors (red-colored bars on a user's screen). The processes that have smaller regions finish their computation early and enter communication, effectively becoming idle until the other MPI processes finish their domain.

In Figure 4, the GuideView display shows the an OpenMP-only version of the same code is able to load balance quite effective.

When you combine both MPI and OpenMP (Figure 5), Vampir shows almost solid computation (almost no red-

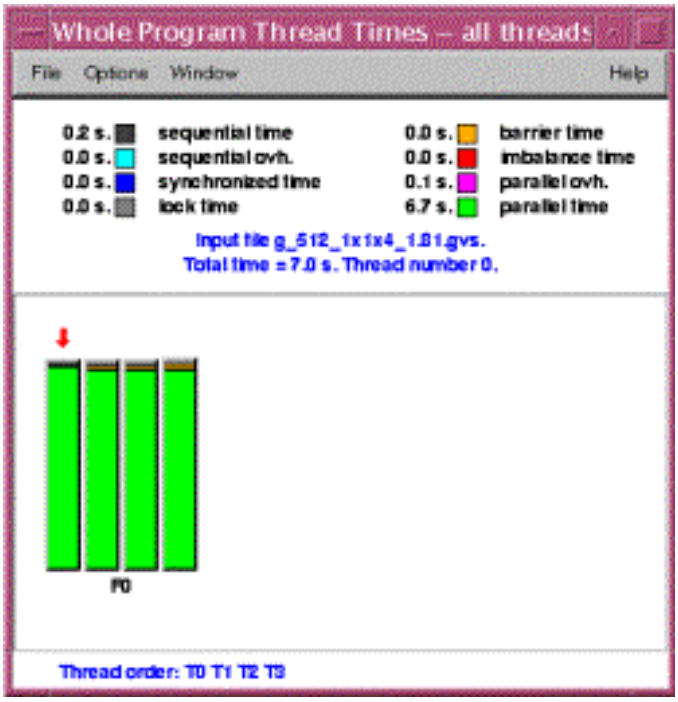

Figure 4: GuideView display of OpenMP-only code, showing load balanced.

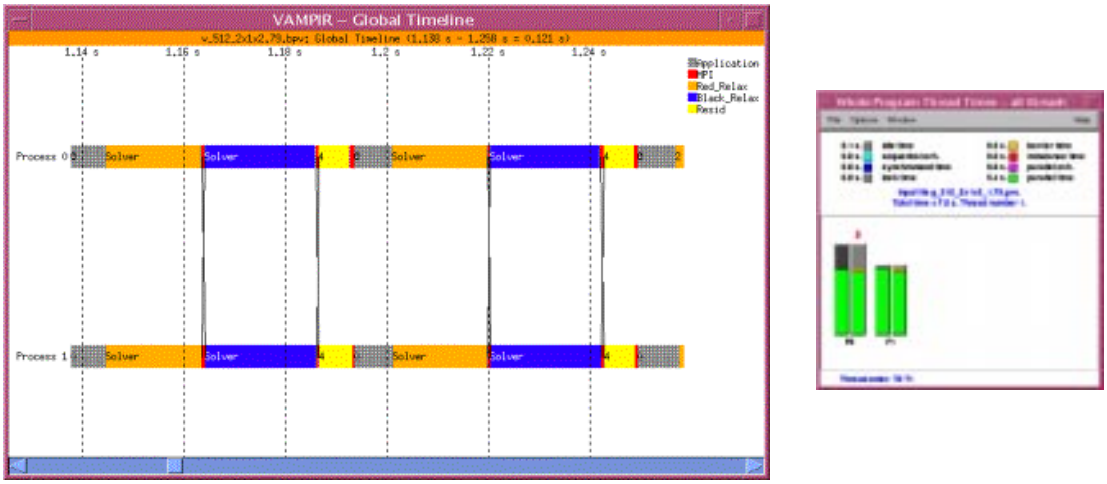

Figure 5: VGV display of MPI/OpenMP code, load is balanced by OpenMP.

colored bars), with little communication delay. The GuideView window shows that the OpenMP level is also very efficient in this version. Finally, notice that the master MPI process is performing more work, which shows up in GuideView.

7 Scalability of the Tool

Prior to this project, GuideView already used light-weight summarization techniques to analyze performance statistics for the OpenMP processors. Vampir, on the other, wrote trace records to a single tracefile for every MPI call. This produces a potentially very large trace file that must not only be stored, but also completely read and analyzed to provide the user with a display.

To be scalable, VGV must adequately address the following issues:

- the disk storage requirements of an event-based tracing tool could become enormous for long runs with large numbers of processors,
- workstation screens have limited space for displaying performance information,
- simply finding a potential performance problem may be very hard in the blizzard of information potentially generated from a massive run.

Some of these issues have already been addressed in the current version of VGV. Others will be implemented during the remainder of the project.

VGV will attempt to reduce the size of the trace file through several means:

event compression - Specialized trace records can be used for some events and encoded to save space. Collective communication events, which usually require a full trace record for every process can be reduced to a single trace record and a series of small records, one per task. Also, source code line numbers can be encoded to save space in each trace record.

event combination - Events occurring commonly together can be replaced by a single event. Very short events which are issued until the MPI state changes (e.g. MPI_IProbe / MPI_Test) can be replaced by a single event covering the entire interaction.

event summarization - Some events can be summarized by maintaining only min/max/average values and discarding the events.

structured trace file - The single trace file can be replaced by multiple, hierarchically structured files. This also saves processing time because a top-level summary file can processed much more quickly than can the whole original trace file. This allows the user to see a summary then drill down to other levels in the hierarchy for display.

tracing/instrumentation control - Tracing can be disabled or enabled according to a variety of criteria. The user could place enable/disable trace calls in the code, or could select specific events to enable/disable, or could trace only certain MPI processes, or a variety of other criteria.

The on-screen presentation of the performance information can be made scalable through vertical scrolling of MPI process time-line information, as well as back-to-front stacking of time-lines.

VGV will use data reduction to attempt to identify potential performance problems for the user. Statistical analysis of the data for a single interval of the user's program can identify processes or processors that require unusual (high or low) amounts of various resources (e.g., cache misses, time, memory access time), and mark them for the user.

We have found that the execution time of the analysis tool can be a major fraction of the time required for the tool. For this reason, the tool will be partitioned into a display component (DC) and a trace processing component (TPC). The TPC can be parallelized, and run on a small number of processors. The DC can be potentially multi-threaded.

8 Conclusion

Vampir/GuideView is intended to be a flexible, easy-to-use tool for finding performance problems in programs written with a combination of MPI and OpenMP, that run for extremely long times and use thousands of processors. We know of no other tool targeted at MPI/OpenMP, and certainly none with the ability to handle such massive runs. During the remaining two years of its development for the ASCI project, we believe that it will become a tool that can be used for the largest ASCI clusters, and will help users pin-point performance anomalies in their codes.

References

[1] A.C. Calder, B.C. Curtis, and et al. High-Performance Reactive Fluid Flow Simulations Using Adaptive Mesh Refinement on Thousands of Processors. In *Supercomputing 2000: High Performance Networking and Computing Conference*, 2000. electronic pub.

[2] A.A. Mirin, R.H. Cohen, and et al. Very High Resolution Simulation of Compressible Turbulence on the IBM-SP System. In *Supercomputing '99: High Performance Networking and Computing Conference*, 1999. electronic pub.

[3] G.F. Pfister. *In Search of Clusters: The Coming Battle in Lowly Parallel Computing*. Prentice Hall, Upper Saddle River, NJ, 1995.