UCRL-TR-209289

High Performance Tools & Technologies

Michael Collette, Bob Corey, and John Johnson

December, 2004

U.S. Department of Energy

Lawrence
Livermore
National
Laboratory

DISCLAIMER

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

Computing Applications and Research Department
Lawrence Livermore National Laboratory

High Performance Tools & Technologies

December, 2004

By

Michael Collette, Bob Corey, and John Johnson

with contributions from

Blaise Barney and Shawn Dawson

High Performance Tools & Technologies*,* a report prepared as a deliverable for a FY2004 Computing Applications and Research Department (CAR) Technology-Based funded activity. The authors of this document are Michael Collette, Bob Corey, and John Johnson.

Table of Contents

Chapter 1: Overview And Findings ***1***

1.1 Executive Summary **1**
1.1.1 Today's High Performance Computing Environment 1
1.1.2 Observations 1
1.1.3 Recommendations 2

1.2 Outline **2**

1.3 Introduction **2**

1.4 Disclaimer **4**

1.5 Analysis Preview **4**

1.6 Programs' / Applications' Current & Future Needs **5**

1.7 Emerging Architectures **5**
1.7.1 Streaming Data-driven Architectures 6
1.7.2 PIM 6
1.7.3 Reconfigurable HW 7
1.7.4 Windows Server (Compute Cluster Edition) 7

1.8 Development Tools **7**
1.8.1 Debuggers 8
1.8.2 Memory Checkers 9
1.8.3 Performance Analysis Tools 9

1.9 Communication & Networking **11**

1.10 Visualization **11**

1.11 Future Tools **12**

1.12 Gaps **12**

1.13 Conclusions **13**

Chapter 2: Programs / Applications Current & Future Needs ***15***
2.1 Engineering 15
2.1.1 ParaDyn 15
2.1.2 Diablo 15
2.1.3 EMSolve 16
2.2 Energy and Environment 16
2.2.1 Atmospheric Science 16
2.2.2 NARAC 17
2.2.3 GNEM 17
2.2.4 ESG 18
2.3 DNT 18
2.3.1 ARES 18
2.3.2 ALE3D 19
2.3.3 KULL 19
2.4 NAI 19
2.4.1 JCATS 19

Chapter 3: Current Generation Computer Systems ***20***
3.1 BlueGene/L (IBM / LLNL) 20
3.2 Purple (IBM / LLNL) 20
3.3 Q (HP / LANL) 20
3.4 Red Storm (Cray / SNL) 20
3.5 X-1 (Cray / ORNL) 21
3.6 Earth Simulator (NEC / Japan) 21

3.7 Linux Clusters (Various Developers / Various Sites) 21
3.8 System X (Apple / Virginia Tech) 22
3.9 Columbia (SGI / NASA) 22
3.10 MareNostrum (IBM / Spain) 22

Chapter 4: Parallel Code Development Tools 23

4.1 Debuggers 23
4.1.1 TotalView 23
4.1.2 DDT 24
4.1.3 Ladebug 25
4.1.4 PGDBG 25
4.1.5 GDB / DDD 26
4.1.6 PDBX 26
4.1.7 IDB 26
4.1.8 Guard 27
4.1.9 printf 27
4.1.10 Other parallel code development aides 27

4.2 Memory Checkers 28
4.2.1 Third Degree 28
4.2.2 Zero Fault 28
4.2.3 Valgrind 28
4.2.4 Insure++ 29
4.2.5 Purify 29
4.2.6 SmartHeap 29
4.2.7 Great Circle (Application Saver) 30
4.2.8 Electric Fence 30
4.2.9 MemUsage 30
4.2.10 Mpatrol 31
4.2.11 Other memory checkers 32

4.3 Performance Analysis Tools: APIs 32
4.3.1 Dyninst 32
4.3.2 DPCL 33
4.3.3 DCPI 33
4.3.4 PAPI 33
4.3.5 PMAPI 33

4.4 Performance Analysis Tools: Profiling Toolkits 34
4.4.1 PE Benchmarker 34
4.4.2 HPM Toolkit 35
4.4.3 HPCToolkit 37
4.4.4 PerfSuite 37
4.4.5 TAU 37
4.4.6 SpeedShop 39
4.4.7 AIMS 40
4.4.8 KOJAK 40

4.5 Performance Analysis Tools: Profilers 41
4.5.1 Dynaprof 41
4.5.2 mpiP / ToolGear 41
4.5.3 MPX / ToolGear 42
4.5.4 VTune 43
4.5.5 prof & gprof 44
4.5.6 Xprofiler 45
4.5.7 DEEP/MPI 46
4.5.8 VProf 47
4.5.9 Hiprof 47
4.5.10 Pixie 48
4.5.11 Perfex 49

4.5.12 PapiEx 49
4.5.13 Tprof 49
4.5.14 SCALEA 50
4.5.15 Pgprof 50
4.5.16 IPM 50

4.6 Performance Analysis Tools: Tracers 51
4.6.1 RootCause 51
4.6.2 Mpitrace 51
4.6.3 Perfometer 51
4.6.4 SiGMA 51

4.7 Performance Analysis Tools: Visualizers & Other Analyzers 52
4.7.1 Vampir / Vampirtrace / VampirGuideView (VGV) 52
4.7.2 Paraver 53
4.7.3 Jumpshot & MPE 54
4.7.4 Paradyn 55
4.7.5 SvPablo 56
4.7.6 ParaGraph 57
4.7.7 Opt 58
4.7.8 OptiPath 58
4.7.9 SeeWithin/Pro 58
4.7.10 Xmpi 58
4.7.11 Dynamic Kappa-Pi 58
4.7.12 Other analysis tools 59

Chapter 5: Communication & Networking 60

5.1 Communication Libraries 60
5.1.1 MPI (& MPI-2) 60
5.1.2 PVM 60
5.1.3 OpenMP (& POSIX threads) 61

5.2 Distributed Computing 61
5.2.1 Globus 61
5.2.2 Condor 62

Chapter 6: Visualization Tools 63
6.1 VisIt 63
6.2 EnSight 64
6.3 ParaView 64
6.4 Tecplot 65
6.5 GRIZ 65
6.6 Maya 65
6.7 Chromium 66

Chapter 7: Other Parallel Tools 67
7.1 UPC 67
7.2 Parallel Matlab 67
7.3 AMPI 68
7.4 JXTA 68
7.5 SOAP 69
7.6 EJB 69
7.7 Eclipse System 69

Chapter 8: Postscript 71
8.1 Omissions & Expectations 71
8.2 Thanks 71
8.3 References 72
8.3.1 Product Information Homepage Links 73

Appendix: Tools Table 77

Chapter 1: Overview And Findings

1.1 Executive Summary

This goal of this project was to evaluate the capability and limits of current scientific simulation development tools and technologies with specific focus on their suitability for use with the next generation of scientific parallel applications and High Performance Computing (HPC) platforms. The opinions expressed in this document are those of the authors, and reflect the authors' current understanding and functionality of the many tools investigated.

As a deliverable for this effort, we are presenting this report describing our findings along with an associated spreadsheet outlining current capabilities and characteristics of leading and emerging tools in the high performance computing arena.

1.1.1 Today's High Performance Computing Environment

Most current applications at LLNL were not designed for high performance computing but rather were retrofitted -- often in an ad-hoc fashion. Many were not originally for multi-processor computing paradigms and have not been designed or tested for scaling. Most are tied to particular multi-processing technologies (e.g. MPI) which have been grafted onto existing single processor codes. Some projects have embraced high performance technologies as core to their architecture and there seems to be a trend for newer high performance applications to incorporate advanced technologies into their design. The majority of applications at LLNL are run on commodity-based MPP systems.

Tools used by application developers are widely varying and ad-hoc, and like their application counterparts, often not designed for high performance computing. There is a plethora of tools available with significant overlap among them. Most tools are platform specific and there are few that span the architectures of interest to the high performance computing community.

1.1.2 Observations

There is a perception that there is a dearth of tools to support high performance computing. However, there are in actuality an enormous number of tools that can be used for aspects of high performance computing. Therefore, *either the existing tools do not address the needs of the high performance computing community or the HPC community is unaware of these tools*. It turns out that both are true. Current tools typically only provide a partial solution and high performance application developers only have access to a handful of what is available.

Today's technologies focus on either flexibility (interoperability) or high speed. However, relatively few address both needs. Next generation applications will need technologies that provide both high performance and a high degree of flexibility.

1.1.3 Recommendations

Recommendations that naturally follow from this review are:

- development of a tools testbed where developers can try out new tools
- investment into efforts to port the most important tools to multiple platforms
- creation of a central repository of information about tools & technologies
- investigate efforts to combine work in high-speed technology and flexible interoperability.

1.2 Outline

This first chapter summarizes our findings (which are detailed in the other chapters) and presents our conclusions, remarks, and anticipations for the future. In the second chapter, we detail how various teams in our local high performance community utilize HPC tools and technologies, and mention some common concerns they have about them. In the third chapter, we review the platforms currently or potentially available to utilize these tools and technologies on to help in software development. Subsequent chapters attempt to provide an exhaustive overview of the available parallel software development tools and technologies, including their strong and weak points and future concerns. We categorize them as debuggers, memory checkers, performance analysis tools, communication libraries, data visualization programs, and other parallel development aides. The last chapter contains our closing information. Included with this paper at the end is a table of the discussed development tools and their operational environment.

1.3 Introduction

The goal of this report is to identify trends in next generation high performance computing and assess current tools and technologies with a focus on their potential future applicability. To do this requires a prediction as to the types of high performance applications that will be running in the future and an estimate of what their needs will be for tools and technologies. To make rough predictions on next generation applications, individuals in a variety of programs across LLNL were informally interviewed to capture where they see their computational needs moving, discover what tools and technologies are currently being used, and identify gaps. In addition, public documents and publications that were relevant to tools and technologies were also used.

From these informal interviews, it became clear that there is an increasing diversity of LLNL programs that will require high performance computing applications in the coming decade. These applications will span the scientific competencies and national and homeland security efforts in each directorate of LLNL. Some of these programs will be new to high performance computing while others are current leaders in developing today's high performance capability. The diversity of programmatic needs will require dramatic new architectural approaches and, in some cases, this will generate new approaches to high performance computing. In addition to the traditional massively parallel or vector-based numerical simulation codes, there are driving needs for computing capability in discrete and integer-based applications. There are applications that must handle massive amounts of streaming data, integrate many distributed

computational, data and other resources, or rapidly surge from small-scale computing (i.e. single processor) to massively parallel computing.

There are many technology paths leading to next generation systems: increases in massive parallelism beyond the 130K processor BlueGene/L system, implementations of alternative computing paradigms (e.g. MIMD) on massively parallel machines, streaming architectures (from the processor to the network), flexible distributed computing based on Web services, vector architectures, systems-on-a-chip, and polymorphous computing. For the most part, all of these trends will persist in some form over the next ten years (many are already becoming commonplace) and will have programmatic supporters.

On the tools side, it is much less clear what the future holds, although there are increasing demands on the tools community to proactively support next generation computing. Currently there is an amazing wealth of tools available with new tools constantly appearing. Ironically, however, there is a perception that there is a dearth of satisfactory tools to support current needs. This leads to the question: *Are application developers unaware of these tools, or do the tools not satisfy the developers' needs?* The answer is both. Application developers either do not know about or do not have access to new tools and when they do, the tools typically only address a small aspect of their needs.

The next generation of capability machines available to LLNL projects will scale to petaflop performance consisting of tens, even hundreds of thousands of processors. As we have seen in the past, we can also expect future machines to span several different architectures and operating systems. In order to achieve high performance, our applications will have to be highly scalable, and so will our development tools and techniques. However, debugging and optimizing massively parallel applications is quite challenging, and users are reluctant to learn new tools of dubious usefulness or short-lived applicability for each new platform. This raises the following questions: *Will our currently employed runtime development tools be sufficient for our future needs? What functionality is lacking? What are the hardware, OS, and scaling limitations of our current development tools, and will future platforms support them? What are future platforms going to be like? What new types of tools will be required? What other tools are available or in development that will support future platforms?* We shall examine these issues by analyzing our current tools, noting their deficiencies from the users' perspective, examining our users' current practices and needs, and seeing what other tools might be applicable. We will also examine the available and emerging architectures which future platforms may be based upon.

Advances in communication technology in the last decade and the ubiquity of the Internet have driven many application areas to pioneer distributed high performance computing solutions that span multiple geographically separated centers and integrate decentralized data sources. The main challenge, yet to be satisfactorily addressed, has been achieving high-speed computation given ever-present physical latencies and administrative barriers. Commodity technology approaches have promoted interoperability over speed of communication as can be seen by the success of TCP/IP-based Web services. For complex loosely coupled systems with small data transfers, this approach works fine but it does not scale to massive tightly coupled computations. There are only a handful of tools to support this mode of computing and the primary challenge for the next generation will be debuggers and monitoring and analysis tools.

Message passing and the networking software technologies that handle data across an ever-growing number of nodes and processors is another aspect to high performance computing that we also examine. Not only can the communication library that developers choose affect the data-handling paradigm implemented into their code, so can the networking software and any automatic parallelization tools they may use. We will review such tools and technologies that influence code development.

Most performance benchmarks are based on highly compute-bound applications that require little in the way of network traffic. Performance is usually measured in MFLOPS. Much of the work performed on new architectures is done with large application codes performing highly compute-intensive tasks such as lengthy numerical calculations for physics or engineering simulations, but which also have significant message passing, memory manipulation, and I/O components. For these, time to solution is another metric that is used to measure performance. However, visualization applications present special needs that are often overlooked when measuring the performance of new architectures, such as response time, which is not factored into the measurement. Response time is driven not only by the performance of the high performance server, but also equally by the performance of the network, the I/O system, the system services software (i.e. the kernel, and in particular the process management system), and the display hardware and environment. With visualization applications, response time is really the only metric that matters. We will report on the available high performance visualization tools and the users' satisfaction level with their performance to round out our analysis of high performance computing tools and technologies.

1.4 Disclaimer

One of the challenges discovered while performing this investigation is the sheer number of tools available for high performance computing. With new tools appearing often while others are dropping out of use, this report can only represent an incomplete snapshot of what is currently available. There are many tools and technologies that are not discussed in the review and this should not have any implication as to their value either for current or future platforms.

1.5 Analysis Preview

We have attempted to provide an overview of a prominent subset of the available high performance computational tools and technologies, identifying their strengths and challenges for addressing next generation computing needs. The tools we reviewed can roughly be categorized as debuggers, memory checkers, performance analysis tools (including profilers, tracers, and visualizers), communication libraries, data visualization programs, and other parallel development and high performance computing aides. The product of this evaluation is a summary table of the discussed development tools and their operational environment.

The first step was to characterize current high performance applications and describe the tools and technologies they employ. The list of tools was collected through interviews with individuals in DNT, Engineering, Earth & Environment Sciences, and NAI while

many others were discovered from openly published sources and product websites. From the interviews, we also identified expectations for high performance computing needs in the next generation.

Ideally, each platform should have the same tools encompassing all the necessary functionality and performance characteristics. However, few tools span even our existing platforms or support all our programming paradigms. Future machines threaten to compound this discord with gaps in necessary functionality.

In some cases, such as distributed computing or other burgeoning high performance computing communities, tools either do not exist or are ad-hoc. In these cases, tools that seem to have some degree of acceptance within the community were evaluated.

Some tools, such as visualization software and debuggers, are run in a client-server mode. Visualization applications are also often developed on top of a suite of libraries and other packages that may or may not be optimized for high performance computing. For example, they may invoke software libraries on either the client or the server to perform functions such as secure communication, data transfer, rendering, animation, and imaging (such as Mesa). In order for the visualization tool to perform well, each of these supporting libraries must also be optimized for HPC.

1.6 Programs' / Applications' Current & Future Needs

As a whole, the development community wants tools that are available on all the various platforms and offer a high level of performance. They loathe the current state of having to learn to use a new tool on a new platform to fill required functionality they have elsewhere. The community is grateful for the few universal tools such as TotalView and TAU that provide needed capabilities across most platforms in our community, but these tools do not fulfill all the needs of the community. For example, an overall impression exists that Linux systems lack adequate memory checking tools, and few highly regarded memory checkers are available to multiple platforms. As developers have found existing tools to be lacking, they have resorted to primitive techniques such as printf to provide the mere basics of the information they seek. They often do not return to use more comprehensive or fully featured tools even many revisions after it has resolved the issues that turned those developers away. Often such primitive techniques are used in our community because developers are on a new machine before desirable tools have been ported. In other cases, a useful tool is lost when a machine is retired and the tool is not ported to another available platform.

1.7 Emerging Architectures

Although it is difficult to predict which technologies will win out over their competitors for a place in the future HPC market, it is safe to assume that, like now, the community will have to adapt to multiple competitive technologies coexisting before one is settled upon, which may take quite awhile. Some of these emerging technologies mentioned below may soon enter into the competition for various portions of the HPC market.

1.7.1 Streaming Data-driven Architectures

There are ongoing efforts to harness streaming data-driven architectures for those applications configured to use them. These can be divided into roughly two categories - Commercial Off-The-Shelf (COTS) systems such as Graphical Processing Units (GPUs) and innovative custom designs.

There is growing interest in the use of GPUs for general-purpose programming. Graphics processors offer the potential for cheap, COTS, high performance computing. GPUs are highly parallel, multiple pipeline, SIMD computation engines that are currently exceeding Moore's law in development. Several institutions, including University of Illinois Urbana Champaign, Stanford University, University of North Carolina, nVidia Corporation, and the GAIA project at LLNL are actively working in this area. Early results have shown that there are some problems which, when offloaded to the GPU, yield orders-of-magnitude performance improvement.

Custom architectures, such as Stanford's Merrimac project, are specifically designed for general purpose streaming data driven computing and offer the potential for strong performance for this computational paradigm.

Associated with these architectures are several compilers and supporting languages such as Cg, an nVidia corporation proprietary language whose goal is to provide a C-like interface to all major graphics cards. The Brook language specification from Stanford University is designed to be a C-like language for streaming architectures in general and will support GPUs as well as Merrimac. Reservoir Lab's R-Stream compiler offers an implementation of the Brook specification with significant extensions. It is designed to run on Merrimac and may potentially be ported to GPUs.

Current challenges with general purpose computing on both GPUs and other streaming architectures include the volatility of the hardware, and the lack of supporting tools. In the case of GPUs, there are significant architecture changes every six to nine months. This makes it difficult to provide a stable software abstraction layer to the programmer. Currently the only publicly available debugger for GPUs is a research project at Stanford University. Given the difficulty of translating general-purpose programs into the graphics programming paradigm, combined with the driver volatility, the need for development and debugging tools for these architectures is paramount.

1.7.2 PIM

One of the challenges for high performance computer architectures is the bottleneck between processors and memory. Processor-In-Memory (PIM) architectures (sometimes referred to as embedded DRAM) seek to address this challenge by combining CMOS and DRAM logic on the same chip. These architectures have a much wider interface to memory data allowing significantly greater bandwidth and the potential for greater overlap in accesses. There are many ongoing embedded DRAM projects especially in the ASIC (application specific integrated circuits) community.

The IRAM project at UC Berkeley is investigating PIM as a general-purpose technology. VIRAM1 is the first implementation of a UC Berkeley IRAM design. VIRAM1 consists

of four main components: scalar core, vector control unit, vector lanes, and embedded DRAM. The scalar core is a MIPS M5Kc 64-bit single issue, scalar core with 8KByte instruction and data caches. Both the vector unit and a single-precision scalar FPU interface to the scalar core through a co-processor interface. There are four vector lanes each with one fourth of an 8KByte vector register file. A full register file stores 32 64-bit, 64 32-bit or 128 16-bit numbers. Each vector lane contains two ALUs, one of which can perform single precision floating point as well as integer operations. VIRAM1 also includes eight embedded 13-bit DRAM macros from IBM.

1.7.3 Reconfigurable HW

Reconfigurable hardware enables the modification of a processor's logic gates functions and interconnectivity at run-time. The most common reconfigurable hardware is FPGAs (Field Programmable Gate Arrays). Current reconfigurable hardware is slower than conventional integrated circuits. Of recent interest in reconfigurable hardware has been the design of chips that have multiple modes of operations and can reassign resources during a computation, referred to as Polymorphous Computing Architectures (PCA).

An example of this type of reconfigurable architecture is the Monarch chip project under development by USC/ISI and Raytheon Corporation. The Monarch chip integrates several levels of granularity including RISC microprocessor cores and arrays of ALU embedded in a dataflow model of computation. The three components of Monarch are a dataflow stream engine, RISC processors coupled with DRAM memory, and a high speed interconnect that allows all the components to communicate. Monarch can be configured to create partitions executing in dataflow, SIMD, or RISC modes. RISC mode is used for logic-intensive computation, while dataflow and SIMD are employed for data-driven, computationally intensive stream computations.

1.7.4 Windows Server (Compute Cluster Edition)

Microsoft is developing a special version of its Windows operating system to run on cluster machines. This is scheduled for release in Fall of 2005. Despite the lack of enthusiasm this generates, we should note that another once seemingly unlikely contender in this market, Apple's Mac OS X, is already established on one of the top ten fastest supercomputers. Other similar systems are being considered due to the favorable performance/price value. Interestingly enough, this OS will not run on clusters using Intel's Itanium 2 chips.

1.8 Development Tools

Software development tools specific to parallel applications typically fall into one of these general categories: debuggers, memory checkers, and performance analyzers (including profilers and tracers). However, the tools' placement into these categories tends to be blurred, especially with regard to the more advanced multi-function tools.

Users are looking for tools with high degrees of robustness, usability, scalability, portability, and versatility. These subjective criteria carry more weight with end users than most objective measurements. Quantitative comparisons across tools are largely

meaningless due to the widely varying feature sets, methods, usage implementations, and platform restrictions. Therefore, various users' experiences and perceptions are related in place of objective side-by-side comparisons for tool evaluations.

For this study, we took some of the installed tools that looked promising and evaluated them with various physics simulation codes, or sought opinions from those who already had. In evaluating the tools, we considered the following criteria:

- **Ease of use** – How much of a time investment will it take to make use of the tool? How steep is the learning curve?
- **Availability** – On which platforms will the tool run? To what degree is the tool supported? What languages are supported by the tool, or is language even a factor?
- **Parallel Support** – Does the tool work with parallel applications? (Which type - Distributed? Shared memory? Threaded?) Does it scale well?
- **Application Impact** – What impact does using the tool have on the application? Are significant modifications required by the developer to implement the tool? How much slower will the application run?
- **Capacity Impact** – Does the tool generate a lot of data or does it require excessive resources to collect sample data?

1.8.1 Debuggers

It has often been the case that new architectures or environments expose latent bugs and portability issues not previously observed even in tranquil code sections. Debugging is unavoidable for large or developing code projects. Some problems may only appear on certain platforms, at large scale, or after lengthy run times. Code size, complexity, various parallel methods, and dependence on external libraries that can contribute to errors all make traditional debugging tools and methods unwieldy. Thus, we need better tools to facilitate the process. These tools need to be effective with our large, highly parallel and threaded, multi-language codes and ideally be available on all our platforms. They also need to integrate two and three dimensional data visualization across multiple processes and present it to the developer in a consolidated manner while providing reasonable response times. Fortunately, we already have some great tools for this purpose, but we took a closer look at what is available.

With the imminent arrival of architectures such as BlueGene/L, with tens of thousands of processors, scalability is probably the most important consideration for HPC debuggers. Working closely with selected vendors would help insure their understanding of the issues involved, and also provide them with a massively parallel environment in which to develop and test their tools.

Tools could be developed to report values of variables recently modified and where they were last set. They could also trace back an offending value to its likely origin. Tools that could automate the analysis of crashes would drastically reduce computer resources currently consumed by users manually hunting down bugs. However, this will require far more than just an execution trace reported upon failures. Extensive logic to interpret the faulty execution needs to be incorporated into the tools. They could then suggest a means

to correct the problem, as well as an informed explanation as to what probably caused the fault. Adding relative or comparative debugging capabilities would also be a welcome improvement such as provided by the Guard debugger described in Chapter 4.

1.8.2 Memory Checkers

Since we received our parallel Linux clusters a few years ago, our user community has identified memory checkers as a major gap in Linux cluster development tools. Adding to this despair is the lack of portable memory checkers common across all of our parallel platforms. Each platform has its own memory checker, each with some advantages and disadvantages over the others, and none that have the desired speed characteristics. Paying this performance penalty for extensive functionality when only specific functionality is desired, or lacking a required functionality under a certain language or on an intended platform is unacceptable. This has led many projects to develop their own memory tools, ourselves included. Given the vast number of memory tools out there, most with a very small and dedicated purpose supporting only one platform and language, it appears like most of the community as a whole also has noticed the lack of general and adequate memory tools, so they resort to filling their needs by themselves as well. Fortunately, our cries are being heard by the TotalView development team who has introduced a number of basic memory checking abilities into its popular debugger over the past year or so and is continuing to add more. Unfortunately, some of the more advanced memory checking features are both costly in time and difficult to port. Considering the other additional overhead imposed by TotalView and that some of its utility will be lacking on some systems, it may not be a feature used by typical users.

As we have seen in our tools review, there are quite a large number of memory checkers of varying capabilities, even on Linux platforms. However, full-featured checkers are few, and they are rarely cross-platform tools. Significant performance degradation results from applying these highly feature-laden checkers too. The ideal memory checker would be very configurable in its operation, allowing the user full control over exactly what functionality it applies to the code, and even where to apply it. Its complete set of features would be portable to all our platforms and support all our languages and parallelism. It should also be able to explain what went wrong as it reports memory faults, or at least give plausible explanations based on the code context where it occurred. Unfortunately, none of the known existing tools comes close to meeting this ideal. Our discussion of the numerous more specific memory checking tools will hopefully offer alternatives to those groups looking for a particular feature. We hope to reduce the number of groups (including ours) who resort to developing memory checkers themselves.

1.8.3 Performance Analysis Tools

The usable hardware lifespan of our high performance computers is typically less than the software development cycle, and given that multiple such architectures are available at any given time, it is not very efficient to tune an application specifically for one platform, especially to the detriment of the others in use. However, generally optimizing codes to use the machines more efficiently benefits everyone by reducing turnaround time, freeing up resources, and improving accessibility. Thus it is an important task for fair sharing of our computing resources, not to mention the added value gained.

Unlike debuggers and memory checkers that are frequently applied to our large parallel codes, few of our large projects routinely apply comprehensive profiling tools. Those that do typically are interested only in basic timing information, and they want an easy way to access it. For them to employ any such tool, they request that it quickly identify problems through top-down analysis. Furthermore, they require there be an easy correlation between performance data and the source code. However, with optimizing compilers that often pipeline statements, unroll loops, reorder commands, and inline functions, the correlation can be confusing.

There are numerous untapped aspects to performance analysis and a great number of tools available to help. Unfortunately, that is part of the problem – users do not know which tools will be worth their time to learn and apply, or what the payoffs will be. There are many specialized tools with steep learning curves or with difficult to interpret results. They may be difficult to implement, and since very few are commercial products, they often lack the maturity and robustness that is desired. Even after the learning, applying, and interpreting steps, a user may implement some derived changes that unwittingly degrade performance on other platforms which that tool or functionality was not available on. Therefore, it is not surprising that performance analysis is rarely done, but that could change if one such tool becomes as widely available and user-friendly as debuggers have.

There are two basic categories of performance analysis tools – profilers and tracers. Profilers provide a summary of execution statistics and/or events. They give an overview of the overall performance of the program, often broken down to the functions, loops or even user-specified sections. They often use periodic sampling during a run with little impact on overhead, so longer runs to improve the accuracy of the results are encouraged. They are best at exposing bottlenecks and hotspots in overall execution. Tracers, on the other hand, often do provide this profile information along with breaking it down along a temporal context. They record a much larger stream of information during a run in order to reconstruct the dynamic behavior over a finely resolved time-sequence. Often with parallel runs, a separate log file is created for each process or thread being traced. These files can quickly become huge, and as such, tracing is not recommended for lengthy runs or at large scales. However, the detail they provide allows a realm of performance tuning options not available through profiling alone. The large amounts of data that tracers generate require a visualization tool or other type of interpreter that presents it comprehensively to users in order to make sense of it.

The biggest challenge for highly parallel performance analysis tools, especially tracers, is consolidating its collected data into a meaningful presentation to the user that highlights aspects for tuning. This task is sometimes handled by a separate GUI visualization tool. Some visualization tools can display data collected from a variety of tracing tools, as well as sometimes having their own integrated tracer. Such tools are described below in their own visualizers' category, which is a subset of the tracers' category. Ideally, they should be able to compare data collected from multiple executions to indicate platform differences or help users identify effects of their modifications, but few allow this. Even fewer provide automated analysis with insight for prominent tuning improvements. Although a tool may have an easy to use GUI, you probably will have to manually navigate its menus to redisplay desired analysis for each data set examined.

A true performance analysis tool should indicate where and which optimizations will yield the greatest cross-platform and machine-specific benefits, including actual code modification suggestions. Although a few attempts have been made to include such logic in an analysis tool, they are not yet mature enough to be effective in the actual user community. Having effective automated parallel performance tuning tools would be quite beneficial. There are a few tools that show promise for being further developed toward the ideal. Unfortunately, one of them, Vampir, seems to be devolving away from it of late (see the Vampir performance visualizer tool in Chapter 4). Hopefully other contenders, like TAU and Vtune, can continue to progress toward it. There are numerous smaller analysis tools also mentioned in Chapter 4 that offer intriguing functionality that may fill a niche which no other tools can as of yet. This discussion of such tools will hopefully benefit some group that was not previously aware of such tools.

1.9 Communication & Networking

As mentioned above, the main challenge for the communication and networking community is achieving high-speed computation across a large distributed area (either geographical area, or a large collection of collocated processing units) given the constraint of physical (e.g. latencies and robustness) and administrative (security and scheduling policies) barriers. On massively parallel systems, there is debate about whether technologies such as MPI will be the dominant paradigm or whether there will be other contenders. In the distributed world, Web services are addressing interoperability issues but they need to be brought into high performance levels. The tools to support this mode of computing have not yet developed, so their next challenge will be distributed debuggers and monitoring and analysis tools.

1.10 Visualization

Visualization tools are an important component in software development that cannot be overlooked when reviewing high-performance computing. Few tools available today address the needs of visualization in a high-performance computing environment. The following are characteristics of visualization tools for high performance computing applications:

- Very large data sets often spread across many files and even across different file systems, platforms, and networks.
- Ability to navigate multiple complex datasets.
- Client/server hardware architecture with high-performance multi-processor display devices networked to the mainframe.
- Requirement for fast response times usually measured in frames per second. A rate of 10 or more frames per second is expected.

In reviewing visualization software for high performance applications, we found that there are few tools that can scale up to meet the demands of applications that will be running on the next generation of supercomputers. Many of the latest tools for these environments are being developed in-house, often by the same groups that write the large-scale applications software – VisIt is an example of this.

1.11 Future Tools

Not long ago there were serious debates about the feasibility of large parallel Linux clusters. Now that Linux has come of age and proven that it can provide record-breaking computational power on relatively inexpensive hardware, we find ourselves wondering what will be next. As we recall the initial lack of development tools on the Linux clusters, and the perception that this persists even today, especially with regard to decent memory checkers, we hopefully will avoid such misfortunes in the future by anticipating upcoming architectures and porting tools to them before deployment.

Future tools will incorporate more logic to interpret the analysis results automatically and suggest improvements. True performance analysis tools will be developed which indicate where and which optimizations should be used, and estimate their impact on multiple architectures. All tools will scale to large numbers of processors and have a user selectable feature set available on numerous platforms. Such abilities are necessary if they are to rise to meet the challenge of our HPC community's current needs.

Concerning visualization and post processing, we can expect to see improvements in the capabilities of visualization tools that are available today, particularly in the area of performance. We expect to see these tools take advantage of new hardware and software technologies to include streaming architectures, programmable GPUs, and lightweight kernels.

It is expected that trends will continue to require increased intercommunication between distributed applications. The current tradeoff between interoperability and performance will increase in severity over the next decade. The current success of Web services and enhanced interoperability will drive the need for better performance solutions and better security. There are only a handful of tools for evaluating the performance of networked applications and most of these are focused on the dynamic structure and performance of the network rather than on the application. It is expected that this will be a fertile and cross-cutting area for development.

1.12 Gaps

Our user community is clamoring for an effective, portable, comprehensive, and fast memory checker for our large and highly scalable codes. There are different memory tools, each with some variation of effectiveness and features, available on our older parallel platforms but nothing effective exists on our growing number of Linux platforms or for our newest 64-bit architectures. A tool similar in performance and features as Purify on the Sun platforms is highly desirable on all our platforms. In addition, more automation in the tools is desired, such as taking much of the tedious tracing of crashes back to earlier errors that could have been detected. Another example is suggesting how to optimize hotspots in addition to identifying them, which is something that Shark, a tool on Mac OS X platforms, does. Users in AX-Division apply this tool on some of their applications, and the implemented optimizations often produce speedups on other platforms as well. Some users could also benefit from a tool that identifies areas where aliasing, roundoff sensitivity, CPU register swapping, sensitivity to floating point reordering, and other such things are occurring so that they can make better use of the compiler's optimization abilities. Also, we lack tools that can compare debug results

across platforms or between slightly different code revisions, which now is a very tedious task often not even attempted that really helps locate new or platform-aggravated bugs.

There are a variety of areas that have yet to be adequately addressed by the current generation of visualization tools. These gaps fall into two categories that include visualization features and support for new hardware technologies. In the area of visualization features, there is a growing need to visualize meshes with complex geometries such as AMR meshes and higher order elements. A capability to support interfaces (or an API) for calling the viz tool from the application would also be beneficial; VisIt supports this feature. The other area where gaps exist is in regard to new hardware technologies. Several new technologies that could impact visualization include streaming architectures, programmable GPUs, and immersive technologies.

As mentioned earlier, the main future challenges for communications technologies are high speed data movement in an interoperable environment and appropriate security mechanisms in environments that may be inherently anonymous (e.g. P2P). There is a greater challenge to the tools community. There are currently no general-purpose debuggers or memory tools for highly distributed computations. Current success in massively parallel high performance computing has shown that, for a distributed paradigm to succeed in the high performance arena, it will require the development of sophisticated tools to meet this need. In addition, tools to measure and analyze the performance of applications that may span multiple distributed resources, some of which may be highly parallel, will be required.

1.13 Conclusions

There are a number of Commercial Off-The-Shelf (COTS) and publicly available software development tools applicable to our parallel programs that have potential to fill in the gaps of needed functionality on our current and future platforms.

There are few tools available in the visualization area that can adequately address the requirements for processing data from large-scale simulations. The performance of these tools is dominated by the network and the architecture of the visualization software. Since the architecture significantly affects the performance of these tools, we believe that only those tools designed specifically to meet these challenges can meet the performance requirements – tools that are retrofitted to address HPC have not had a good track record in meeting these challenges.

Over the past several years, there have been trends towards COTS technologies because of the significant price/performance ratio they offer. The main challenge in using COTS technologies for HPC is that tools and software are typically designed for some purpose other than HPC (e.g. Linux clusters are used for server management in business applications, GPUs are only designed for the gaming industry). This trend will undoubtedly continue, but the burden will be on high performance tools designers to develop custom tools to meet the specific needs of the high performance community. One advantage of the adoption of COTS and open source technologies is the possibility of convergence tools that span multiple platforms.

Custom high performance hardware will continue to exist in arenas that are not easily satisfied by commodity products. For example, areas such as interconnect technology may still require development beyond what the marketplace demands. In this case, it is less likely that general-purpose tools will be available and vendor specific solutions may dominate.

Chapter 2: Programs / Applications Current & Future Needs

2.1 Engineering

A variety of large analysis codes are developed and used in Engineering. Applications include structural mechanics, fluid dynamics and heat transfer, and electromagnetics. These codes are run on platforms ranging from Linux workstations to supercomputers.

Developers on Engineering code projects use a variety of tools and technologies to debug and optimize their applications including Vampir, TotalView, ZeroFault, Purify, Valgrind, and HPMCount. TotalView is used in serial and parallel, with up to 1024 processors, for debugging their applications. Watchpoints are used heavily to stop execution when a variable changes.

To visualize MPI (message passing interface) performance, developers use Vampir. However, developers would like faster visualization performance, and lower memory overhead. To measure cache performance and FLOP rate, they use HPMCount in serial mode on the IBM. For their future requirements, they would like a tool that would show performance (fp, cache, etc) on a line-by-line basis, such as the long defunct Thinking Machine's PRISM did.

2.1.1 ParaDyn

ParaDyn is a finite-element code used for structural analysis applications, not to be confused with the performance analysis tool of the same name described later. It allows full system and detailed component level modeling. Problem sizes in the range of one to ten million elements are now being simulated with ParaDyn.

ParaDyn is designed for distributed memory parallel systems with MPI software. It currently runs on several parallel systems and workstation clusters. These include the IBM SP2, Origin 2000 systems, CRI T3D/T3E, Dec Alpha clusters, and Linux Clusters. The partitioning of the full problem domain into sub domains is an automated step for the analyst. Efficient and robust parallel contact algorithms have been a significant focus of the algorithm development in ParaDyn over the last several years. Developers on the ParaDyn Project use a variety of tools for debugging and visualization including TotalView, Purify, GRIZ and VisIt (for visualization), Tecplot, and the Mili I/O library for storing output. ParaDyn is written primarily in Fortran (F77 and F90).

2.1.2 Diablo

Diablo is a new finite-element code used for structural multi-mechanics applications. By multi-mechanics, we mean that it can perform coupled-computations for multiple types of physics including structural, thermal, and electro-magnetic. Diablo runs in parallel using MPI for distributed communications. Typical problems contain from 20,000 to 500,000 elements and up to 100 materials. Large problems typically run on up to 512 processors. The Diablo code developers use TotalView for debugging and GRIZ for visualization. VisIt is also beginning to be used for viewing Diablo results. For I/O, the Vista I/O library is used for restart files, and the Mili I/O library and HDF5 are used for graphics

files. These files are used only for visualization. For some types of output, Tecplot is also used. Diablo is written primarily in Fortran90 with a minimal amount of C.

2.1.3 EMSolve

EMSolve is a new parallel electromagnetics code under development in Engineering. The goal of the EMSolve project is to develop a simulation code to model the time-dependant Maxwell equations of electrodynamics on 3D unstructured grids. This code is based on FEMSTER, a finite element library implementing high-order discrete differential form basis functions. These basis functions include both curl conforming and divergence conforming basis functions which can be used to more accurately model the vector fields arising in electromagnetics. The approximate solutions obtained by using these bases are energy and charge conserving and the time-integration methods are provably stable. EMSolve uses MPI for distributed communications, and the Hypre solver library. I/O is performed using the SILO I/O library. The development group uses TotalView and Insure++ for debugging. VisIt is used for visualization. EMSolve is written is C++. It has a GUI that was written in Java.

2.2 Energy and Environment

Various modeling and simulation codes are used within the Energy and Environment Directorate. The directorate is engaged in many diverse software projects to support research in atmospheric science, earth science, energy and environmental science. These codes support capabilities ranging from understanding atmospheric processes in the areas of global climate, chemistry, biogeochemistry, and mesoscale meteorology, to the study of water/hydrology, geochemistry, environmental science and risk analysis.

2.2.1 Atmospheric Science

Scientists in the Atmospheric Sciences Division (ASD) perform research on global climate and conduct predictions for global and local transport of toxic pollutants through the atmosphere. A large program within ASD is the Program for Climate Model Diagnosis and Intercomparison (PCMDI). The PCMDI mission is to develop improved methods and tools for the diagnosis and comparison of general circulation models (GCMs) that simulate the global climate. One of PCMDI's large software systems is called the Climate Data Analysis Tool, or CDAT. This is an innovative system that supports exploration and visualization of climate scientific datasets. As an "open system", the software subsystems (i.e., modules) are independent and freely available to the global climate community. CDAT is easily extended to include new modules. The Live Access Server (LAS) and the Distributed Oceanographic Data System (DODS) are examples of other components integrated with CDAT. CDAT also makes use of the Globus middleware software. The power of CDAT comes from Python and its ability to seamlessly interconnect software. Python provides a general purpose and full-featured scripting language with a variety of user interfaces including command-line interaction, stand-alone scripts (applications) and graphical user interfaces (GUI). The CDAT subsystems, implemented as modules, provide access to and management of gridded data (Climate Data Management System or CDMS), large-array numerical operations (Numerical Python), and visualization (Visualization and Control System or VCS).

Their code developers in ASD use a variety of tools for debugging and performance analysis including Flint, TotalView, and Vampir (indirectly). Developers primarily use TotalView for debugging, and visualization is done using the in-house developed tool CDAT. They primarily use NetCDF for their portable file format, but HDF and ASCII files can also generated and post-processed. IDL is also heavily used for visualization. Developers in ASD felt that TotalView is limited in that it can only be used easily for interactive jobs on small numbers of processors. [TotalView has made improvements in this area but the impression still remains]. For their future requirements, they are heading towards higher resolution and will need faster computers with more processors. In addition, their disk usage and file storage requirements will increase by an order of magnitude.

One of the most difficult challenges facing climate researchers today is the cataloging and analysis of massive amounts of multi-dimensional global atmospheric and oceanic model data. To reduce the labor intensive and time-consuming process of data management, retrieval, and analysis, many research sites have come together to develop intelligent filing system and data management software for the linking of storage devices located throughout the United States and the international climate research community. This effort, headed by PCMDI, NCAR, and ANL will allow users anywhere to remotely access this distributed multi-petabyte archive and perform analysis.

2.2.2 NARAC

The core of NARAC's mission is an atmospheric simulation system, ARAC-3, used for emergency responses to hazardous atmospheric releases. In addition, NARAC develops a suite of end-user tools including the NARAC iClient, NARAC Web, and NARAC PDA. ARAC-3 is an object-oriented set of distributed UNIX servers using Orbix for middleware and Objectstore for object persistence. The end-user tools allow customers to remotely access NARAC services. These tools are built using J2EE, EJB, JSP, and JDBC technologies.

2.2.3 GNEM

The Ground-Based Nuclear Explosion Monitoring Research & Engineering Program (GNEM R&E) is responsible for deriving seismic calibrations to improve techniques to monitor underground nuclear explosions on a worldwide basis. A central aspect of this monitoring work is the discrimination of nuclear explosions from naturally occurring phenomena such as earthquakes and man-made disturbances such as mining explosions.

In order to automate the development of calibration products, two things are required. First, a mechanism is needed for the initial collection of waveform and parameter data from which signal travel-time and amplitude correction surfaces can be derived. Second, the development of correction surfaces and other calibrations are needed. In the past, it has been possible for researchers to download individual waveforms and make event-by-event measurements. However, today researchers are bombarded with an influx of Terabytes of data with orders of magnitude increase expected in the next decade. The next generation computational framework for handling voluminous real-time data sources will need to support time-critical modeling, simulation and analysis.

The LLNL GNEM group provides a seismic research database (SRDB) to support the larger multi-institution GNEM R&E project. The current software is a Java based extensible object oriented acquisition system framework that supports multiple information types in a relational database, geographic information systems, and product/data visualization tools relying heavily on Oracle database support.

The next generation software will develop new online acquisition capabilities that will enable focusing resources on multiple areas of interest and support computationally intensive simulation capability for adaptive calibration and validation.

2.2.4 ESG

The Earth System Grid (ESG) is a virtual collaborative environment that links distributed centers, users, models, and data. It provides scientists with virtual proximity to the distributed data and resources that they require to perform their research. The ESG integrates and extends a range of Grid and collaboratory technologies, including the DODS remote access protocols for environmental data, the Globus Toolkit technologies for authentication, resource discovery, and resource access, and the Data Grid technologies developed in other projects. In addition, the ESG group is developing new technologies for creating and operating "filtering servers" capable of performing sophisticated analyses, and for delivering results to users.

2.3 DNT

DNT develops a variety of large multi-physics simulation codes that operate on multiple parallel platforms and can utilize large numbers of processors. Developers employ a wide range of tools and technologies to debug and optimize their applications. They are also often early adopters of the latest available cutting-edge parallel platforms.

2.3.1 ARES

Ares is a multi-block, regularly connected 1D, 2D, and 3D multi-material radiation hydrodynamics code. Ares is written mostly in C but has some C++ and FORTRAN as well. It can use both MPI and OpenMP / POSIX threads to achieve parallelism. MPI is required for inter-node communication while OpenMP / POSIX threads can optionally be employed for intra-node communication. It is a large, highly parallel code designed to run on hundreds of thousands of processors (theoretically, that is, barring external scaling issues). TotalView, as well as command line debuggers such as gdb and dbx are frequently used. Developers periodically apply a variety of memory checkers including ZeroFault, Third Degree, and Purify. The thread correctness checker Assure has been applied, and profilers such as Vampir and prof have been used extensively during ongoing optimization efforts. Users typically run on up to 512 processors or more when such resources are available. For graphics visualization, Ultra and VisIt, as well as built in parallel graphics capabilities, are used.

2.3.2 ALE3D

Ale3D is a 3D multi-physics hydrodynamics code that was originally written in C but has been largely migrated into a C++ code, plus sections of FORTRAN 90 and FORTRAN 77. It parallelizes using MPI and can operate using threads, although they are rarely used. It is a large, highly parallel code designed to run on hundreds of thousands of processors. Developers mainly use TotalView but also gdb for debugging. They have also employed Purify, Valgrind, and sometimes ZeroFault, and Third Degree to verify memory correctness. Their experiences with ZeroFault and Third Degree have not met their expectations, and they are limited by Valgrind's lack of parallel support. They highly desire an effective parallel memory checker like Purify on all our platforms. The developers use VisIt for visualization. Users typically run on up to 512 processors.

2.3.3 KULL

Kull is a 3D modeling code for inertial confinement fusion driven by Python and written mainly in C++. It has some C portions and a small bit of FORTRAN as well. It parallelizes using MPI communication and does not use threads. It is a large, highly parallel code designed to run on hundreds of thousands of processors. Developers use TotalView for debugging, but often employ printf statements instead due to TotalView's sluggishness and lack of support for some of their C++ constructs. They also employ ZeroFault quite successfully for memory analysis as needed. Additionally, their code incorporates TAU, mainly for its timers and FLOP counters. It is most often run on under 256 processors. For visualization, they primarily use VisIt, but also use Ultra and other graphing tools. They desire a more comprehensive 2D visualization tool incorporating all the features found scattered across the variety of visualizers they use. They also need a 64-bit version of ZeroFault on our newest IBM platforms. They are interested in tools that work with Python. They have experienced problems using Third Degree and report that Electric Fence is too limited in its capabilities. They are looking for another decent memory checker on more platforms than just AIX.

2.4 NAI

NAI uses a variety of tools and technologies designed to meet the diverse programmatic needs. These span the range from large-scale massively parallel simulations using MPI running on Linux clusters to distributed applications using Web service technologies, XSD, XSLT, SOAP and RMI to lightweight applications running on embedded systems and FPGAs.

2.4.1 JCATS

JCATS is an object-oriented software system providing an entity-level human-in-the-loop, event-based conflict simulation. DOD, DOE and the Secret Service and others use JCATS for training and analysis. Applications include military operations, peacekeeping, drug interdiction and site security. Software is primarily developed in C++ on UNIX platforms. UNIX development tools used include make, CVS, emacs/xemacs, debuggers and Purify/Quantify.

Chapter 3: Current Generation Computer Systems

3.1 BlueGene/L (IBM / LLNL)

BlueGene/L (BGL), due at LLNL next spring is an experimental architecture. Compared to the other mentioned platforms, it has vastly more nodes and CPUs and quite a bit less memory per CPU. BGL will have 65,536 nodes, each with two 667MHz PowerPC 440 processors, for a total of 131,072 processors. Each node will have only 512 MB of memory. The lower amount of per-node memory will require that large problems be run on many more nodes than existing parallel platform configurations. As such, efficient scalability is crucial for effective performance. System functions will be offloaded onto separate I/O and service nodes, allowing compute nodes to focus on application execution. The set of front-end nodes with which the users will interact will be running a more traditional OS than the streamlined version operating on the compute nodes. This lightweight OS micro-kernel running on the compute nodes was custom developed by IBM Research and may restrict the use of OpenMP threading. Instead of nodal task time-sharing, strict one-to-one task distribution among CPUs is deployed. Its distributed network will employ a Federated Ethernet switching infrastructure. Although its CPU has been around for many years, its unique operating system and hardware environment provides its own challenges to those porting tools to it. Will memory tools be available? Will message passing be efficient? Will performance tools be available?

3.2 Purple (IBM / LLNL)

Purple will be a new ASC machine at Lawrence Livermore National Laboratory (LLNL) similar to, but larger than, our latest IBM platforms. It will have 1528 nodes, with eight 1.5GHz Power5 processors and 16 GB of memory per node. It will employ a new 64-bit CPU and a Federation interconnect. Development tools will need to be ported to this platform, but porting them to our current 64-bit Power4 platforms first should simplify that task.

3.3 Q (HP / LANL)

Q is an ASC supercomputer at Los Alamos National Laboratory (LANL). Each of its two segments has 1024 AlphaServer ES45 nodes with four EV68 1.25GHz 16MB cache processors each. All of its nodes have at least 8 GB of memory although one quarter of them have double or quadruple that. Three-quarters of these higher memory nodes have 16 GB while the remainder have 32 GB. It has a Quadrics Elan3 interconnect and is controlled with RMS (Resource Management System) and LSF (Load Sharing Facility). It uses PFS for its parallel file system. The well-established Alpha chip based servers, interconnect system, and OS it is based on already has had numerous development tools ported to it.

3.4 Red Storm (Cray / SNL)

Sandia National Laboratory's soon-to-be-deployed Red Storm system will have 108 node cabinets with ninety-six 2.0GHz AMD Opteron processors each. This is a rather new CPU among the ranks of top performance machines to which many parallel tools have yet

to be ported. The processors are arranged four to a board with 4 GB of memory per board. They will have a 3-D mesh-based Cray interconnect, with networking chips on a daughter board atop each processor board. It is not one of Cray's vectorizing platforms. Like BlueGene/L, Red Storm will run a specialized lightweight OS (Catamount, developed at Sandia) on its compute nodes instead of a common full-featured OS.

3.5 X-1 (Cray / ORNL)

The Cray X-1 is designed to combine current MPP processing architectures with vector processing capabilities and a high performance interconnect. In contrast to other MPP computers, the processor in the Cray X-1 is custom designed to achieve 12.8 GFlop peak performance. At the lowest level of the design are single-streaming processors (SSP) with two 32-stage 64-bit floating-point vector units and a one scalar unit. At the next level up, four SSP's are combined to form one multi-streaming processor (MSP). Each SSP is capable of 3.2 GFlops when the vector units are fully utilized. This gives a peak performance of 12.8 GFlops per MSP. Four MSP's are placed on each node, along with either 16 or 32 Gbytes of RDRAM memory. The per-node bandwidth is 204 GB/sec, which is designed to keep the vector-unit from starving. The interconnect is designed for low-latency and high bandwidth. The nodes feature a distributed shared memory design, so the memory is physically distributed with each processor, yet is logically shared by other processors. To enable full use of this design, compilers that can find and utilize vector parallelism will be required. It will also depend on programmers re-working particularly compute intensive sections of code to enable efficient vectorization on this platform.

3.6 Earth Simulator (NEC / Japan)

The Earth Simulator in Japan is a vector machine based on the NEC SX architecture. Its 640 nodes each have eight vector processors capable of 8 GFlops/s performance apiece. This distributed memory system is interconnected with 640x640 single-stage crossbar switches. Inter-node parallelism is achieved using either HPF (High Performance FORTRAN) or MPI, while intra-node communication can be achieved through OpenMP. Their program development environment, called PSUITE, runs on a workstation, providing a source browser, editor, debugger, and performance analysis tool. For their FORTRAN codes, they employ FSA – a tool to statically analyze programs – and HPFPROF – an execution performance analysis tool that incorporates an easy-to-use GUI. More familiar to our users, they also employ pdbx as their symbolic debugger, and Vampir as their MPI performance analysis tool.

3.7 Linux Clusters (Various Developers / Various Sites)

The general category of "Linux clusters" encompasses a wide range of supercomputers, with differing processors, networks, file systems and vendors. They are popular for their favorable price to performance ratio. Large examples of such are:

- **Lightning** (Linux Networx / LANL) has 1408 nodes (4 GB memory per node) with 2816 AMD 2.0 GHz Opteron processors, a Myrinet 2000

Interconnect, and a Beowulf Distributed Process Space.

- **Thunder** (California Digital / LLNL) has 1024 nodes (8 GB memory per node) with 4096 Intel 1.4 GHz Itanium II Madison Tiger 4 processors, a Quadrics QSNetII Elan4 interconnect, the Lustre file system and runs the RedHat / Chaos O/S.

- **MCR** (Intel / LLNL) has 1152 nodes (4 GB memory per node) with 2304 Intel 2.4GHz Xeon processors, a QSNet Elan3 interconnect, the Red Hat / Chaos OS, and a Lustre file system. Note that ALC (ASC Linux / LLNL) and other Linux clusters at LLNL have similar features to MCR.

3.8 System X (Apple / Virginia Tech)

Virginia Tech's homemade System X uses 1100 Apple XServe G5 2.3GHz dual processor nodes with 4GB of memory each. It has recently completed its upgrade from its original Power Mac G5 nodes, which were slower, hotter, lacked error-correcting code RAM and occupied three times more physical space in their facility. It is interconnected via InfiniBand switches and is running Mac OS X, which is a variant of Linux. Its message passing library is MVAPICH, InfiniBand's version of MPI, and uses IBM's xlc and xlf C and FORTRAN compilers. They have no scalable parallel debugger or other development tools yet but TotalView is being ported. They primarily use printf for debugging but are running mostly mature codes.

3.9 Columbia (SGI / NASA)

SGI's Columbia at NASA uses 10240 Intel Itanium 2 processors. This system is different from most supercomputers in that it uses a "single-system image" approach, where each of its twenty 512-processor Altix 3000 machines interconnected with an InfiniBand network run a single operating system, making large scale runs inefficient and more complicated than in other systems. Each of the twenty machines will share 1TB of memory. It runs the Linux OS.

3.10 MareNostrum (IBM / Spain)

The MareNostrum system at the Barcelona Supercomputer Center in Spain is a collection of 1782 IBM JS20 BladeCenters with two 2.2GHz PowerPC 970 CPUs and 4GB of memory each. Its interconnect system is by Myricom. When fully configured in the spring of 2005, it will have 2282 servers with 4564 total processors.

Chapter 4: Parallel Code Development Tools

4.1 Debuggers

4.1.1 TotalView

The TotalView debugger is a product of Etnus, LLC. Without a doubt, TotalView is the HPC industry's most powerful, fullest-featured, parallel debugger. TotalView supports all common parallel programming methods and languages including MPI, threads, SHMEM, OpenMP, HPF, PVM, fork/exec, C, C++, FORTRAN, mixed-language and hybrid-messaging codes. It is also supported on all major HPC platforms in the US including IBM, Intel, HP, Sun and SGI.

TotalView incorporates all of the usual and expected debugger functionality, plus a variety of features too numerous to mention here. Users have complete execution control and data visibility at the program, process, and thread level. TotalView provides many useful features such as the ability to "dive" into objects, value lamination across threads and processes, data watchpoints, process barrier points, evaluation points and run-time expression evaluation. It also includes advanced features such as data analysis, call tree analysis, array filtering, sorting and visualization, and MPI message queue statistics/graphing. TotalView's list of features continues to grow with each version release, and most recently, has added several very useful memory debugging features, including leak detection, heap memory debugging, and memory usage statistics. TotalView offers a command line interface for power users and automated batch script debugging, which eliminates GUI overhead and the need for interactivity. The ability to customize the GUI interface and a fully integrated help system are two user-friendly features also worth mentioning.

TotalView is the most commonly used Tri-lab parallel debugging tool. It is one of the few tools that is popular with virtually all code development teams. Because of its usefulness and portability, it is available on all of our parallel platforms. Additionally, and importantly, LLNL works closely with the vendor to develop new features requested from our HPC community. These features typically find their way into future product releases. Feedback and bug reports from our users are tracked and responded to by the vendor in a manner concomitant with this productive relationship.

Scalability to thousands of processors is the primary concern. The GUI's startup time and overall performance degradation, especially at larger scales, are issues. However, the vendor has responded to these concerns in the past and has made significant progress in these areas in recent versions. More work is needed though. The product is not fully integrated with C++ yet, which is an issue especially for the AX-Division users. Some architectural issues affecting TotalView also limit its debugging ability. Currently, on AIX platforms, only one variable can be watched at a time. Also, on the Intel Linux clusters, only up to 4 bytes can be watched at a time. Users would like at least 8 bytes, so a double can be entirely watched.

The ability of the vendor to address new platforms, such as BlueGene/L that is expected to posses a minimal kernel, minimal execution environment, tens of thousands of processors and new network topologies, is high on the list of concerns. Hopefully, the vendor's years of experience with MPP systems and past performance should enable them to meet these challenges. High cost is a very serious concern for most customers.
This tool is essential to have on all upcoming platforms as long as the competition lags. The vendor should be encouraged to continue their close ties with the labs' developers and their responsiveness to our issues. They should enhance their product to work well on future generation massively parallel systems and maintain their clear leadership position in the industry. Tri-lab efforts should also be extended to promote competition, which will result in more competitive pricing, performance and functionality.

4.1.2 DDT

DDT (Distributed Debugging Tool) is a product of Allinea Software (part of Streamline Computing). DDT is another comprehensive, GUI-driven, parallel debugger. DDT supports C, C++, and FORTRAN codes on common HPC platforms such as Intel/AMD x86, Itanium2, AMD Opteron (32 + 64), IBM Power, SGI MIPS, Sun Ultrasparc and HP Itanium & PA-RISC. DDT is relatively new on the HPC debugger scene, and is perhaps the only viable competitor in TotalView's domain. Like TotalView, DDT provides all of the basic and expected debugger functionality, including breakpoints, watchpoints, expression evaluation, execution control (step, next, halt, etc.), register access, viewing and modifying data, attaching/detaching to processes, core file debugging, etc. Similarly, DDT has a number of advanced features, some of which are similar to TotalView and some of which are unique. Similar features include complete support for threads (both pthreads and OpenMP threads), multi-process MPI, focus group definition to effect commands on a subset of threads/processes, conditional breakpoints, 3-D visualization of array data, MPI message queue graphing, watchpoints, and multi-dimensional data viewing. Unique features include expression comparisons across processors (statistically, graphically, and by user-specified precision equivalence), the more user-friendly way it handles stdout and stderr, and better support for C++ constructs. What is also unique about DDT is that it actually depends upon a lower-level native debugger, such as gdb, idb or pgdbg. However, DDT does lack a few desirable features such as TotalView's new memory checking abilities and several unique features specifically requested by our users. To some users, DDT provides a better-organized and more intuitive interface than TotalView. One of the strongest points in DDT's favor is its cost compared to TotalView, as it is significantly less expensive.

The issues facing DDT are similar to those facing TotalView in terms of performance and scalability. In addition, since DDT is a newcomer to the HPC parallel debugger field, with a very strong (if not monopolistic) competitor, there is some concern whether the HPC community will accept DDT. This latter concern would impact the longevity of the product and how extensively it might be enhanced in the future.

DDT has not been tested extensively yet with large, production level, parallel jobs within the Tri-lab and not much information is locally available on how it would perform with jobs that have hundreds of MPI processes in our environment. The DDT documentation mentions that it can be used with over 256 processes, but initial experiences with our installation suggest this fact needs to be explored further.

DDT is a new alternative to TotalView and is much more reasonably priced. As with TotalView, the vendor should be encouraged to be responsive to our users' improvement requests on both current and new platforms. In order to do this however, DDT needs to be better "publicized" locally, and then gain sufficient critical mass such that users can actually provide useful feedback to the vendor. Offering training opportunities and seminars for local users may help the publicity effort.

4.1.3 Ladebug

Ladebug is HP/Compaq's parallel debugger supporting C, C++, and FORTRAN programs on Tru64 and Alpha Linux platforms. It provides both a command-line interface and GUI. Ladebug supports debugging multi-process programs and multi-threaded programs, however its parallel interface and functionality are substantially more primitive and limited than TotalView's and DDT's. For example, although multiple threads/processes can be debugged in the same session, only one thread/process can debugged at a time. In addition, it does not support user defined process/thread (focus) groups or viewing message queue data. Aside from this, most of the usual and expected debugger functionality is provided, including breakpoints, watchpoints, expression evaluation, execution control (step, next, halt, etc.), register access, viewing and modifying data, attaching/detaching to processes, core file debugging, etc. One unique and potentially very useful feature supported by Ladebug is kernel debugging.

Ladebug's lack of support for any non-Alpha-based machines prevents it from gaining much popularity. Currently, the only Alpha-based system of interest to the HPC community is LANL's Q system, and with the demise of the Alpha processor line, Ladebug's longevity is limited.

It needs to be ported to other platforms in order to survive.

4.1.4 PGDBG

The PGDBG debugger is a product of the Portland Group, Inc. It provides both a command-line interface and GUI to debug C, C++, and FORTRAN codes on 32-bit and 64-bit Linux systems. PGDBG can be used to debug threaded (pthreads, OpenMP, Linuxthreads) SMP applications, multi-process MPI applications, and hybrid programs that include both threads and MPI. Most of the usual and expected debugger functionality is provided by PGDBG including breakpoints, watchpoints, expression evaluation, execution control (step, next, halt, etc.), register access, viewing and modifying data, attaching/detaching to processes, etc. PGDBG does not support core file debugging, however. For parallel programs, the user can define focus groups comprised of selected processes and/or threads. Focus groups permit execution control commands to be applied to a subset of processes/threads. This useful functionality is similar to that provided by TotalView and DDT. PGDBG is also able to display MPI message queue information.

Its scalability is limited to 64 processes.

PGDBG has only recently been enhanced to provide full parallel debugging capabilities. Not much has been documented locally with respect to how useful and robust these new capabilities are. One suggestion would be to further explore PGDBG's parallel debugging capabilities, and to compare/contrast with TotalView and DDT.

4.1.5 GDB / DDD

GNU's portable debugger gdb is primarily a traditional command-line serial debugger found on most UNIX systems, much like dbx. The most recent version of gdb supports some elementary thread-level parallel debugging, and is the basis of some other debuggers, but it lacks a GUI interface. DDD (the Data Display Debugger) on Linux platforms can function as a GUI for gdb, dbx, and even supports Python, Perl, and Java.

Although DDD has many features and has been widely ported, large debugging sessions can easily overwhelm this tool, and parallelism is limited.

These are mature, useful debugging tools. They may serve as underlying parts in future parallel tools, but no development in these tools themselves will enable them to be suitable for debugging problems that are only manifested in large parallel runs.

4.1.6 PDBX

Under AIX, IBM provides a parallel, command-line tool built upon dbx, called pdbx. At one time, IBM also supported a GUI for pdbx, called xpdbx; however, this is no longer the case. Although pdbx supports debugging parallel MPI programs, it is limited to use under AIX, is entirely command-line driven, lacks support for integrated process/threads debugging and by most other measures is very primitive and lacking in functionality when compared to other parallel debuggers.

The command line tools are cumbersome when controlling parallel sessions. However, they are useful for debugging serial bugs in applications that would otherwise run in parallel, and for automation of simple debugging features such as traces. The pdbx tool lacks many desirable features like advanced data visualization, and becomes unwieldy as it scales.

It is possible that some users may find pdbx useful for "quick and dirty", low-level debugging tasks, however most users would be better advised to pursue an alternative such as TotalView or DDT.

4.1.7 IDB

IDB is Intel's Linux debugger for 32-bit and 64-bit systems, and supports debugging C/C++ and FORTRAN programs. IDB provides both a command-line interface and GUI, in either DBX or GDB mode. Most basic and expected debugging functions are provided. In previous versions of IDB, there was support for multi-threaded and multi-process programs. However, in the most recent version (8.0), this support is no longer available, and it is not clear whether it will be restored to the product any time soon.

4.1.8 Guard

Often, a bug seems to appear only on one platform, or between two executables or two problem inputs whose minor differences don't suggest the bugs cause. In these cases, having relative debugging capabilities would be quite helpful. Relative or comparative debugging tools compare runs between two executing programs, be they on different platforms, running different executables, or using different input. Using this technique, one can quickly identify where the code deviates from acceptable behavior, and the user does not need to have intimate knowledge of the expected internal workings which they might otherwise need to have when using a traditional debugger to, say, debug a slight result discrepancy. This feature would also be beneficial when porting codes & determining the effects of numerically sensitive compiler optimizations. A parallel relative debugger called Guard, from Guardsoft in Australia, might fill this need. It is available on Linux, AIX, and Solaris systems and works on C, C++, and FORTRAN MPI codes. It employs the gdb debugger but unfortunately, it has no GUI interface.

4.1.9 printf

Printf is the fallback and time-honored method of debugging, but is hardly appropriate for large parallel codes. It is wrought with tediousness, often requiring numerous recompiles and pouring through pages and pages of code as you narrow in on a problem. Its applicability is limited and often useless for certain memory problems, like in the stack, or with intercepted signals, like from an unsuspected process.

It is unfortunate that this method is sometimes the only one available to some code groups in our HPC community for tracking down some types of bugs, especially on the newest platforms while developers wait for tools to be ported.

One benefit of this, albeit with many other costly tradeoffs, is that it can be done in batch, eliminating the frequent idling of parallel resources while the user is interacting with the tool, such as via a GUI.

4.1.10 Other parallel code development aides

Specifically for automating the error detection among your message passing via MPI is a tool called MPI_Check. Not only can it detect run-time message deadlocks and memory buffer errors, it can also be applied at compile-time to check for argument consistency. It works only on FORTRAN codes, although a C/C++ version is reportedly being developed. A similar tool that checks MPI correctness in C code called Umpire is under development here. However, it only operates on SMP machines, not on distributed memory systems. A similar but more portable tool called MARMOT is being developed within the European CrossGrid project. Released early this year, it is free, supports C and FORTRAN MPI codes, but not MPI-2, on Linux, AIX, and a few other clusters. Another tool called S-Check can predict how code refinements will impact parallel performance.

4.2 Memory Checkers

4.2.1 Third Degree

Third Degree is a memory correctness tool on Tru64 platforms supporting C and C++ codes. It reports memory leaks, uninitialized memory, and invalid addressing, among other things. It analyzes both heap and stack memory on all tasks of an MPI or threaded application. Apply it to an executable and it will create an instrumented version that you then run. It creates a text file report as output that is directly readable or it can be interpreted by dxheap. It finds problems found in all included libraries as well, including system ones, in which it occasionally reports errors that really are not problems. Since faults are recorded as they are encountered, users typically scroll through the report looking for the potentially catastrophic errors nestled amongst others that are rather insignificant or erroneous. This tool was created using Atom, which users can employ themselves to create their own tools for collecting such information as cache utilization, branch prediction, and execution tracing, among other things. However, such usage is beyond the average user.

4.2.2 Zero Fault

ZeroFault is available only to 32-bit AIX applications but has been successfully applied to large parallel runs, albeit with a slowdown factor of roughly 40 times. It is a robust memory checker with a GUI under which you can search for errors in real-time or analyze upon completion. Its GUI interface is similar to Purify's. If you are certain you know which process a problem is occurring on, you can restrict the processes it analyzes to a smaller subset or even just one of the processes your application is running on. Unfortunately, it tends to report false problems in optimized codes, or even misses problems, and works better on unoptimized codes. It supports FORTRAN, C and C++ codes without needing any recompilation. Our users hope this will be ported to our new 64-bit AIX platforms.

4.2.3 Valgrind

Valgrind is a free Linux tool that allows you to detect memory management and threading bugs. However, it does not effectively work for parallel programs. It uses dynamic binary translation so you do not need to modify, recompile, or even re-link your applications. It supports C, C++, and FORTRAN codes, but, like most memory checkers, programs run significantly slower – about 20x in this case. There are five tools included in Valgrind. Memcheck is its complete memory checker. Addrcheck is a less robust memory checker with fewer features but faster operation. Cachegrind is its cache profiler, which is an uncommon feature in a memory checker, but its speed degradation is quite considerable. Massif is its heap profiler and shows usage over time with locality. Helgrind is its thread debugger showing data races between POSIX threads. This tool is quite unique among memory checkers and is a welcome addition. If only it also supported MPI applications, this might be the memory tool our users feel is lacking on the Linux clusters.

We found that Valgrind was easy to install and use. Of particular interest was the fact that it worked directly on the binary executable. Valgrind's set of tools for debugging and

profiling Linux-x86 executables revolve around the main tool, called Valgrind, which is really an x86 CPU software simulator. Specific debugging or profiling tools are added to this core package as 'skins'. One of these skins, for example, is the cache simulator cachegrind. Another nice feature of Valgrind is the capability to write new skins. Skins consist of shared objects that are loaded at runtime.

4.2.4 Insure++

Insure++ is a powerful all-encompassing memory correctness tool for finding all types of memory reference errors, including those in the stack and global memory as well as the heap and dynamic memory. It finds leaks, corruptions, invalid and uninitialized pointers, plus mismatched types and operations on unrelated data blocks. Although it does not require source code modifications, your source does need to be compiled through its insight tool. During this stage, it can identify algorithmic and variable name errors, and other sloppy coding issues. However, for some codes, getting past this stage may require source code modifications. It has successfully been deployed on large parallel programs; however, it does suffer from the usual speed degradation plaguing most robust memory checkers. It is more than just a memory checker too. Its Inuse tool can visualize, in real-time, memory manipulation. In addition, its TCA – Total Coverage Analysis - tool provides code coverage information. Insure++ is available on AIX, Tru64, Linux, Irix, and SunOS systems for C, C++, and FORTRAN codes. It is a commercial product available from ParaSoft.

4.2.5 Purify

Purify detects memory errors and leaks. It is a commercial product available separately from IBM's Rational Software and is also included in their PurifyPlus software which also performs execution profiling and code coverage analysis by their Quantify tool. It supports C and C++ codes without any recompilation. Most popular among our users from its SunOS support version, it now also supports HP-UX, Irix, and Linux. Also, support for AIX is expected by Purple's arrival. However, the version for these newly supported systems requires GNU compiled executables and still needs significant functionality and performance improvements before it equals the mature SunOS version, which many of our users hold in high regard against which other checkers are measured. The most popular response we found users requesting was a memory checker "like Purify" available across our platforms.

4.2.6 SmartHeap

SmartHeap SMP is a memory checker optimized for multi-processor systems. It provides comprehensive memory debugging APIs that are portable, high-performance, and thread-safe. Unfortunately, it does not work with distributed memory parallelism, and is limited to 72 threads. However, it is a rather well regarded tool with a large user base of major corporations who are developing standard industry software for use on the quickly growing number of SMP servers. Hopefully they will expand this tool to support MPI applications in the future.

4.2.7 Great Circle (Application Saver)

Great Circle is a bit different in that it also provides automated memory management tools. One interface they provide, called GCTransparent, performs transparent garbage collection, while another, called GCPointers, employs reference counters for storage reclamation. This commercial tool will soon be replaced with Application Saver, a more comprehensive debugging system that may deserve further investigation but will initially only support C/C++ code with threaded parallelism. It can detect subtle scalability bottlenecks such as excessive lock contentions and virtual memory use. It also captures extensive forensics for application faults, including per-thread statement-level execution histories, and includes a separate Forensics Viewer that enables rapid root cause analysis of application failures. It can capture detailed snapshots that can be used to completely reconstruct the events leading up to an application fault or failure.

4.2.8 Electric Fence

Electric Fence is a freely available shared library replacement for malloc routines. It is different in that is uses the system's virtual memory hardware and stops execution on bounds violations so that the offending statement is identified when used in conjunction with a debugger. It is available on Linux, Tru64, HP/UX, and Solaris systems. However, it is too limited in its features to be an effective analysis tool for our applications.

4.2.9 MemUsage

MemUsage is a memory analysis tool developed by our group that allows large parallel codes to dynamically track memory usage across a range of computing platforms. This library presents current real memory usage (per node and per process, which is also typically per processor), remaining real memory usage (per node or per process), and the high-water memory usage for these categories (on platforms that support it) to the application. All this data is available in mega-bytes and as a percentage of real memory.

Applications must know when they are close to depleting 'real' memory (and thus getting into swap space). Crossing into virtual memory can result in serious performance penalties. Applications need a cross-platform method to detect real memory usage. Also, they must know the maximum amount of memory allocatable by a single process. On some systems this is less than the amount of real memory, on others it is greater.

It is not enough for each process to individually track memory usage. Many supercomputers have nodes with multiple processors that share a bank of memory. On such platforms, the amount of real memory per node can be depleted before each individual process on that node uses all of its per processes real memory limit. This requires that the tool be aware of the process-to-node mapping and track real memory both on a process and a node basis. The MemUsage library meets this need, and with minimal overhead, which few other tools do.

The memory profile tracking features of the tool require that the library contain methods to collect and present the memory usage across thousands of processors and hundreds of nodes in a manner readily available to the application. This allows the application to use this information in real time and make execution adjustments accordingly.

Large parallel applications need to be balanced in terms of computational demands, memory usage, interconnect demands (latency, bandwidth, and frequency of message passing), and I/O demands. This library provides the raw data for memory usage, which may be used by load balancing algorithms during execution.

In addition, this library may be used to determine the memory footprint of the libraries linked with the main application. On machines such as BlueGene/L, which have less memory per node than other HPC machines, this ability allows application and library developers to quantify the memory requirements of their code, and identify areas that may need to be reworked for ports to new machines or to better run on existing machines.

This library works on Intel Linux, Dec OSF1, IBM AIX, and Sun SunOS systems. This library provides low-level and a high-level interface routines. The high-level routines provide information that is analyzed across all processes and nodes in a parallel system, and provides minimum and maximum memory usage both for nodes and for processors. In contrast, the low-level routines provide all the summary memory information for all nodes and processors. The high-level functions are useful when a parent process needs to monitor memory usage and know if any single processor is getting close to swapping. The low-level routines provide data that may be used by load balancing algorithms to determine how best to balance the workload of the problem across the nodes.

Our group created this memory analysis tool in order to fill a need for knowing memory high-water-marks, plus available and remaining node and total memory. This collection of features was not found in any one low overhead tool available to us previously. The Parallel Tools Consortium had been developing a similar tool, MUTT – a Memory Utilization Tracking Tool – along with a variety of other Ptools projects that are now employed in a variety of performance analysis and parallel development tools, but MUTT was not completed before the consortium disbanded a year or so ago. However, another of their tools, PAPI, is still being developed and has recently incorporated some useful memory tracking features of its own.

4.2.10 Mpatrol

Mpatrol is a free C and C++ library that provides extensive debugging, profiling, and tracing of memory allocations. It is highly configurable and can use a fixed-size static array for allocations instead of using the heap. It is thread-safe and can be included as one large object file so that it can be linked directly with the application. Hooks for its replaced memory functions are provided for debugging. Snapshots of the memory can be taken, and utilization can be displayed during execution. Allocation failures can even be mimicked to test error-recovery subroutines. However, a header file is required to be included during compilation to implement this tool. Fortunately, environment variable settings can control its abilities in executables it is compiled into, which prevents tedious recompilations.

4.2.11 Other memory checkers

There are numerous memory checking utilities that focus only on small subsets of a code's memory management tasks. Ccmalloc, for example, provides a C & C++ version of the malloc library calls that log memory leaks and corruptions. It lacks a GUI and other sophistication and cannot detect some types of illegal memory reads. However, it is easy to link into your existing code by specifying '-lccmalloc –ldl', and it works with optimized versions. Dmalloc replaces many standard C memory management routines with more robust error checking versions, plus it works with threaded codes. Other such tools for C include fda, and memwatch. Similar debuggers for C++ codes include LeakTracer, libcdw, NJAMD (Not Just Another Malloc Debugger) and YAMD (Yet Another Memory Debugger). MemCheck Deluxe is a memory tracking program for C and C++ codes that reports the largest and smallest allocations, the most number of allocations, and the total memory usage. Mprof profiles your memory allocation per function, reports leaks, and features an ease of use similar to the gprof execution profiler.

Assure was KAI/Pallas's tool for threads checking in C, C++, and FORTRAN codes. It validated correctness of OpenMP parallel codes and identified thread-safety issues. A few years ago, Intel took it over and stopped developing it. This is an unfortunate loss of important functionality available to our parallel users, however not many codes in the Tri-Lab community have incorporated pthreads. In addition, the tool was exceptionally slow and difficult to implement and only simulated the threading environment by analyzing data movement sequentially, sometimes leading it to falsely report problems, so the outcry has been minimal. Fortunately, Valgrind has added thread-checking functionality, and if they add support for MPI applications too, that will likely fill this void.

4.3 Performance Analysis Tools: APIs

There is a plethora of tools for parallel performance analysis, none of which are clear leaders in their field. They typically suffer from scalability problems, portability limitations, and ease of use issues. However there are a few underlying APIs (Application Programming Interfaces) which many of the tools share in common to gather their data. The APIs dictate how a code is instrumented for use with an analysis tool, or what hardware monitors the tool can inquire. The least desirable tools involve source code modifications for performance instrumentation.

4.3.1 Dyninst

An ideal feature that users clamor for incorporates dynamic instrumentation, so no recompilation is required. Many tools accomplish this by using Dyninst, the portable dynamic instrumentation API permitting code insertion into running executables. Other APIs have been developed around Dyninst. It is very portable, well maintained, and has advancing functionality.

4.3.2 DPCL

Many tools also accomplish dynamic instrumentation by using DPCL – the Dynamic Probe Class Library – which is based on Dyninst. The DPCL library is an object based C++ class library used an infrastructure for developing dynamic tools. Numerous tools have been developed which use Dyninst and/or DPCL, including Paradyn, Dynaprof, Dynamic Kappa-Pi, TAU, and LLNL's ToolGear.

4.3.3 DCPI

DCPI – HP's Digital Continuous Profiling Infrastructure – is basically Tru64's version of PAPI. HP's uprofile, kprofile and pfmon tools employ these counters for performance analysis. As with all measurements, all these counters introduce overhead that can affect the speed, cache, and memory performance of a code. The overhead from DCPI is significantly lower than that experienced using PAPI on other systems, but unfortunately, DCPI only provides system information and cannot virtualize counters on a per-process basis. Its future is in doubt as Alpha based systems are no longer being developed.

4.3.4 PAPI

PAPI (Performance Application Programming Interface) is an interface standard for accessing hardware performance counters. It is widely available and provides a rich set of hardware counters for measuring the machines performance. PAPI provides two levels of interfaces: a high-level interface for measuring pre-defined sets of events and a low-level interface that gives the user the ability to define complex event sets.

The low-level PAPI interface deals with hardware events in groups called *EventSets*. EventSets reflect how the counters are most frequently used, such as taking simultaneous measurements of different hardware events and relating them to one another. For example, relating cycles to memory references or Flops to level 1 cache misses can indicate poor locality and memory management. In addition, EventSets allow a highly efficient implementation that translates to more detailed and accurate measurements. EventSets are fully programmable and have features such as guaranteed thread safety, writing of counter values, multiplexing and notification on threshold crossing, as well as processor specific features. The high-level interface simply provides the ability to start, stop and read specific events, one at a time.

PAPI is now a standard for obtaining hardware counter values on all of our platforms. Like DPCL, it was initially developed by the Parallel Tools Consortium. Unfortunately, using PAPI on large-scale applications has encountered scalability issues. Future versions of PAPI will implement hardware sampling and estimating aggregate counts to improve its speed. Other planned improvements include reporting node and total memory available, memory high-water-marks, and process memory locality.

4.3.5 PMAPI

PMAPI is IBM's native performance counter library, which is basically their version of PAPI. It works with IBM's Power3 and Power4 systems running AIX 4.3.3 and up. It is essentially a set of kernel extensions that provide most of the PAPI functionality, and as

such, it is often interchangeable with PAPI in most general discussions and literature (including this paper). When PAPI on these IBM systems is referenced, it implies that PMAPI is being used.

4.4 Performance Analysis Tools: Profiling Toolkits

4.4.1 PE Benchmarker

The PE Benchmarker Toolset comprises a suite of tools used to collect and analyze program event trace and hardware performance data. Like the Xprofiler tool, it is provided as part of IBM's Parallel Environment software on only its AIX SP platform. The toolset consists of three main components: The Performance Collection Tool (PCT) for collecting MPI and event data, a set of Uniform Trace Environment (UTE) utilities for merging and converting their trace files into other formats, and the Profile Visualization Tool (PVT) for trace file data visualization.

The PCT tool provides run-time instrumentation to collect either MPI and user event data, or hardware and operating system profiles for one or more application tasks. PCT is built on the Dynamic Probe Class Library (DPCL) dynamic instrumentation technology. By using dynamic instrumentation, there is no need to recompile an application, and instrumentation overhead is reduced. The user can select through the PCT GUI which MPI events should be traced, and dynamically specify arbitrary points within a code where tracing should be turned-on and off. Runtime instrumentation of MPI and user event data results in AIX trace files, one per task. Instrumentation of hardware and operating system profiles results in the creation of netCDF output files, one per task. PCT's functionality is also available through a command-line interface.

The UTE conversion utilities are used for several purposes. The uteconvert utility is used to convert individual AIX trace files produced by PCT into the UTE format, which is easier to use for GUI viewers. The utemerge utility merges multiple UTE files into a single file and the utestats utility is used to generate a text report of performance metrics. The remaining utility provided is the slogmerge utility. Slogmerge converts UTE files into a single SLOG file for use by Argonne's Jumpshot graphical MPI viewer. It should be noted that IBM does not currently provide a graphical viewer for its own UTE files with the PE Benchmarker toolset, but instead, defers creation of such to third parties.

The Profile Visualization Tool (PVT) component is used for viewing and analyzing hardware and operating system profiles collected by the PCT as netCDF files. The metrics that can be viewed include wall clock time, call counts, CPU usage, memory usage, paging activity, context switches, and a subset of PAPI hardware counter events (floating point operations, loads, stores, etc.). PVT's main display window is divided into two panes. The left pane presents a source view, which is actually a hierarchical list of the functions/loops that were profiled, not the true source code. The right pane presents the data profile view, which is a histogram for the selected metric (call counts, CPU time, loads, etc.). Histogram bars are aligned next to their corresponding function/loop in the source view. Both halves of the display scroll to maintain the alignment. PVT can also produce and display a variety of reports. PVT has built in help, and also offers its functionality through a command-line interface.

The PE Benchmarker toolset was not well received during its evaluation at LLNL. In addition to issues of GUI awkwardness and slowness, scalability was a concern. It also suffers the same major drawback as Xprofiler – it can only be found on IBM AIX platforms. Its GUI interface seems unintuitive and rather featureless. Employing its scalable log file format (SLOG) is essential for large runs.

4.4.2 HPM Toolkit

IBM's Hardware Performance Monitor (HPM) Toolkit uses the PMAPI hardware performance counter events for performance measurement of applications running on AIX systems. It supports serial and parallel (MPI, threaded, and mixed mode) applications written in FORTRAN, C, and C++. The HPM toolkit was developed to measure performance on Power-3 and Power-4 processors. The HPM Toolkit consists of three main components: the hpmcount utility, the libhpm instrumentation library, and the hpmviz GUI.

The hpmcount command-line utility is used to start the application and specify which PAPI events should be used and where to direct output (the default is stdout). PAPI events can be specified as a comma separated list of specific event numbers, or as predefined event sets. During execution, performance data is automatically collected without the need for code modification or recompilation of the executable. Following execution, performance data is written to output files, one per process. Output consists of a plain text report showing execution wall clock time, hardware performance counters information, derived hardware metrics, and resource utilization statistics. When your application terminates, a summary of performance metrics is printed. One would use this utility if they wanted to gather performance data without modifying their application code. We found this tool useful as a first step in gauging the application's overall performance.

The instrumentation library, libhpm, provides routines for manual instrumentation of source code. Its functions can be called from the application code to collect hardware counter information. Developers can selectively instrument parts of their code to collect certain specific information. With the library, the developer can isolate performance of specific code regions even down to a single line of code. Such instrumentation requires recompilation and linking with the library. Specification of PAPI events is similar to hpmcount. Output consists of a plain text summary report for each process that was instrumented, and also, *.viz file(s) which can be used as input to the hpmviz utility for graphical display.

The hpmviz component of the toolkit provides a relatively simple graphical visualization of the performance data generated by hpmcount and contained in libhpm's output files. The main window of the GUI is divided in two panes. The left pane displays for each instrumented section (identified by its label) the inclusive duration, exclusive duration, and count. Sorting in ascending/descending order can be performed on any of these three metrics. Right clicking on a labeled instrumentation section opens a "metrics window" which displays additional detail on that particular section, including the derived metrics. Left clicking on a labeled instrumentation section displays the corresponding source code.

Recently, this toolkit was incorporated into a newly created toolkit with many more analysis tools called the IBM High Performance Computing Toolkit. To avoid confusion with Rice University's similarly named HPCToolkit, we will continue to refer to this as the HPM Toolkit, albeit repackaged with expanded functionality. Aside from the HMP Toolkit, this new IBM HPC Toolkit also includes a suite of additional performance analysis tools for IBM AIX platforms only. This toolkit provides the ability to analyze hardware performance, shared memory OpenMP performance, message passing performance, performance simulation, and SHMEM routine performance. All of these capabilities are integrated into its PeekPerf GUI that is part of the toolkit. Note that most of the components of this toolkit have been distributed by IBM as separate toolkits/libraries, and may still be obtained as separate items. The IBM HPC Toolkit is supported for C/C++ and FORTRAN serial, threaded and MPI programs.

The toolkit's new dynamic performance monitor for OpenMP, called DPOMP, is a tool based upon the industry-standard POMP (Performance OpenMP) API. It can be used to generate a detailed profile describing overheads and time spent by each thread in three key regions of the parallel application: parallel regions, OpenMP loops inside a parallel region, and user-defined functions. The profile data is presented in the form of an XML file that can be visualized by the GUI.

MPI message passing performance is accomplished through several included, low-overhead libraries. No source code modification is required in order to profile and trace MPI calls. The PeekPerf GUI is used to display the performance data in a number of formats including summary reports, detailed reports with source code traceback, and graphical charts. Additionally, the toolkit includes two "turbo" libraries that implement high-performance collective calls and MPI-2 put/get calls.

For those who wish to use CRAY SHMEM parallelism, the toolkit provides an IBM implementation of most SHMEM routines, and the ability to profile their performance. The primary reason a developer might choose to use SHMEM is the performance gains over alternate programming methods.

Finally, IBM's ACTC (Advanced Computing Technology Center) is continuing to develop this toolkit. Work is currently underway to incorporate MIO (modular I/O) and its sparse matrix libraries into the package.

The PeekPerf and DPOMP tools in the toolkit are very new and, from the outside, seem to offer a lot of functionality within a unified GUI framework. Some of its components, such as the HPM Toolkit and the MPI profiling and tracing libraries have been around for a while and have proven themselves useful. However it and other OS specific tools suffers from the inherent drawback that it is (and almost certainly will always be) available only for one platform.

The HPM Toolkit provides a quick and easy way to automatically collect hardware performance metrics on IBM AIX Power3 and Power4 platforms. It also provides the flexibility for user-defined instrumentation regions. However, like all platform specific tools, its usefulness in a multi-platform HPC environment is limited, especially since most of the functionality of this toolkit can be found in other multi-platform tools.

We instrumented an application with the HPM toolkit and found it a valuable tool for measuring performance characteristics of the IBM SP2.

4.4.3 HPCToolkit

The HPCToolkit is an open-source multi-platform suite of analysis tools supporting Tru64, Linux, and Irix platforms. Their papirun tool profiles executables using statistical sampling of hardware performance counters, while their papiprof and xprof tools provide more detailed profiling. In particular, papiprof is used to map profiles collected using papirun back to program source lines. HPCView is their tool for manipulating the performance data, computing derived metrics, and correlating program structure information to produce a performance database. The resulting database is browsable through its HTML interface. There are various panes that display program files, source code, and tables of data. Data is displayed hierarchical, with buttons to flatten and un-flatten it to speed up top-down analysis. Since the tables are sorted, the flatten operation makes short work of diving into the program from the top to identify the most important files, procedures, loops, and statements. In addition, their bloop tool analyzes executables' structure to identify loops and source lines. This extraction of the hierarchical program structure includes that of its libraries and, since it works on binaries, is largely independent of the language used. To facilitate automation, the utilities in the toolkit are intended to be run using scripts and configuration files. Once these are set up, rerunning the program to collect new data and all of the steps that go into generating a browsable dataset can be entirely reproduced. The scripts automate the collection of data and conversion of profile data into an XML-based format.

4.4.4 PerfSuite

PerfSuite is a set of tools for Intel-Linux systems for performance analysis. They include their OptView, PerfExplore, psinv, psrun, and ProfView tools, as well as the libperfsuite, libshwpc, and libpspmpi instrumentation libraries. The psinv utility reports the machine's hardware characteristics. Its psrun tool performs hardware event counting and profiling of MPI and thread-based applications dynamically without relinking or changing any code. The resulting collected data can be visualized with PerfExplore, while the PerfView tool allows exploration of profile data across multiple data files. The graphical OptView tool can assist with interpreting the effectiveness of various compiler optimizations. Instrumentation can also be achieved through its libraries. The suite includes other tools as well, including a tool to estimate the potential speedup of parallel programs, and a version of perfex. Although still in beta development status, their goal is to freely provide simple and portable tools that give a comprehensive overview of highly parallel performance to help focus your optimization efforts.

4.4.5 TAU

TAU – Tuning and Analysis Utilities – is a portable profiling, tracing and visualization toolkit for parallel codes. It supports MPI, threads (OpenMP and pthreads), and hybrid (MPI+threads) programs written in C, C++, FORTRAN, Python and Java. All C++ language features are supported by TAU, and for C++ programmers who want profiling

that handles per-instance, per-class, and per-template class information, TAU is perhaps the only viable performance analysis tool. TAU is very portable and has been ported to a number of platforms including SGI IRIX, Intel x86 Linux, Sun Solaris, IBM AIX, HP HP-UX, HP Alpha Tru64, NEX SX, Cray X1, T3E, SV-1, Hitachi SR8000, Apple OS X and Microsoft Windows.

The TAU development team, located at the University of Oregon, receives funding from the Department of Energy for the development of a performance analysis tool that can meet the demanding needs of the DOE supercomputing community. Specifically, TAU targets the development of a multi-platform performance analysis tool that operates in the capacity range of the ASC machine environments. The project's goal is to assist the ASC application developers in understanding their codes and optimizing them to exploit the full potential of the ASC architectures.

Towards those goals, TAU's features are numerous. It is probably one of the most full-featured HPC performance analysis tools. TAU combines the technologies of several other performance analysis tools including Dyninst dynamic instrumentation, PAPI hardware counters, Opari OpenMP instrumentation, and Vampir and Paraver trace visualization.

Like a number of other performance analysis tools, TAU depends upon instrumenting the application to be analyzed. During execution, performance data is collected and written to files, which may then be used as input for a visualizer. TAU provides several options for instrumentation. It can be inserted in the source code using an automatic instrumenting tool (Performance Database Toolkit), dynamically using the DyninstAPI, at runtime in the Java virtual machine, or manually using TAU's instrumentation API. Manual instrumentation requires source code modification by the programmer and usually consists of inserting TAU routine calls in those portions of the code that need to be traced. However, TAU provides a means to easily turn off instrumentation without having to remove the inserted TAU routine calls. Users find that this feature is both useful and convenient. Finally, TAU provides a way for the user to define groups of routines that can be profiled together.

TAU has a collection of tools for collecting and analyzing application performance data. It can be configured to do profiling, tracing, or both. It has multiple instrumentation interfaces, including dynamic, linked, and source level interfaces, and it is highly configurable. Like many of the other performance toolsets that we used, it provides a library of functions that can be called from your application code to collect performance metrics. The user selects counters of interest through environment variables. TAU also provides the capability to use the PAPI hardware counters.

Profile data can be viewed in plain text format through TAU's pprof utility, which is good for basic profiling, or graphically using the paraprof GUI. Paraprof's displays include a number of color-coded statistical histograms that show selected metrics (wallclock time, hardware performance counter events, etc.) per routine, per process/per thread. Displays can be sorted in a number of ways such as inclusive execution time, exclusive execution time, execution time over all nodes, etc. Paraprof is also able to display profile data in a 3D "terrain" format and as scatterplots. A nice feature of

paraprof is the ability to view profiles generated by other profilers such as mpiP, vprof, dynaprof and papiprof that have been converted through supplied utilities.

Trace data is collected in files separate from profile data, and must be merged into a single file with the tau_merge utility. The tau_convert utility is then used to convert the merged file into a format that can be used by a third-party viewer. TAU does not provide a trace viewer, but instead, supports conversion into formats used by Vampir, SDDF, ALOG and Paraver.

PerfDB is a performance database tool related to the TAU framework. The PerfDB database is designed to store and provide access to TAU profile data. A number of utility programs have been written in Java to load the data into PerfDB and to query the data. With PerfDB, users can perform performance analyses such as regression analysis, scalability analysis across multiple trials, and so on. A large number of comparative analyses are available through the PerfDB toolkit. Work is being done to provide the user with standard analysis tools, and an API has been developed to access the data with standard Java classes. The TAU toolset also includes a GUI for viewing performance data called RACY (Routine and data ACces profile displaY).

Following is an example of TAU calls that we placed in an application:

Initializing TAU:

```
TAU_PROFILE_INIT(argc, argv);
TAU_PROFILE_SET_NODE(0);
```

Enabling the counters:

```
TAU_PROFILE_TIMER(t1, "main-loop", "int(int, char**) C", TAU_USER);
TAU_PROFILE_START(t1);
```

TAU is probably the most versatile, configurable and portable performance analysis tool available. It scales well with threaded and distributed parallelism and offers robust timing and hardware performance measurements using PAPI. Although it is not a commercial product, it is quite mature and stands out in this field for its versatility and portability. TAU has been around for a while and by all outward appearances, should continue to stay around and continue to be developed for the foreseeable future.

TAU's single greatest drawback however is its steep learning curve and the fact that some of its most desirable functionality is easier to get from other tools, such as basic hardware counter info that can be obtained directly from PAPI. Another drawback is that building instrumented executables can be a tedious and error prone process involving complex makefiles. Although TAU provides tracing functionality through the VTF3 trace library, the best use of the tool is for profiling.

4.4.6 SpeedShop

SpeedShop is another performance analyzer suite for SGI systems. It supports C, C++, and FORTRAN codes employing MPI, pthreads, or OpenMP. This integrated package of tools can run performance experiments on an executable and examine the results. Such experiments report CPU usage statistics and hardware counter sampling. It also supports attaching Purify and debuggers on the executables. To accomplish this, some instrumentation of the executable is necessary, which it will perform automatically. Its

ssrun tool is similar to pixie in many ways. It records info collected during execution experiments. This data can be used for compiler feedback. In addition, its own API routines insert caliper points into your code. Other tools include a detailed timing tool, and memory tools to restrict memory usage or force paging. It includes a library that helps solve floating-point exceptions too. What may be a drawback for some is that the programs must be built using shared libraries. SGI recently announced it is developing an open-source version of SpeedShop for Linux, named Open/SpeedShop, but it is not expected for release until the summer of 2006.

4.4.7 AIMS

AIMS (Automated Instrumentation and Monitoring System) is a software toolkit for analyzing C and FORTRAN77 parallel programs supporting a variety of communication libraries. Its tools can illustrate algorithm behavior, help analyze execution, and highlight problem areas. The suite consists of four tools. Its automatic instrumenting tool is called xinstrument that inserts event recorders to trace subroutine invocations, message passing, and synchronization operations. It has a statistics mode to reduce overhead, and a GUI that allows users to load specific files for custom instrumentation. Code segments can be displayed and the user can pick instrumentable constructs. Data structure instrumentation is also done, as well as automatic generation of models useful when studying scalability. Its monitor library must also be linked in, which records communications, barriers, I/O, and other state transitions. Once a tracefile has been generated, their perturbation compensation (or pc) tool can attempt to remove the effects the instrumentation has incurred on the execution. AIMS has four different visualization tools. Their VK or View Kernel displays the dynamics of program execution using animations. It also supports simultaneous visualization of computational and communication patterns as well as analysis of data movement. Its tally tool presents an overall execution profile including where time was spent. Its Xisk tool shows statistics and can explain performance failures via simple indices. It provides plausible explanations for observed performance in terms of commonly occurring performance problems in message-passing programs. Finally, its MK tool models the parallel performance for performance prediction and scalability analysis. It can estimate how the program would behave if the execution environment were modified. Its tracefiles can also be converted into formants used by ParaGraph, which has an easier to use interface, and SvPablo, which offers a larger range of options.

4.4.8 KOJAK

The KOJAK tool suite includes new functionality to help automate the performance analysis of parallel applications. Particular emphasis is on having automation techniques to transform trace data into an overview presentation of performance behavior. Its component called expert automatically analyzes event traces to uncover performance problems. Its opari tool automatically instruments OpenMP code while its epilog library collects event traces that can be presented via their visualizer cube. This new tool suite shows much promise but needs to become more mature for general users' acceptance.

4.5 Performance Analysis Tools: Profilers

4.5.1 Dynaprof

Dynaprof is a parallel performance analysis tool designed to dynamically insert all of its performance measurement instrumentation directly into an application's address space at run-time. Dynaprof uses either DPCL or DynInst to insert its instrumentation in the form of "probes". Currently supported probes implement PAPI for collecting hardware counter data, and a wallclock probe for measuring elapsed time, both on a per-process and per-thread basis. Because the instrumentation occurs at run-time on binary executables, there is no need to recompile applications or link with any libraries.

Using this command-line tool is extremely simple. Just call "dynaprof" with the name of the executable to instrument and the desired options and arguments. Then, issue simple Dynaprof commands to define probes, specify which functions to instrument, and then run the program. Output format is actually controlled by the probe used and for both of the probes currently provided, consists of compact ASCII files written to disk, one file per-thread, per-process. A utility is provided for each probe to process the selected output file into a human-readable report. Users may also write their own probes and use whatever output format is appropriate, for example a real-time data feed to a visualization tool or a static data file dumped to disk at the end of the run.

Development of Dynaprof is an ongoing software project at the University of Tennessee's Innovative Computing Lab. The software is available for AIX, IRIX, Solaris and Linux, and can be downloaded from their web page.

The Dynaprof tool is certainly an easy to use and useful tool for collecting PAPI and wallclock run-time profiling information, if those are the only metrics desired. The fact that users can write their own probes and employ them within Dynaprof's framework could help expand the usefulness of this tool. The provided output format for the PAPI and wallclock probes is basic by design, and probably of not much use out-of-the-box. Trying to analyze plain text reports, one per-process, per-thread, is unrealistic for large applications. It seems to have been the plan of the developers to allow users to develop their own GUI for analyzing and visualizing probe output files, just as they state users can design their own probes and use whatever output they desire. Probably the major impediment to using this tool is that its functionality is included in most other, fuller featured performance analysis tools.

4.5.2 mpiP / ToolGear

ToolGear is a software infrastructure created here at LLNL to assist development of tools for ASC code projects. mpiP is a lightweight communication profiling library for MPI applications that is developed and distributed by LLNL. mpiP’s lightness is attributed to the fact that it only collects statistical information about MPI functions, and therefore generates considerably less overhead and much less data than tracing tools. Furthermore, all the information captured by mpiP is task-local. It only uses communication during report generation, typically at the end of the execution, to merge results from all of the tasks into one output file.

mpiP can be used with C/C++ and FORTRAN programs on Intel 32-bit Linux, IBM AIX and HP Alpha Tru64 platforms. Using mpiP is simple and straightforward and only requires linking an application with the required libraries. Following execution and the automatic merging of the task-local information, mpiP will produce a single output file in the format of a plain text report. The report has five sections including a header with basic program information, the call site section that shows every place where MPI calls are made, aggregate time for the top 20 MPI call sites, aggregate message size for the top 20 MPI call sites and then a final section that details statistics for every MPI call site across all tasks. Additionally, mpiP statistics can be viewed graphically through the ToolGear mpiP viewer and through the ParaProf tool.

mpiP collects statistics for most relevant MPI-1 routines, and a subset of relevant MPI-2 I/O routines. It has proven to be scalable with benchmarks run on LANL's Q machine (3584 tasks) and LLNL's ASC White machine (4096 tasks).

mpiP is extremely simple to use, scalable and provides useful statistics for an application's MPI usage. It can easily help find an application's most costly communications. The recently introduced functionality within ToolGear and ParaProf for graphically viewing mpiP profiling is another plus. Because mpiP is a locally developed and supported tool, there is virtually no chance of it disappearing against our wishes. Expanding mpiP to include more MPI-2 support would improve its usefulness to those programmers who use such routines.

4.5.3 MPX / ToolGear

MPX is an LLNL developed tool used to multiplex PAPI hardware performance counter events. Although PAPI allows programs to request measurements for combinations of hardware counter events (such as loads and Flops), it does not allow users to request combinations that are not supported in the underlying hardware (e.g., loads and L1 cache misses on the PowerPC 604e). Multiplexing software helps overcome this limitation by time-slicing the hardware counters.

The MPX user interface is modeled on the PAPI interface. The user passes MPX a list of events to measure (for a given thread), and MPX measures each event in turn for some period of time. MPX also keeps track of how long each event was measured, and it uses this information to compute an estimate of the total number of times each event occurred during the measurement period. Thus, MPX values are only estimates, unlike PAPI values, which are exact. In most cases, MPX estimates are within a few percent or less of actual values, but the estimates can be much farther off if the total measurement period is too short (roughly 10 milliseconds per measured event) or if the application has highly tuned L1 cache performance, since the multiplex software pollutes the cache somewhat.
MPX can be used with C/C++ and FORTRAN serial and parallel MPI programs on 32-bit IBM AIX platforms. Similar projects include the TULIP counters for lightweight threads at LANL, and the Sphinx project for mixed MPI and threads models.

MPX has recently been implemented within LLNL's ToolGear infrastructure, which loads and controls the program, manages data, and presents the user interface. Within this infrastructure, MPX works like a debugger. You can load and execute an application, view the source code, and set and remove instrumentation points (instead of breakpoints)

without modifying the original program. However, unlike a debugger, the tool does not let you examine variables or single-step the program.

MPX's primary value is that it enables the profiling of PAPI events that are otherwise not compatible. Within the ToolGear framework, an added benefit is ease of use through point-and-click instrumentation and visual feedback on hardware performance counter statistics. MPX's primary drawback is that the only platform it can be used on is 32-bit IBM AIX, which is a nearly obsolete platform now, and that there appear to be no plans to further develop the software or port it to other platforms.

4.5.4 VTune

VTune is a sophisticated and full featured performance analysis tool from Intel for use on 32-bit and 64-bit (Pentium/Itanium) Linux and Windows systems. Vtune consists of two primary components – a sampling collector agent and the analyzer GUI, and can run in two different modes. In "native" mode, both the sampling collector agent and the GUI run on the same system. In "remote" mode, the sampling collector agent runs where the target application is executing and the GUI runs on another system, connected via a network. Vtune also provides a command-line interface if use of the GUI is not desired. Vtune's is rich in features, but its main functionality falls into three main categories: sampling, call graph profiling, and the Intel Tuning Assistant.

Sampling data is collected and displayed in real-time, so there is no need for the creation of huge trace files. Sampling data is collected system-wide, which means that it reflects more than just the target application. Sampling features include the ability to conduct both time-based sampling and event-based sampling. In time-based sampling, the sampling collector periodically interrupts the processor to collect information. Event-based sampling is triggered by specific events, such as L2 cache misses, cache misses, etc. Two powerful sampling features include the ability to sample processor events at the system-wide level, which reflects the activity of all processes running on a processor, and the ability to track events at the source statement level. Sampling data can be sorted, filtered and viewed in a number of ways through the GUI. Additional benefits are that source code does not have to be modified, sampling overhead is very low, and that sampling data can be viewed at the process, thread or module level.

Vtune's call graph profiling is a very useful component of the tool. It tracks each function's entry and exit points at run-time. Because Vtune uses binary instrumentation, there is no need to modify source code. Unlike sampling, call graph profiling is specific to the target application – it is not system-wide. Call graph profiling data is graphically portrayed by the GUI, and includes the ability to show the critical (most time consuming) path, filtering, and self-time, total time and number of calls per function. Navigating complex and large call graphs is made easy through a call graph overview window. The GUI can also display call graph data in a tabular text format if desired. Finally, users have the ability to override the default collection of call graph data for all functions, and can select only those functions they are interested in profiling.

The Intel Tuning Assistant is an advanced and potentially very useful component of Vtune. It is able to automatically detect performance bottlenecks and hotspots. Furthermore, the Tuning Assistant is able to provide possible explanations for both of

these, and make recommendations on how to improve the code's performance at these points. All of this is done through the GUI.

Until recently, Vtune was strictly a single node tool – not enabled for multi-node MPI programs. Intel now has a Linux platform, MPI enabled version of Vtune that has not yet been evaluated by the lab due to its newness.

Vtune is without a doubt one of the best performance analysis tools on the market. The fact that it is low overhead, does not require source code modification, and provides such rich and useful features are definite assets. It is greatest drawback however, is that it is supported only on Intel 32-bit and 64-bit systems and Intel will probably never support it on non-Intel architectures. This is a fairly substantial drawback for the Tri-lab users who are accustomed to running on multiple architectures and porting to new systems.

The new MPI enabled version of Vtune is potentially exciting, particularly if it proves to be low overhead, full-featured and robust. This has yet to be determined however. Very unfortunately, it seems as if Intel might halt or delay its plans to develop a parallel version. This would be a great shame since the HPC community seems quite anxious for this capability in this tool.

4.5.5 prof & gprof

prof and gprof are two basic profiling utilities that are available on most UNIX systems, such as IBM AIX, Sun Solaris and HP/Compaq Tru64. On SGI Irix, the Speedshop profiler includes a prof utility that is functionally equivalent to both prof and gprof found on the other UNIX systems, however it's usage is slightly different than what will be described here. Linux systems have only the gprof utility, which is the same as that found on UNIX systems.

Using prof and gprof is a simple process that does not require modification of source code. The user simply compiles the program with the appropriate flag (-p for prof and -pg for gprof). When the user program is run, a monitor process is automatically started. The monitor periodically interrupts the program and logs the location of the program counter. Following execution, a binary output file (or multiple files for parallel programs) is produced, named mon.out for prof, or gmon.out for gprof. The prof/gprof utility is then used to read the binary output file to produce a human readable, plain text report.

The prof report consists of a flat statistical profile of the CPU time used by each routine that a program calls, including system and library routines. It is sorted by CPU time with the most CPU intensive procedures appearing at the top of the report. Several columns of information are presented, and include metrics such as actual CPU time used, percentage of the program's total CPU time used, the number of times the routine was called, and the average time in milliseconds for a call to each routine. The gprof report provides all of the information of the prof report plus a call graph (call tree) profile. The call graph profile shows the relationship between called and calling routines, that is which routine called which other routines, thus providing more detailed information than prof does.

Both prof and gprof can be used with C/C++ and FORTRAN programs. Depending upon the implementation, these tools may also be used with parallel threaded and MPI programs.

Both of these utilities are quick and easy to use, and for the most part, are portable across most UNIX-type systems. Depending upon the implementation, they have been used successfully for medium-scale (hundreds of tasks) parallel jobs. There are a couple drawbacks to both utilities however, such as the possibility for phase problems to occur which cause disproportionately high, or low, execution times to be reported. In addition, since they use sampling to collect data, they only provide CPU-time, not wall-clock time, and cannot provide useful MPI or I/O information. Scalability for large parallel jobs (thousands of tasks) presents another problem with time-consuming generation of the output files.

Gprof can also be used in conjunction with gcov, GNU's code coverage analyzer, to tell you which lines were executed how many times and for how much of the time. However, gcov is usable only with the gcc compiler. Other code coverage tools available are purecov from Rational, and McCabe's code coverage tool.

4.5.6 Xprofiler

Xprofiler is a graphical profiling tool that was formerly provided as part of IBM's Parallel Environment software on its AIX SP platform, but is now part of the AIX operating system software. Xprofiler is actually a GUI for the well-known UNIX gprof profiling utility, which has been enhanced by IBM to work with multi-process MPI programs. Xprofiler can be used with either serial or parallel MPI applications written in C/C++, FORTRAN or mixed language.

Using Xprofiler begins with compiling the application as one would for gprof by using the –pg compiler/linker option. Additionally, the –g option is required if profiling at the source line level is desired. Following compilation, the program is run as normal to produce the usual gmon.out file, or in the case of multi-MPI programs, multiple gmon.out files, one per MPI task. The Xprofiler GUI can then be started from the command line, supplying it with the name of the desired gmon.out files as arguments. The specified gmon.out files are then digested by Xprofiler and used for analysis in the GUI. Alternately, Xprofiler can be started alone and the desired gmon.out files selected through the GUI's menus.

The GUI itself is rather simple. For the most part, it consists of one large window with several pull-down menus. Within the large window, the execution call graph of the application is shown as green function boxes and connecting blue arcs. Each profiled function comprises a green box that is sized according to the amount of CPU time it used: bigger green boxes depict functions that used more CPU. The blue arcs that connect boxes describe the caller/callee relationship between functions. The entire display can be zoomed-in or zoomed-out as desired. For convenience and clarity, functions are grouped into cluster boxes. All of the functions within the application are grouped into a single box, as are those of each library used by the function. In this way, "clutter" can be reduced by easily removing uninteresting library clusters from the viewing area. Undesired clutter can also be removed by filtering – selecting only those functions of

interest, such as by function name, CPU time or counts. In addition to graphically depicting the application's call graph, Xprofiler provides all of gprof's usual flat text reports, which can be printed/saved from the pull-down menus. Finally, Xprofiler has several user-configuration options that can be stored in a configuration file.

As a GUI for gprof, Xprofiler is a single purpose profiling tool. It is easy to learn and use and provides a quick way to graphically view gprof performance information. There is no overhead imposed by this tool since it simply uses post-execution the gmon.out files produced by gprof. Gprof itself, however, can introduce significant overhead when profiling an application, oftentimes with its monitor function being the lead CPU consumer under AIX. Since it uses sampling to collect data, it only provides CPU-time, not wall-clock time, and cannot provide useful MPI or I/O information. This tool breaks down at large scale, as gmon.out files are generated serially. For large jobs, this process can be prohibitively time consuming and has also been known to hang. Although gprof is available on other platforms, Xprofiler is not.

There are two major drawbacks to the Xprofiler tool however, with the most obvious one being that it only works on the IBM AIX platform. The second drawback is that large complex applications can overwhelm the GUI and make visual analysis difficult, tedious and perhaps useless. For example, a representative B-division code calls over 5000 routines. The number of green boxes and blue arcs this requires necessitates creating a huge "spider web" display, a very small portion of which fits into the GUI window when zooming into a human-readable level. Tracking the caller/callee paths requires scrolling all over the map. Of course, filtering out functions by CPU time or some other parameter can be used to simplify the display, but at the cost of removing call graph information. Overall, Xprofiler has limited usefulness beyond standard gprof for large, complex HPC applications.

4.5.7 DEEP/MPI

DEEP/MPI is a commercial parallel program analysis tool from Veridian/Pacific-Sierra Research. DEEP stands for DEvelopment Environment for Parallel programs. DEEP/MPI provides an integrated GUI for performance analysis of shared memory (threads), distributed memory MPI, and hybrid (shared + MPI) parallel programs. To use DEEP/MPI, one must first compile the MPI program with the DEEP profiling driver mpiprof. This step collects compile-time information and also instruments the code. After executing the program in the usual manner, the user can view performance information using the GUI. Supported languages include C/C++, FORTRAN and mixed. Supported platforms include Linux x86, SGI IRIX, Sun Solaris, IBM AIX, Windows NT x86.

The DEEP/MPI GUI includes a call tree viewer for program structure browsing and tools for examining profiling data at various levels. It displays whole program data such as the wallclock time used by procedures. After identifying procedures of interest, the user can bring up additional information for those procedures, such as loop performance tables. The DEEP Performance Advisor suggests which procedures or loops the user should examine first. MPI performance data views allow users to identify MPI calls that may constitute a bottleneck. Clicking on a loop in a loop performance table or on an MPI call site takes the user to the relevant source code. CPU balance and message balance displays show the distribution of work and number of messages, respectively, among the

processes. DEEP provides PAPI hardware counter support and can do profiling based on any of the PAPI metrics.

4.5.8 VProf

The Visual Profiler, VProf, is a basic profiling tool that can be used with serial and parallel MPI programs written in C/C++ or FORTRAN. VProf samples clock ticks and PAPI hardware counter events. VProf is developed and distributed by Sandia National Lab. Using VProf is simple and straightforward. It requires only static relinking of an application that has been compiled with normal optimization options and the "-g" option. The application is then run to collect profiling data, which include clock ticks and PAPI hardware counter events. Profiling data is written to vmon.out files, one per process. Vmon.out files are then used to generate performance summaries sorted by source code line, by file, and by function. This information can be displayed either with the graphical user interface (vprof) or to the command-line interface (cprof). VProf runs on Intel Linux, IBM AIX and HP Alpha Linux platforms, and should also run on most other UNIX-like systems.

Vprof's online documentation leaves something to be desired, and it does not seem to be actively being developed and supported anymore.

4.5.9 Hiprof

Hiprof (Hierarchical instruction profiler) is used to generate profiles of a program's execution time based on its procedure call graph. Hiprof is one of the ATOM-based tools found on HP/Compaq Tru64 systems. ATOM tools work by taking a non-stripped executable and generating a new executable that can analyze itself as it runs. Using this command line tool is simple. One first compiles their code (C/C++ or FORTRAN) with the -g option to produce an executable. The hiprof command is then called using the name of the executable as its argument. It produces an instrumented version of the executable that is then run to produce a binary output file. Finally, this output file is provided to the gprof utility to produce a human-readable profile report. Note that creating the instrumented executable, running it and displaying the output can all be done in one step by hiprof if desired.

Hiprof can profile numerous metrics. One is the CPU time spent in each procedure (or optionally, each source line instruction), measured by sampling the program counter about every millisecond. Another metric is the CPU time spent in each procedure and procedure call, measured as machine cycles, including the effects of any memory-access delays. Also, the number of page faults suffered by each procedure and procedure call can be profiled. Hiprof also allows you to combine profiles from multiple runs, dump profile information from a running process with out killing it, and to profile only part of a run. This tool can be used for parallel threaded programs (pthreads, OpenMP) and for parallel MPI programs, however combining both MPI and threads is problematic. In addition, there are some restrictions on the data that can be collected for parallel programs. For example, page faults and shared library profiling are not supported.

Hiprof's output is plain text reports, similar to gprof, and in fact, hiprof actually uses gprof to produce its reports. This is no coincidence since both hiprof and gprof use the same monitor produced output files for their reports. HP's dxprof tool provides a GUI for analyzing data from hiprof.

Hiprof claims to produce more reliable measurements than traditional gprof does. Gprof estimates how much of a procedure's execution time was spent on behalf of each caller, based on the execution frequency of each call site. This estimate is usually good, but is sometimes completely wrong. In contrast, hiprof directly measures the time spent for each call, and does not need to guess. However, this tool's most obvious drawback is that it is only available on Tru64 systems.

Note that this tool should not be confused with the similarly named, but very different, commercial tool from Tracepoint, called HiProf, which is used for the same purpose on Win32 systems.

4.5.10 Pixie

Pixie is another HP/Compaq profiler tool, similar to prof, gprof and hiprof. In fact, pixie is used in a manner that is practically identical to the hiprof utility. By calling a properly compiled executable with pixie, an instrumented version of the program is created, which can then be run to produce a binary output file. The binary output file can then be used to produce a human-readable statistical report, either automatically by pixie or manually via the prof utility (in either case though, prof is actually used to produce the report).

The primary difference is that pixie reports instruction counts instead of CPU time. Instruction counting can be performed at the procedure level and at the source line level, which is particularly useful in isolating the most time-consuming subroutines, loops and individual instructions of a program. Both user code and shared library code can be profiled. Like hiprof, source code can be C/C++ or FORTRAN, and parallel via threading or MPI. Also like hiprof, there are some unsupported features for parallel profiling. Additionally, there are tools (dxprof, Cvperf) that provide a GUI for viewing pixie data.

Pixie's information can also be used to help the compiler perform additional optimizations when used with recompilation options such as -cord or -om, or with reordering tools like spike. These tools reorder the application's procedures so that the most frequently executed instructions are stored in the fast memory cache. The data can also be used for coverage analysis.

Pixie is simple and easy to use, and offers a particularly nice way to isolate sub-procedure sized sections of code (loops, individual instructions) for performance analysis. Like hiprof though, its major drawback is that it only runs on Tru64 platforms. Pixie's ability to provide information to the compiler's optimizer is also a nice feature. Experience here has shown though such analysis and reordering is often highly problem dependant and not very applicable to our large code projects.

4.5.11 Perfex

Perfex reports the hardware counts of selected events on SGI R10000 platforms. A nearly identical tool called lperfex exists for Intel-Linux IA32 systems as well. You can profile either the whole program's event counts or only the event counts of a selected small section of your program. You can get the exact counts of two select counters or you can get the average counts of 32 events with some statistical error. For exact counts, you must specify which two events you want counted. You can also manually instrument your source code to limit which parts of your code you wish to profile. Time spent in each event counter is also available. Unfortunately, you cannot apply perfex selectively to a run. It must be active for an entire run.

4.5.12 PapiEx

The PapiEx tool is very similar to Perfex and pfmon, but is easier to use. It is a part of the newer PAPI version 3 distribution. It is a performance analysis tool designed to transparently and passively measure the hardware performance counters during a run. However, it cannot selectively instrument an application. Instead, it measures the entire run. For selective instrumentation, use DynaProf or psrun. PapiEx can optionally monitor all subprocesses and threads. It can also do counter multiplexing and a host of other nifty features. It uses library preloading to intercept process and thread creation. To instrument your code, it must be linked in as a shared library. Unfortunately, it does not support AIX platforms.

4.5.13 Tprof

The tprof command is an IBM AIX profiling utility that reports CPU usage for individual programs and/or the system as a whole. CPU time can be profiled for object files, processes (system wide), threads, subroutines and even for individual program instructions. Profiling is very low-overhead because it depends upon the AIX trace utility which automatically executes 100 times per second. Tprof usage varies depending upon the type of profiling desired. To profile any object file, including shell commands, user programs or UNIX commands at the process level, one simply invokes tprof with the object file name at the command line. Following execution of the command, a plain text output file will be produced which is tprof's report of CPU usage. The contents of the report depend upon the options tprof was invoked with, but generally include total CPU time for each process/thread, CPU time by user, kernel and shared library, and a frequency count. To profile user codes at the subroutine or instruction level, the -g compiler flag must be used. This utility can show CPU statistics for each thread in a multi-threaded code, but has not been implemented for parallel MPI programs.

Tprof presents some very useful features, such as being able to clearly see what other "system" processes are doing while the user application is running, and the ability to effect source line level profiling. It is easy to use and is very low-overhead. However, it is only available on IBM AIX platforms and it has not been implemented for parallel MPI programs. The latter obstacle can be overcome if the user places executables in processor local directories (such as /usr/tmp), which would result in unique tprof output files being written locally on each processor. The user would then need to manually "collect" each output file following program execution.

4.5.14 SCALEA

SCALEA is a performance instrumentation, measurement, analysis, and visualization tool for parallel FORTRAN programs. It can analyze OpenMP, MPI, or mixed codes. They also have a version for grid computing. It supports multiple experiment performance analysis that allows it to compare and to evaluate the performance outcome of several experiments. Their SIS tool instruments programs. It allows the user to select the code regions and performance metrics of interest, but it is integrated with the VFC compiler. It has a profiling library while hardware parameters are determined through an interface with PAPI. This tool is part of the Austrian based Askalon tool set for cluster and grid computing. Other tools of interest in the suite, which implement SCALEA, include ASKUM, an automatic performance analysis tool, and Performance PROPHET, a modeling and prediction system.

4.5.15 Pgprof

Pgprof is an interactive postmortem statistical analysis tool for MPI and OpenMP parallel C, C++, and FORTRAN applications on Linux clusters, including those with 64-bit processors. It illustrates the frequency and duration of your function calls down to the source line level. It also illustrates MPI communication and thread profiling, and can measure scalability between multiple executions. Its GUI can display statistics as percentages, bar charts, or absolute values, and sort them by name, value, or time. It can even be applied to optimized executables at a coarser level. Recompilation is necessary to instrument your code. Although it operates on MPI and thread-enabled codes, its parallelism, however, is quite limited.

4.5.16 IPM

IPM is NERSC's new lightweight MPI communications profiler. It accomplishes this by using a hash-based approach instead of histogramming, resulting in a smaller footprint while reducing sampling error. Its sampling of a run's message passing statistics has a low enough overhead that this tool can be applied automatically to all parallel executions. The combined data from all these runs could be beneficial to system operators to inform them how the machine is performing or how it is generally being used. Optimizing the system for this average case could then be done for an overall improvement in machine utilization. It can also be used to show the effects of different versions of MPI, network hardware upgrades, or certain system environment settings on communications. Analyzing the data for a specific executable can also be done, although since its profiling summary applies to an entire run, it does not correlate performance bottlenecks to specific areas of code. IPM is still in an early development stage, and will eventually offer some tracing ability in addition to profiling. It may become like the computational grid system analysis tool MAGNET (Monitoring Apparatus for General kerNel-Event Tracing), which allows monitoring of OS kernel events on nodes of a cluster or grid.

4.6 Performance Analysis Tools: Tracers

4.6.1 RootCause

RootCause, based on Aprobe, is a sophisticated tracing tool that, thru a GUI, a user can selectively choose the data to be collected or navigated. It attaches to a program at runtime, without requiring any changes to the application. Instrumentation happens in memory automatically and happens during the execution without any modification to the disk-resident version of the application. The tool dynamically inserts probes into an application to collect data. The traces can be conditional, contingent upon certain events, or data-dependent. Additional information, such as timing and memory tracking, can also be stored. It is used primarily for end-users to send collected data back to developers when reporting a problem that eliminates the need to recreate the events or even ask further questions. It provides a mechanism for a snapshot to be taken programmatically. The designers equate this tool to a flight recorder for software. I figure that this feature would be handy if applied to a crashed MPI task. This tool is not parallel-aware but the designers say it could be applied to every process individually. Then just the crashed task needs analysis. Although only C/C++ and Java are officially supported, they have had some success under FORTRAN. However, it does not work well with NFS storage of its log files.

4.6.2 Mpitrace

Mpitrace is a very low overhead library that reports the elapsed time spent inside each MPI communication routine. It incorporates a direct method to convert timetable structures into seconds, which is much faster than using other available conversion routines. It is available only for AIX systems. Additionally, there is an mpihpm library which includes these trace wrappers with the HPM Toolkit counters to collect data on Power-4 architectures. Unfortunately, this library is not thread-safe.

4.6.3 Perfometer

Perfometer is the real-time performance monitor distributed with PAPI. It provides a fast coarse-grained easy way for developers to spot performance bottlenecks. The GUI can display real-time FLOP rate performance characteristics, or write a trace file. The source code needs to be modified for proper instrumentation. Significant expansion of its capabilities have been introduced with recent PAPI releases

4.6.4 SiGMA

SiGMA (Simulation Guided Memory Analyzer) is a toolkit for analyzing bottlenecks and inefficiencies due to the memory hierarchy. It provides detailed information about the memory subsystem useful for understanding the cache behavior of algorithms and the codes interaction with memory. More uniquely, it provides an infrastructure for asking “what-if” questions for data structure and other parameter perturbations that could improve performance. There is a significant impact on performance while using this toolkit, and unfortunately, it can operate on only one processor currently. However, improvements are in development to evolve it into a robust and effective middleware

layer standard for HPC tools. It will also likely be renamed. It works only on AIX platforms and recently become part of the new IBM HPC Toolkit (as did HPM Toolkit).

4.7 Performance Analysis Tools: Visualizers & Other Analyzers

4.7.1 Vampir / Vampirtrace / VampirGuideView (VGV)

Vampir and Vampirtrace are complementary parallel performance analysis tools formerly developed and marketed by Pallas GmbH in Bruehl, Germany. These tools work together to provide one of the best, and probably most widely used, MPI performance analysis toolkits available to code developers. The Vampir / Vampirtrace toolkit can be used with C/C++ and FORTRAN programs, and is also very portable, supporting most popular HPC platforms ranging from Linux PC's to teraflop computers including Intel, Compaq, Cray, Fujitsu, Hitachi, HP, IBM, NEC, Scali, SGI and Sun. It is available on our AIX, Intel-Linux and Tru64 platforms. Unfortunately, this commercial tool set was recently acquired by Intel and its support of future platforms is in question.

As its name implies, the Vampirtrace component is used to trace program execution. It consists of an instrumented MPI library that is linked into a user's code to automatically generate a set of trace files that describe a program's run-time behavior. Vampirtrace records all calls to MPI routines, including point-to-point as well as collective communication. In addition, arbitrary application-defined events can be defined and recorded, such as entry and exit of subroutines or code blocks. Trace data collection can be dynamically switched on or off during runtime, and a configurable filtering mechanism helps to limit the amount of trace data and focus on relevant events. In order to minimize instrumentation overhead, trace data is kept locally in each processor's memory, and then post-processed and saved to disk when the application is about to finish. Vampirtrace is also able to automatically correct clock offsets and skew on systems without a globally consistent clock. Using Vampirtrace to record MPI events requires nothing more than re-linking the application with the Vampirtrace library. To trace arbitrary application-defined events usually requires insertion of Vampirtrace routine calls in the source code and recompilation. Vampirtrace is completely thread-safe, which is a decided plus for multi-threaded MPI programs.

The Vampir component consists of a very full-featured GUI that is used to graphically display event information captured in Vampirtrace trace files. It provides an effective means for users to understand their application's overall behavior, evaluate load balance, identify communication hotspots, and analyze communication patterns and performance. This is accomplished through a number of different activity and summary displays including detailed timeline views of events and parallel communications, statistical analysis of program execution, statistical analysis of communication operations, system snapshot and animation, dynamic calling tree and more. Most displays are available in global (entire program) and per-process modes. Navigation of trace files, and zooming in to increase detail is easily accomplished and is one of Vampir's strong points. It also displays activity and summary charts, and can report message and file I/O statistics. Context sensitive menus are also provided. In the most recent versions of Vampir provided to LLNL, the GUI has been enhanced by parallelizing it, making analysis of large trace files much faster and more user-friendly.

VampirGuideView (VGV) is a synthesis of Pallas' Vampir / Vampirtrace product with the Intel KAI lab's KAP/Pro Toolset for OpenMP analysis. Development of VGV was in part, funded through the ASCI PathForward Parallel System Performance Project. An implicit goal of this project was to accelerate the development and commercialization of a scalable performance analysis tool that could be applied to hybrid programs within the Tri-lab environment. Hybrid programs use both distributed memory MPI and shared memory OpenMP parallelism. In addition to providing hybrid performance analysis capabilities, VGV acquired new functionality, such as support for hardware performance counters through PAPI, support of the platform independent OpenMP POMP performance interface, and implementation of an application statistics profiler. The VGV development effort was successful in the fact that it delivered such a tool, which showed increasing scalability as the project progressed. However, VGV never became an actual product, and its future is at best uncertain for reasons concerning Intel's acquisition of it.

At this time, the single greatest concern for Vampir / Vampirtrace / VGV arises from Intel's acquisition of Pallas' development team in September 2003. At that point, Vampir / Vampirtrace became products of Intel, and were renamed Intel Trace Analyzer and Collector, although these incarnations lack VGV's threads support. The follow-on to VGV within Intel has not yet surfaced. Support for existing non-Intel platforms is expected to be short-lived, and development for new non-Intel platforms is not anticipated. It will be unfortunate to lose these tools on non-Intel platforms. Although there is rumor and speculation that the original, pre-Pallas developers may keep Vampir / Vampirtrace alive as some similar multi-platform toolkit in the future, a replacement should not be expected anytime soon. The future for VGV seems even more uncertain, even though Intel has acquired the development teams of both KAI and Pallas who authored this software. It would be quite a shame to loose this tool, as it has already proven its worth to many of our large parallel codes.

Lesser concerns include those shared by all trace analysis tools. With real applications, trace files can easily become prohibitively large, and the information displayed can become muddled, especially at large processor scales, making it impossible to use the GUI for viewing and analysis. The most common way of dealing with this problem is to only collect detailed trace information for small sections of code, or by turning event capture on/off only in selected locations. Fortunately, this is achieved by selectively instrumenting the code. Another concern is the relatively steep learning curve, which again, is a concern shared by most performance analysis tools as full-featured as this one, although for this tool it seems larger than most.

4.7.2 Paraver

The European Center for Parallelism of Barcelona (CEPBA) develops and distributes a multi-platform, parallel performance visualization and analysis tool called Paraver. This highly graphical tool is similar to VGV in a number of respects. It supports MPI, OpenMP, and hybrid programming environments on AIX, Tru64, Linux, and Irix platforms, in C/C++ and FORTRAN. Some of Paraver's key features include both quantitative and qualitative displays for message passing activity, hardware performance counters, and operating system activity. It can profile hardware counters per function, provide histograms of parallel functions duration, and show timelines of task-to-

processor mappings. It displays a profile of the parallelism, CPU consumption, and communication load useful for load balancing different parallel loops. One of Paraver's advanced features is the ability to compare two trace files, something that is useful for comparing code versions, machines, scalability, or even problem size effects. Paraver's ability to display and analyze operating system activity can be very useful when attempting to understand an application's performance in the complete context of the machine it is running on. User customizations and preferences to Paraver displays can be stored by means of a configuration file. For those who do not wish to use the GUI interface, Paraver's functionality is available through the Paramedir tool, although the GUI is intuitively easier to use.

Paraver's GUI depends upon trace files produced by instrumenting an application's execution. There are several utilities that may be used for this purpose, all of which are downloadable from the Paraver website, including OMPtrace, MPItrace, OMPItrace, and SCPUs. Instrumentation occurs dynamically by running an application under one of these utilities, resulting in trace files that can then be merged into a single Paraver trace file. Currently, these utilities have only been developed for SGI IRIX and IBM AIX platforms. Non-Paraver traces produced by the IBM AIX trace utility and the IBM UTE utility can be converted to Paraver format through two conversion tools also available for download. Paraver also reads trace file data from another CEPBA tool called Dimemas.

Dimemas is a simulation tool that reconstructs the behavior of a parallel machine modeled by performance parameters so that portability and scalability experiments can be performed. It enables the user to develop and tune parallel applications on a single-CPU workstation, with the goal of predicting performance on the parallel target machine. The supported target architecture classes include networks of workstations, single and clustered SMPs, distributed memory parallel computers, and even heterogeneous systems. Dimemas generates trace files that are suitable for both Paraver and Vampir, either of which may then be used to examine the performance characteristics indicated by a simulator run.

As a trace-driven, parallel performance analysis tool, Paraver shares several important concerns with similar tools. Trace files for real HPC applications can become prohibitively large, and the tool's scalability is poor with large numbers of processes. Additionally, there is a significant learning curve necessitated by Paraver's complexity. Another important concern relates to what appears to be halted development. Documentation for Paraver and its associated tools dates to 2000-2002 on their web site. Within that documentation, there is mention of the trace generation utilities being ported to architectures besides SGI and IBM, however there is no indication this has ever happened, and versions of these tools for other architectures are not available for download. The absence of a Linux version of the trace generation utilities is particularly obvious. Finally, although Paraver and its related tools seems to be known generally within the HPC community, actual use of them seems minimal, particularly within the Tri-lab sphere.

4.7.3 Jumpshot & MPE

Jumpshot is a Java-based visualization tool for doing postmortem performance analysis for serial, MPI and threaded programs written in C/C++ and FORTRAN. Jumpshot is

developed and made freely available from Argonne National Lab and has been in existence for a number of years now, with continuing improvement in each new version. The most recent version is Jumpshot-4. The GUI now uses Java instead of Tcl/Tk to improve its portability, maintainability and functionality. It has also been redesigned to use the SLOG-2 scalable logfile format. This new file format allows logfiles to be scalable into the gigabyte range. It also allows the Jumpshot viewer to provide functionality never made possible before. For example, level-of-detail support through preview drawables provides a high-level abstraction of the details without reading in a huge amount of data into the graphical display engine.

Other new features include seamless scrolling from the beginning until the end of the logfile at any zoom-level, dragged-zoom, instant zoom in/out, grasp and scroll, easy vertical expansion of the timeline, timeline manipulation, and the new Legend table that provides a central control for both the Timeline and Histogram modules. Additionally, a new search/scan facility is provided to locate hard-to-find objects in very large logfiles, and a graphical analysis of MPI overhead in user MPI applications. Still, this tool's features list seems inferior to the capabilities provided by VGV.

It works effectively with data from large scale and long-running jobs. It can also provide an estimation of your application's MPI communication overhead. It simplifies data presentation with a preview display that facilitates navigating lengthy timeline histories.

4.7.4 Paradyn

The Paradyn parallel performance analysis tool comes from of an ongoing research and software development project originating at the University of Wisconsin. Paradyn's key feature is that it is able to collect performance data dynamically during run-time execution. Because performance analysis data is collected at run-time with binaries, source code does not require modification, which is an added benefit. Furthermore, the selection of performance data is user driven. Paradyn offers multi-platform support including Solaris (SPARC), Linux (x86), Windows NT/2000 (x86), and AIX (RS6000), and also, heterogeneous combinations of these systems. Multi-threading support is offered on the Solaris and AIX platforms. Paradyn's dynamic instrumentation is built upon Dyninst, an API for run-time code generation, and also the result of the group that created Paradyn. Several other unrelated parallel performance analysis tools have been developed using the Dyninst API.

Paradyn is able to monitor program performance according to 20+ different metrics. These include statistics for CPU utilization, I/O activity, MPI message passing, function calls and synchronization. The GUI's display of these run-time metrics takes the format of user selected visuals, such as histograms, bar charts, 3D terrain and tables. Data can be represented chronologically at a global level (entire program timeline) or in finer detail as a local phase, which is user defined. The GUI interface requires Tc/Tkl.

Paradyn has the ability to automate much of the search for performance bottlenecks. The Performance Consultant is a sophisticated utility featured within Paradyn that enables automatically determination of the where-when-why of performance problems. It is designed to eliminate the guesswork of manual problem determination methods. In its normal mode of operation, a user simply tells it to start searching for performance

problems. The Performance Consultant will continually select and refine which performance metrics are enabled and for which foci they will be enabled. This involves building a hierarchy of hypothetical possible causes and then evaluating and exploring the hierarchy with real-time instrumentation and analysis.

Documentation for this tool is better than most other similar, non-commercial tools. In addition to a substantial User's Guide, a tutorial is provided, as are installation and release documents.

For the purposes of user expansion, Paradyn defines an API that allows users to add new run-time visualizers and external analysis tools that use Paradyn performance data.

Paradyn's documentation is as good as or better than similar tools, and as part of that documentation, the limitations of the tool are noted. In particular, there are quite a number of details associated with support for specific architectures. Users and developers alike will want to read this information. Installation details follow this suggestion also. A number of other caveats are documented, many of which are explained as being resolved in a future release.

One significant limitation of Paradyn is the nature of its MPI support. In most cases, only MPICH is supported. Support of vendor MPI is the exception, with IBM being the sole case. Another current deficiency is the inability to "detach" from a monitored program without killing all processes associated with it. This limitation is targeted for removal at a future release.

This software project has been existence for close to 10 years, and has undergone continued development and improvement. The development team has published a number of papers and presented the tool at relevant venues. One might anticipate upon these facts that the project possesses some longevity worth considering for future engagements.

Paradyn documentation claims that it scales to long running programs (hours or days) and large (thousand node) systems, that it automates much of the search for performance bottlenecks, and that it can provide precise performance data down to the procedure and statement level. Notably, its overhead cost can be limited to a user specified threshold. Investigation of these important features seems like a worthwhile endeavor for the Tri-lab tools staff.

4.7.5 SvPablo

SvPablo is a graphical performance analysis tools that originates from the Pablo Research Group at the University of Illinois, Urbana-Champaigne, with funding provided by NASA, DOE and DARPA. Pablo's main features include source code instrumentation (interactive or automatic), performance data capture at the routine and outer loop level, browsing and analysis of performance data, collection of PAPI hardware performance counter data, statistical summaries for long-running codes (no traces), and an option for real-time data transmission via its associated Autopilot tool. Autopilot is an infrastructure for real-time adaptive control of parallel and distributed computing resources. SvPablo is also able to collect performance data on MPI-I/O routines and UNIX I/O. SvPablo

supports C and FORTRAN serial and parallel MPI codes on Sun Solaris, IBM SP, SGI Origin, HP/Compaq Alpha and Linux (IA-32 and IA-64) platforms. The latest versions also support threaded codes, and C++ support is under integration via ROSE.

Interactive instrumentation is performed through the GUI by simply clicking on the source code lines that contain instrumentable constructs such as procedure calls and outer loops. Automatic instrumentation is performed by selecting the appropriate options from the GUI's pull down menus. SvPablo's basic metrics for instrumented constructs include counts, inclusive duration and exclusive duration. PAPI events, which include multiplexing, are specified in a user configuration file. It uses statistical measurements, rather than detailed traces, for faster execution and can operate on problems taking days.

Instrumented code is then compiled to produce instrumented object code. Execution of the instrumented object produces per-task performance data output files. These files are then merged using the SvPabloCombine utility. The resulting file can then be used by the GUI to visualize performance data. SvPablo uses the Self-Defining Data Format (SDDF) for its merged performance data file. SDDF files can be either compact binary format or human-readable ASCII text.

As expected, the GUI provides a variety of displays. The ability to view multiple performance statistics graphically at the source level is one of the tool's most useful and unique features. Additional detail for each instrumented source line can be displayed also, including value, max, min, mean and standard deviation for each of the collected metrics across all tasks.

SvPablo has evolved over the years and now supports OpenMP codes and allows interactive instrumentation, which provides control over the instrumentation overhead imposed. Its scalability analysis and predictions features based on symbolic expressions derived from compiler-generated code have evolved into a separate tool called Delphi.

Unfortunately, this tool lacks support for C++ codes. Also, its instrumentation overhead can be significant. However, it has been an ongoing project for years and we hope they continue to make progress. Hopefully, analysis of the performance of MPI routines will be included in the future.

4.7.6 ParaGraph

ParaGraph is a graphical display tool for visualizing the behavior and performance of parallel programs that use MPI. It takes as input the execution trace data provided by MPICL, the MPI portable instrumented communication library, developed at Oak Ridge National Laboratory. MPICL instruments C or FORTRAN applications via the developer adding no more than a few statements into the source code in order to collect information. It can then collect profile data and analyze time spent in communication and user-defined events for each processor. It can also collect detailed traces of each event to be viewed with ParaGraph. Paragraph replays the trace data pictorially to provide a dynamic depiction of the behavior of the parallel program. Different visual perspectives are available to provide additional insight.

4.7.7 Opt

Opt is a new optimization and profiling tool from Allinea which will share the same look and feel as DDT and can also interface with it. It will be able to visualize MPI communications, highlight and analyze bottlenecks, display a call-graph, and provide hardware counter statistics via PAPI. No code instrumentation will be necessary. It is expected to be available in early 2005 for C/C++ and FORTRAN codes using MPI or OpenMP on AIX and Linux systems, including 64-bit Itanium-2 and AMD clusters. It will also be a grid-enabled tool. Their goal is to make it highly scalable, versatile, and intuitive. How it integrates with the DDT debugger and what new added benefits or conveniences this will provide are intriguing.

4.7.8 OptiPath

PathScale's OptiPath MPI Acceleration Tools promise to identify the root causes preventing applications from scaling on clusters. Not only will the tool rank the bottlenecks but also recommend how to improve the application's scalability. It will automate much of the analysis and even show "before" and "after" effects. It implements a guided problem/solution approach with comparisons to other runs which could vary by code changes, data sets, or even just scale. A complex series of experiments with variable parameters can be run with a single click, and the whole set of run can be analyzed together. In addition to a ranked list of bottlenecks with source code lines identified, it will suggest the root cause of the problems and how to fix them. Such automation of the result analysis is a welcome and long sought addition to any performance analysis tool.

4.7.9 SeeWithin/Pro

Verari System's new SeeWithin/Pro scalable performance analysis tool works with MPI applications written in C and FORTRAN on Linux and Windows NT/2000/XP platforms including Intel and AMD clusters. It can reveal hidden performance issues, provide hardware counter information via PAPI, analyze over a user specified interval, collect coarse grain trace data for large runs, and provide cook-book analysis for performance bottleneck detection.

4.7.10 Xmpi

Xmpi is a GUI tool for visualizing MPI communications. It is available only for LAM/MPI. Its basic interface and simple features do not lend themselves well to large-scale parallel runs. However, it is an excellent tool for teaching because it vividly shows the results of message-passing functions, such as by employing a stoplight process icon.

4.7.11 Dynamic Kappa-Pi

Dynamic Kappa-Pi provides parallel program analysis for MPI or PVM applications. Kappa-Pi is a research project from the Universitat Autònoma de Barcelona, and stands for Knowledge-based Analyser of Parallel Program Applications and Performance Improver. The primary goal of Kappa-Pi is to automatically analyze the performance of parallel applications, detect bottlenecks, explain their reasons and provide hints to the

developer on how to improve performance. Two approaches are used to accomplish this: the Static approach, based on trace files and source code analysis (Kappa-Pi), and the Dynamic approach, using "on the fly" analysis of run-time performance data, an application model and a static call graph (Dynamic Kappa-Pi). A closely related goal of the project is to produce a tool that is able to automatically tune the performance of a parallel application during its execution, without the need to recompile and rerun the application. Dynamic Kappa-Pi does not profile or trace applications itself, but depends upon a third party tool such as VampirTrace or Tape/PVM.

Currently, there is no production ready software product that might be employed here, or any hint of such being available soon. In fact, most web pages and papers that relate to this project fall within the 1998-2000 timeframe, with the Kappa-Pi homepage itself stating it has not been changed since October 1999. Although Kappa-Pi is documented as working with MPI programs, the limited documentation, which consists of several PowerPoint presentations and a few papers, presents studies done with PVM. PVM went out of vogue with the arrival of MPI over 10 years ago. Given these considerations, it would be impossible to realistically consider this software project for serious use in our environment, although its goals still have yet to be achieved by any other analysis tool.

4.7.12 Other analysis tools

There are other parallel performance tool development efforts which seem to have been abandoned over the years, such as Annai, Falcon, Faust, FORGE, SUIF, KAP/PRO, PAT, pedb, Prism, and VT, to name a few. Numerous others do not support any of our currently installed parallel systems, such as those for Windows and Macintosh SMP systems, but most notably for the Cray and vectorizing platforms that may be of concern for us in the future.

For modeling an applications performance and predicting its performance on other platforms, there is an infrastructure available called Prophesy. At its core is a relational database for recording analysis data, system features, and application details. Its performance data collector, PAIDE, automatically instruments a code according to user specifications and stores the info in the database. The entire database is accessible via the web and analytical performance models are generated from optimization techniques derived from numerous data collection runs. When comparing these models with system information, insight into performance predicted for alternative system can be gained.

Chapter 5: Communication & Networking

5.1 Communication Libraries

5.1.1 MPI (& MPI-2)

MPI (Message Passing Interface) is an international standard developed by The MPI Forum - a group of researchers, vendors and application developers from government, academia and industry. The standard specifies the exchange of messages among multiple processors. Particular features of the standard include specifications for point-to-point communication using send and receive calls, collective communication, the ability to define data types, and the ability to define virtual topologies. MPI 1.1 contains over 100 distinct calls, although only a few are typically used. Specific considerations in the development of MPI included the determination that MPI would be a library, not a distributed operating system. Other considerations were that it would not mandate thread-safe implementations although it would allow them, it would be capable of delivering high performance on high performance systems, it would be modular and extensible, it would support heterogeneous computing and it would have well defined behavior. MPI is portable in the sense that the specification is machine independent and supports heterogeneous computing in the sense that it provides the capability for translating between machine architectures that have different byte orderings and supports both SIMD and MIMD parallelism. MPI-1.2 clarifies and corrects the 1.0 and 1.1 standards.

MPI-2 extends the MPI specification to include dynamic process management, parallel I/O, remote memory operations, mixed language programming, and the capability for synchronized access to shared data via threads.

MPI has become the de-facto standard for massively parallel communication and MPI 1.1 and 1.2 implementations are available for a wide range of platforms, including IBM, Alpha, Cray, Sun, SGI architectures and Linux-based clusters. It is important to note that many of the implementations of the MPI standard may not have the flexibility implied in the standard (e.g., support for different byte ordering and both SIMD and MIMD processing).

There are numerous implementations of the MPI (and MPI-2) standards. Some of these include MPICH, MVAPICH, LAM/MPI, LA-MPI, FT-MPI, PACX-MPI, Open MPI, MPICH-V, WMPI II, MP_Lite, GridMPI, and MPI-XF, to name a few. In addition, Cray and SGI have versions of their own shared memory access library called SHMEM that further extends the capabilities of MPI. Comparing the nuances and performance characteristics of each requires more detailed analysis than this paper provides.

5.1.2 PVM

PVM is a software package that permits a heterogeneous collection of UNIX and/or Windows computers hooked together by a network to be used as a single large parallel computer. PVM provides a portable heterogeneous environment for using clusters of machines employing socket-based communications over TCP/IP. The design goals of PVM include support for a user-configured host pool, access to the hardware by the

application programmer, task-based parallelism where the granularity of the parallelism may be UNIX sub-process (i.e. a thread), explicit message passing model where message size is only limited by the available memory, support for heterogeneity and support for multiprocessors. PVM supports both functional and data parallelism as well as hybrid modes. PVM is portable in the sense that the specification is machine independent. PVM is capable of translating between machine architectures that have different byte orderings and supports both SIMD and MIMD parallelism. PVM currently has C, C++ and FORTRAN language bindings.

While MPI is the de facto standard for massively parallel computing, PVM is often found in more heterogeneous distributed computing environments. PVM provides better support for fault tolerance and recovery than MPI. Fault tolerance and recovery are current research areas in MPI.

5.1.3 OpenMP (& POSIX threads)

OpenMP is an interface specification for multi-threaded shared memory parallelism. It consists of compiler directives, runtime library routines, and environment variables. It is specified for FORTRAN, C and C++ and runs on most platforms including most Unix-based systems as well Windows. Similar to MPI, OpenMP was developed by a group consisting of hardware and software vendors plus researchers and developers from government and academia. Also in similar vein to the goals of MPI and PVM, the goal was to provide a standard among the many shared memory platforms. OpenMP is thread-based with shared memory processes consisting of multiple threads using a fork-join model for parallelism. Parallelism is specified by compiler directives and the specification supports dynamic thread creation and destruction.

Porting thread-based parallel applications to non-shared memory architectures such as clusters has required implementing an additional message passing interface of some sort such as MPI. Such hybrid codes can then be widely ported and able to take the best advantage of each platform's optimal communication methodology. However, OpenMP programming is typically easier to implement into a code (with the caveat of thread-safety issues, should there be any). A new tool promising to assist a code's migration onto cluster systems is called ClusterThreads. It provides software-based virtual shared memory on clusters and an identical (or at least similar) simple interface as OpenMP's. This emerging technology enables execution of threads-based parallel applications on clusters with minimal decrease in performance. Engineered Intelligence is developing it.

There are a few utilities available that automate or help to translate serial codes into parallel ones. One such utility is called ParaWise for FORTRAN codes which is different from others in this field in that it can generate MPI code, OpenMP code, or hybrid code.

5.2 Distributed Computing

5.2.1 Globus

The Globus Toolkit is an open source collection of software services and libraries for resource management, monitoring, discovery, security and file management to support

Grid Computing. Its latest version, GT3, marks a significant change from previous versions in that it employs a full-scale implementation of new Open Grid Services Architecture (OGSA). OSGA specifications were developed in participation with the Globus Alliance that distributes the Globus toolkit. The previous generation toolkit, GT2, did not use Web service standards and you can still install non-Web services versions of the Security, GridFTP, Resource Management (GRAM), Replica Location Service, and Information Services (MDS2). One of the challenges for larger acceptance of Globus has been the volatility of its architecture from release to release. The recent focus on OSGA and release of GT3 hope to alleviate some of these concerns.

The GT3 has Java and C language APIs. Platforms supported include UNIX/Linux and Windows, however only the Java API is available for Windows development. The non-Web service standard features are only available for UNIX platforms. GT requires the following support software: Java SDK, ant, Junit, C compiler, YACC or Bison, and GNU tar. The following support software is optional: Jakarta Tomcat, .NET, JDBC-compliant database.

5.2.2 Condor

Condor is a specialized workload management system supporting High Throughput Computing on collections of distributed computing resources. Condor provides mechanisms for job queuing, policy scheduling and priority schemes, resource monitoring and management. The scope of systems on which Condor can be used ranges from clusters of dedicated compute nodes to idle CPUs on desktop workstations. Condor does not rely on a shared file system but can redirect I/O so that an organization's computational resources can be combined into a single resource.

Condor can use Grid environments that cross administrative boundaries and Condor-G is interoperable with the Globus toolkit.

Condor 6.7.0 supported platforms

Architecture	Operating System
Hewlett Packard PA-RISC (both PA7000 & PA8000 series)	HPUX 10.20
Sun SPARC Sun4m,Sun4c, Sun UltraSPARC	Solaris 2.6, 2.7, 8, 9
Silicon Graphics MIPS (R5000, R8000, R10000)	IRIX 6.5
Intel x86	Red Hat Linux 7.1, 7.2, 7.3, 8.0, 9.0 Windows 2000 Professional and Server, 2003 Server Windows XP Professional
ALPHA	Digital UNIX 4.0 Red Hat Linux 7.1, 7.2, 7.3 Tru64 5.1
PowerPC	Macintosh OS X AIX 5.2L
Itanium	Red Hat Linux 7.1, 7.2, 7.3 SuSE Linux Enterprise 8.1

Table taken from the Condor Web site, http://www.cs.wisc.edu/condor/manual/v6.7.1/8_2Development_Release.html .

Chapter 6: Visualization Tools

In this chapter, we will summarize tools we identified that address the visualization of data from HPC applications. The focus is on the major parallel visualization tools: VisIt (including MeshTV), EnSight, and ParaView. However, there are other parallel utilities and numerous serial tools that are also used for presenting data from HPC applications. There are other categories of tools that present such data in other ways, such as by modeling, rendering, and animating, which are not discussed here except for a few popular selections among our local community. Although we do mention a few other tools for post-processing, including analysis of data and modeling, used by the projects we surveyed, our focus is on parallel visualization tools, of which there are unfortunately few.

6.1 VisIt

VisIt is an interactive parallel visualization and data analysis tool for viewing scientific data on UNIX, Windows, and Macintosh platforms. It is an open source tool developed here at LLNL and is the successor to MeshTV. Users can quickly generate visualizations from their data, animate them through time, manipulate them, and save the resulting images for presentations. VisIt contains a rich set of visualization features to view data in a variety of ways. It can be used to visualize scalar, vector, and tensor fields and it has a variety of both surface and volumetric rendering methods. It supports two- and three-dimensional structured, unstructured, and AMR meshes, and it supports hierarchical data organization by domains, blocks, parts, and even value-based decompositions such as material regions that require interface reconstruction.

It has many data analysis features as well. For example, it contains a powerful expression language that includes standard mathematical operations (plus, times, gradient, and so on). It allows for data to be queried in many ways, including picking on a single zone, creating "lineouts" where variable value is plotted against distance along a user-specified line, the ability to determine the surface area, and many more. In addition, these capabilities can be intuitively combined. For example, a user can use VisIt's material interface reconstruction to look at a certain material, restrict the region of interest using one of VisIt's data manipulation operators, and then determine the mass inside that box using the query operator.

VisIt was designed to handle very large data set sizes in the terascale range. It was recently used to visualize results of a 12 billion cell calculation on 1600 processors. It uses MPI for parallel communication and scales easily to 512 processors. Since most visualization operations are easily parallelizable, scaling should remain good beyond that.

VisIt has a number of features that allow it to handle these large datasets. For example, it has a componentized architecture that allows a parallel job to run on the largest of supercomputers and deliver the processed, reduced geometry to a viewer on the user's desktop machine where it can leverage the power of modern graphics cards. Conversely, if even the reduced geometry is too large for a desktop workstation to handle, VisIt will automatically switch into a parallel rendering mode, where it makes use of the power of the supercomputer to render the images in software and delivers only images to the

desktop. An advantage of VisIt is that it does not require a domain decomposition tool to separate the data prior to a parallel calculation.

As another example, VisIt makes use of metadata (like interval trees) to avoid processing data that will not be visible in the final picture. It works on data in the format written by the simulation; this means not only can users read data where it was written and without conversion, it can use the decomposition applied to the simulation for parallelism.

There are many advantages to VisIt. It was designed from the start to be a parallel visualization tool, so it scales well on large numbers of processors. It also runs in a client server mode so that the compute intensive tasks (such as rendering) can be run on the mainframe and the display functions can be executed on the local workstation. VisIt supports live connection to simulations, and supports a Python scripting interface, and VisIt is extensible through XML and plug-ins.

We did identify some areas of improvement for VisIt, such as adding support for higher order elements, and providing better support for AMR meshes.

6.2 EnSight

EnSight, from Computational Engineering International (CEI), is a general-purpose visualization toolset used for post-processing and analysis of scientific and engineering data sets. EnSight can be used for analyzing, visualizing and communicating high-end scientific and engineering datasets that can take full advantage of the parallel-processing machines and can handle models containing hundreds of millions of nodes, and in the near future, billions of nodes. It provides readers and translators for all common engineering analysis codes, as well as interfaces to common aerospace formats. Through EnSight's reader library, you can add your own custom data readers.

EnSight runs in parallel, sharing the workload between a server process (handling data I/O and all compute intensive functions) and a client process (managing user-interface interaction and graphic rendering). EnSight has limited use at LLNL, but is used heavily by other DOE sites. EnSight is the most mature of the parallel visualization tools, and is highly productized. It also has the richest feature-set of all the parallel visualization tools for scientific post-processing and presentation graphics. However, it does not have a parallel rendering facility (a feature enjoyed by some at LLNL) so it requires the use of local graphics hardware. It also is not an open source code so it can only extend though file readers without going through CEI.

6.3 ParaView

ParaView is visualization tool for displaying complex data sets from a variety of sources including VTK, Plot3D, EnSight, STL, and Wavefront. ParaView was developed by Kitware as part of a three-year contract awarded by the National Laboratories. ParaView is a parallel visualization tool built upon the VTK Library, providing transparent multi-processing support. Data streaming is employed to process large datasets. The GUI uses the Tk widget set.

ParaView was designed as a tool to assist researchers in developing new algorithms and techniques in an infrastructure that allows their new methods to interact with existing methods. ParaView is not a mature product, but it is OpenSource and is extendable through the use of plug-ins with a rich feature set. Its parallel model does require a pre-processing domain decomposition step, which is a disadvantage.

Features of ParaView include: support for scripting using Tcl, support for data streaming, support for data parallelism, support for parallel software rendering with MPI and shared memory architectures, and the capability to add user defined filters.

6.4 Tecplot

Although not a parallel visualization tool, Tecplot is an analysis tool that lets you create, manipulate, animate, and display complex data sets. It has extensive 2D and 3D capabilities for visualizing technical data from analyses, simulations and experiments. Tecplot is basically a general engineering plotting tool with 3-D scientific data visualization capabilities. Tecplot is used by various groups across LLNL. Tecplot runs on a variety of platforms including MS-Windows, Mac OS X, Linux PCs, UNIX workstations, HP, IBM, SUN, and SGI. Tecplot uses Open/GL for graphics so an Open/GL accelerated graphics card is recommended. It does not run in parallel and we would not consider it a good candidate for processing data for very large simulations.

6.5 GRIZ

GRIZ is an interactive serial application that was developed in LLNL's Engineering Directorate for visualizing finite element analysis results on three-dimensional unstructured grids. GRIZ calculates and displays derived variables for a variety of codes. Currently, GRIZ works with the family of Methods Development Group (MDG) analysis codes, including DYNA3D, NIKE3D and TOPAZ3D. GRIZ reads in data files in the "MDG plotfile" format.

In addition to a basic state variable display, GRIZ provides modern 3D visualization techniques such as isocontours and isosurfaces, cutting planes, vector field display, and particle traces. GRIZ provides flexible control of mesh materials on an individual basis, allowing the user to concentrate analysis and visual focus on important subsets of the mesh. GRIZ incorporates the ability to animate all representations over time.

The most significant limitation of GRIZ is the fact that it is a serial tool and suffers performance wise with large data sets. Its advantages are that it is very easy to use, lightweight, highly portable, and tailored to engineering analysis problems that are modeled using finite-element codes such as Dyna-3D.

6.6 Maya

Maya is a modeling and rendering suite of powerful 3D graphics programs for generating complex animations. Capabilities provided by Maya include modeling and texturing, animation, character animation, rendering, and paint effects. Although the package is mainly geared towards production houses, it has found limited use at LLNL for

developing complex animations; in fact, many of LLNL's scientific animations are produced using Maya. Maya runs on a variety of platforms from small desktop workstations to large mainframes including SGI, Compaq, PCs, and Macs. Rendering can be done in parallel on platforms such as SGI and IBM.

6.7 Chromium

Chromium is a system for interactive management of streams of OpenGL graphics commands on clusters of workstations. It supports rendering techniques such as sort-first and sort-last. The framework is general so cluster node operations on streams may be customized by the user. The rendering resources in a cluster are virtualized in Chromium so existing serial and parallel OpenGL applications can be ported to clusters very easily. It does not support architectures that require communication between stages in the visualization pipeline (e.g. between geometry and rasterization stages). Chromium is extensible and can employ a variety of underlying algorithms. Chromium runs on Microsoft Windows and Unix varieties (including Linux and IRIX).

Chapter 7: Other Parallel Tools

7.1 UPC

Unified Parallel C (UPC) developed at LBL is an extension to the C programming language that provides a single programming model to support both shared memory and distributed memory. It uses a single program multiple data (SPMD) computing paradigm in which each processor sees a uniform, shared, partitioned, address space which can be read from or written to by any processor, but which associates a variable with a single processor. It extends ISO C 99 by including "an explicit parallel execution model, a shared address space, synchronization primitives and a memory consistency model", and "memory management primitives." A table of its currently supported platforms can be found at the UPC Web site http://upc.lbl.gov/download/index.shtml.

There also is an explicitly parallel dialect of Java being developed at UC Berkeley called Titanium. It too uses an SPMD control model and is similar to UPC. For these types of parallel global address space SPMD languages, UC Berkeley has a high-performance communication primitive tailored to them called GASNet, however it is intended for use by runtime library writers, not end users. There is a performance analysis tool for it called GASNet Trace. TotalView will soon be able to debug executables generated by UPC too.

7.2 Parallel Matlab

Matlab is a high-level technical computing language and interactive environment for algorithm development, data visualization, data analysis, and numerical computation. Its scripting capability, rapid prototyping and ease of use make it popular for small-scale scientific studies. Matlab supports calling other applications in C/C++, Java, and FORTRAN. It supports access to Web services and data access for ODBC/JDBC compliant databases. Matlab can read scientific databases such as HDF and HDF5 as well as binary I/O. It imports and exports a variety of image, sound and video formats and supports reading and writing of XML.

There are currently over twenty projects to develop a parallel version of Matlab. The projects vary in their approach to parallelization from embedding communications libraries (e.g. MPI, PVM), to communication between concurrent Matlab sessions to parallel applications that use Matlab as a front-end to Matlab compilers that compile directly into parallel code.

MatlabMPI implements a subset of the MPI specification as a set of Matlab scripts enabling any Matlab program to run on a parallel computer. For large messages, MatlabMPI shows comparable performance to native C / MPI.

pMatlab from MIT Lincoln Labs is an effort to enable parallelization of Matlab scripts with minimal impact on the Matlab user. With just a few small changes, a serial Matlab script can be run in parallel. pMatlab runs on top of MatlabMPI. pMatlab has twice the latency of MatlabMPI, but does not incur any other significant overhead beyond MatlabMPI. Both MatlabMPI and pMatlab run on multiple platforms including Linux, Windows, and Mac OS X.

The Cornell Multitasking Toolbox for Matlab is a parallelization effort that enables multiple running copies on a network of workstations to exchange matrices. It is distributed for both UNIX and Windows platforms.

The PVM Toolbox for Matlab (PVMTB) and the MPI Toolbox for Matlab (MPITB) enable parallelization using PVM and MPI respectively. PVMTB is supported on Linux platforms and requires at least RedHat 6.1, Matlab 5.2 - 5.3 and PVM 3.4.2. MPITB requires LAM 7.0 and Matlab 6.5 (installed on each node).

Parmatlab, distributed by MathWorks, enables the distribution of processes of multiple Matlab instances distributed across the Internet. It operates in both multiple program multiple data (MPMD) or single program multiple data (SPMD) mode. It does not require a common files system. All communications are TCP/IP based.

Conlab (CONcurrent LABoratory) is an extension to Matlab that natively includes control structures for explicit message passing, shared memory and synchronization. Conlab is distributed with its own compiler CLC (CONLAB compiler) which is built on the BLAS and LAPACK libraries.

7.3 AMPI

Adaptive Message Passing Interface, (AMPI), developed at UIUC, is an implementation of MPI that supports dynamic load balancing and multithreading for MPI applications. AMPI associates user-level threads with message passing objects. These user level threads exist as virtual processors that are mapped onto real processors. AMPI manages the adaptation of overlapping computation and communication. It is not currently fully compliant with MPI-1.1 and does not support MPI-2

AMPI is distributed as part of the Charm++/Converse distribution. AMPI runs on multiple supercomputing platforms, networks of workstations or on single-processor UNIX machines.

7.4 JXTA

JXTA is a collection of open protocols developed by Sun Microsystems to allow any devices connected on a network to communicate as peers. JXTA creates a virtual network overlay on top of the Internet that hides the underlying physical topology. Network devices ranging from cell phones to PDAs to workstations to high-end servers can use this P2P technology. JXTA enables peers to collaborate across firewalls, NATs and over different network transports. The core JXTA protocols allow peer discovery, peer group formation, peer communication, peer monitoring and security. Future directions for JXTA include improving scalability, increasing security and monitoring, and stronger integration with other Web services such as SOAP, UDDI, and WSDL.

7.5 SOAP

SOAP is an official recommendation from the Internet Engineering Task Force (IETF) to provide a packaging protocol for message sharing between applications. It uses HTTP for transport and delivers XML messages. SOAP messages are string-based and provide a capability for two applications to openly share information in a heterogeneous environment independent of operating system, programming language, and other technical implementation-specific details. It operates using a request/response model but does not require a strong connection between client and server like that found in proprietary protocols such as DCOM and RMI. SOAP fills the gap in Internet environments where one cannot guarantee system characteristics between client and server – only the fact that they are both communicating HTTP. It thus enables interoperability between heterogeneous and potentially incompatible systems.

One of the main limitations of SOAP is limited functionality and performance. Since it is string based, it relies on TCP protocols such as HTPP and SMTP and thus cannot offer as rich a functionality as other distributed message sharing protocols such as DCOM and RMI. Adding to the overhead of the TCP transmission is the parsing of XML, which further decreases performance.

7.6 EJB

Enterprise Java Bean technology, developed by Sun Microsystems, describes the server-side component architecture for the Java 2 Enterprise Edition (J2EE) platform. EJB defines a standard for distributed and transactional applications using Java.

Application level communications technologies continue to standardize – particularly around Web services. The two primary technological challenges are achieving high performance over Internet or Internet-like technologies, and providing adequate security mechanisms, especially in anonymous environments such as p2p computing. The greater challenge is to the tools community where there will be a need for debugging and interacting with computations that may span distributed communications environments.

7.7 Eclipse System

Eclipse is a tool for building integrated development environments for a wide variety of applications that employ technologies as diverse as Web development, Enterprise Java Beans, and C++. It is an open source project managed by eclipse.org, which was founded by several industrial partners including IBM, Borland, Rational, RedHat and others. It runs on Windows and Linux platforms and supports GUI and non-GUI application development and a wide range of content types (C, Java, html, GIF, etc.). The parallel functionality for this system is currently being developed at LANL. Existing parallel tools offer developers a confusing proliferation of GUI and command-line interfaces. Few tools have integrated cross-category features, such as debugging alongside performance analysis, and so users encounter different interfaces with each tool. This switching annoyingly between tools for different functionality is disruptive to the workflow and reduces productivity. The ambitious Eclipse Parallel Tools project aims to combine a key set of parallel development tools into a single, uniform, and highly integrated development environment that can overcome many of the problems inherent

with the varying existing tools. This new highly customizable user interface will be easy to tailor to an individual user's needs, and will employ a technique known as perspectives so that the user can completely reconfigure the layout of the tools with a click of a button. Perspectives enable a user to view system activity and process status while concurrently viewing output, debugging, and even performance analyzing one or more processes. Eclipse also gives support for multiple-run configurations, thus enabling a user to predefine resources required for executing a program. Different runs can then be selected with a single mouse click.

Chapter 8: Postscript

8.1 Omissions & Expectations

Although we have discussed a great many public and commercial development tools in this paper, it is not by any means an exhaustive list. There are numerous other memory tools, for example, which we didn't mention for various reasons, including some developed within the Tri-Lab community, and likely others which we simply don't know about or are in initial development. Hopefully though, we have identified all the major players from which we can pick and choose or even have improved and adapted so we have adequate tools on all our future platforms for efficient development of all our codes. The collection presented here was gathered without bias or consideration of costs, contracts, licenses, or other issues that might restrict deployment of these tools on our current or future platforms.

Attempts at developing automated parallel performance tuning tools have been made, including software projects such as S-Check, Ursa Minor (including Merlin), and Interpol, but have met with limited success. Investigating these and other such tools under development could be quite beneficial to our parallel software development process.

There are several key technologies that will impact the future of visualization tools including streaming architectures, programmable graphics processing units (GPUs), lightweight kernels, and immersive environments. We also expect visualization tools to improve in their capability to handle more complex geometries including high-order elements and AMR meshes.

Some categories of high performance computing aides were not reviewed for this paper but certainly have significant impact on parallel performance or program development, such as I/O technologies, algorithmic libraries (of math solvers, sorting routines, etc), and scripting and programming languages, to name a few. Although the initial intention was to encompass all influencing technologies, the subset we did discuss proved to be quite challenging in itself to present here given our limited time and resources. Hardware, compiler optimizations, and operating system influences on program development were considered beyond the intended scope of this paper. Reviewing all of these overlooked topics, and keeping up with developments in the presented tools and technologies, along with including emerging advances in these areas, is likely enough work for many future papers.

8.2 Thanks

Special thanks to Blaise Barney and Shawn Dawson for their significant contributions to this paper. Thanks also goes out to those who provided input for this paper and/or the tools table, especially Chris Chambreau, John Gyllenhaal, Kathleen McCandless, and the great many tools and projects representatives who answered our many questions about their software.

We would also like to thank Jeremy Meredith for providing input for the Visualization sections and John Tannahill and Dan Bergman for providing input for the section on Earth Sciences computing. Additional thanks goes to all those who helped edit and review this lengthy report, including Burl Hall, Jeff Keasler, Bill Loewe, Scott Futral, and some others already mentioned.

8.3 References

B. Barney, *Performance Analysis Tools*, (LLNL UCRL-MI-133316, Dec 2002), See URL http://www.llnl.gov/computing/tutorials/performance_tools/

S. Futral, *Computing Tools*, (LLNL UCRL-MI-126792), See URL http://www.llnl.gov/icc/lc/DEG/tols_table.html

S. Futral, *Development Environment Group Tools Support*, (LLNL DCOM Colloquium, Feb 2004)

I. Corey, J. Johnson, *Optimization Techniques for RICS Cache-based Architectures*, (LLNL, Jun 2003)

J. Brown, A. Geist, C. Pancake, D. Rover, *Software Tools for Developing Parallel Applications Part 1: Code Development and Debugging*, (LANL ORNL/CP—97321, Mar 1997)

J. Brown, A. Geist, C. Pancake, D. Rover, *Software Tools for Developing Parallel Applications Part 2: Interactive Control and Performance Tuning*, (LANL ORNL/CP—97244, Mar 1997)

L. DeRose, *IBM End User Tools*, (IBM Research Center, Mar 2003)

J. Dongarra, K. London, S. Moore, P. Mucci, D. Terpstra, H. You, M. Zhou, *Experiences and Lessons Learned with a Portable Interface to Hardware Performance Counters*, (U. of Tennessee)

J. Dongarra, S. Moore, P. Mucci, *Performance Instrumentation and Measurement for TeraScale Systems*, (U. of Tennessee, Jun 2003)

K. London, J. Dongarra, S. Moore, P. Mucci, K. Seymour, T. Spencer, *End-user Tools for Application Performance Analysis Using Hardware Counters*, (U. of Tennessee)

S. Browne, J. Dongarra, K. London, *Review of Performance Analysis Tools for MPI Parallel Programs*, (U. of Tennessee, Dec 1997)

S. Moore, D. Cronk, K. London, J. Dongarra, *Review of Performance Analysis Tools for MPI Parallel Programs*, (U. of Tennessee)

J. Dongarra, A. Malony, S. Moore, P. Mucci, S. Shende, *Performance Instrumentation and Measurement for Tersscale Systems*, (U. of Tennessee)

C. Pancake, Performance *Tools for Today's HPC: Are We Addressing the Right Issues?*, (Northwest Alliance for Computational Science and Engineering)

I. Park, M. Voss, S. Kim, R. Eigenmann, *Parallel Programming Environment for OpenMP*, (Purdue University, Nov 2002)

S. Kim, I. Park, R. Eigenmann, *A Performance Advisor Tool for Novice Programmers in Parallel Computing*, (Purdue University)

R. Moore, *Survey of Parallel Performance Tools and Debuggers*, (Macquarie University, Mar 1999)

M. Heath, J. Finger, ParaGraph*: A Performance Visualization Tool for MPI*, (U. of Illinois, Aug 2003)

P. Mucci, *Dynaprof and PAPI*, (LBNL, Aug 2002)

J. May, J. Gyllenhaal, *Tool Gear: Infrastructure for Parallel Tools*, (LLNL UCRL-JC-152873, Apr 2003)

K. McCandless, *A/X Computer Science Software Development: Leveraging LC Supported Tools to get it done*, (LLNL DCOM Colloquium, Apr 2004)

J. Meredith, *Tools: B-Division Computer Science*, (LLNL DCOM Colloquium, Apr 2004)

B. Anderson, *Linux Debuggers*, (LLNL) See URL http://www-r.llnl.gov/CASC/computing/software/linux_debuggers.html

B. Zorn, *Debugging Tools for Dynamic Storage Allocation and Memory Management*, (U. of Colorado, Feb 2001) See URL http://www.cs.colorado.edu/homes/zorn/public_html/MallocDebug.html

G. Milmeister, *Linux Development Tools*, (HotFeet, Feb 2001) See URL http://www.hotfeet.ch/~gemi/LDT

TotalView Users Guide Version 6.5, (Etnus, Jun 2004)

Alexey Loginov, Suan Hsi Yong, Susan Horwitz and Thomas Reps, *Debugging via Run-Time Type Checking* (Proceedings of Proceedings of Fundamental Approaches to Software Engineering.April 2001)

Bill Allcock, Ian Foster, et. Al.. *High-Performance Remote Access to Climate Simulation Data: A Challenge Problem for Data* grid Technologies, (SC2001 November 2001 Conference Proceedings)

http://www-unix.mcs.anl.gov/mpi/index.html

William Gropp, Ewing Lusk, *Goals Guiding Design: PVM and MPI*. (available from http://www-unix.mcs.anl.gov/mpi/index.html) http://www.csm.ornl.gov/pvm/pvm_home.html

http://www.netlib.org/pvm3/book/node17.html

http://www.llnl.gov/computing/tutorials/workshops/workshop/openMP/MAIN.html#Introduction

http://www.mathworks.com/products/matlab/

http://www-pcmdi.llnl.gov/

8.3.1 Product Information Homepage Links

Debuggers

DDT:	http://www.allinea.com/
décor:	http://www.llnl.gov/icc/lc/DEG/
gdb & DDD:	http://www.gnu.org/software/ddd/
Guard:	http://www.guardsoft.net/classicguard.html
IDB:	http://www.intel.com/software/products/compilers/docs/linux/idb_manual_l.html
Ladebug:	http://h30097.www3.hp.com/dtk/ladebug_ov.html
pdbx:	http://publib.boulder.ibm.com/clresctr/windows/public/pebooks.html
pgdbg:	http://www.pgroup.com/doc/pgitools.pdf
TotalView:	http://www.etnus.com/TotalView/index.html

Memory Checkers

ccmalloc:	http://www.inf.ethz.ch/personal/biere/projects/ccmalloc/
dmalloc:	http://dmalloc.com/

Electric Fence: http://www.perens.com/FreeSoftware/
Application Saver: http://eval.veritas.com/mktginfo/products/Datasheets/Application_Performance/veritas_application_saver_datasheet.pdf
Insure++: http://www.parasoft.com/jsp/products/home.jsp?product=Insure
MemCheck Deluxe: http://prj.softpixel.com/mcd/
MemUsage (ours): http://www.llnl.gov/bdiv/bdiv_openhome.html
Mpatrol: http://www.cbmamiga.demon.co.uk/mpatrol/
Mprof: http://www.utdallas.edu/~cantrell/ee6345/4_4BSD-Lite/usr/src/contrib/mprof/readme
Purify: http://www.pts.com/purify.cfm
SmartHeap: http://www.microquill.com/smartheapsmp/index.html
Third Degree: http://h30097.www3.hp.com/developerstoolkit/tools.html
Valgrind: http://valgrind.kde.org/
ZeroFault: http://www.zerofault.com/zf/

Profiling APIs

DCPI: http://h30097.www3.hp.com/dcpi/
DPCL: http://oss.software.ibm.com/developerworks/opensource/dpcl/
Dyninst: http://www.dyninst.org/
PAPI: http://icl.cs.utk.edu/papi/
PMAPI: http://www.alphaworks.ibm.com/tech/pmapi

Profiling Toolkits

AIMS: http://www.nas.nasa.gov/Groups/Tools/Projects/AIMS/
HPCToolkit: http://www.hipersoft.rice.edu/hpctoolkit/
HPM Toolkit: http://www.alphaworks.ibm.com/tech/hpmtoolkit
KOJAK: http://www.fz-juelich.de/zam/kojak/
PE Benchmarker: http://publib.boulder.ibm.com/clresctr/windows/public/pebooks.html
PerfSuite: http://perfsuite.ncsa.uiuc.edu/
SpeedShop: http://techpubs.sgi.com/library/tpl/cgi-bin/browse.cgi?coll=0650&db=bks&cmd=toc&pth=/SGI_Developer/SShop_UG
TAU: http://www.cs.uoregon.edu/research/paracomp/tau/tautools/

Profilers

DEEP/MPI: http://www.crescentbaysoftware.com/deep_mpi_top.html
Dynaprof: http://www.cs.utk.edu/~mucci/dynaprof/
gprof: http://www.gnu.org/software/binutils/manual/gprof-2.9.1/html_mono/gprof.html
Hiprof: http://www.research.compaq.com/wrl/projects/om/hiprof.html
IPM: http://www.nersc.gov/nusers/resources/software/ibm/hpmcount/poe+.php
MPIP (ToolGear): http://www.llnl.gov/CASC/mpip/
MPX (ToolGear): http://www.llnl.gov/CASC/mpx/
PapiEx: http://icl.cs.utk.edu/~mucci/papiex/
Perfex: http://techpubs.sgi.com/library/tpl/cgi-bin/getdoc.cgi?coll=&db=man&fname=/usr/share/catman/u_man/cat1/perfex.z
pgprof: http://www.pgroup.com/doc/pgitools.pdf

Pixie:	http://techpubs.sgi.com/library/tpl/cgi-bin/getdoc.cgi?coll=0650&db=bks&fname=/SGI_Developer/books/Cplr_PTG/sgi_html/ch04.html&srch=procedure+register
prof:	http://techpubs.sgi.com/library/tpl/cgi-bin/getdoc.cgi?coll=&db=man&fname=/usr/share/catman/u_man/cat1/prof.z
SCALEA:	http://www.par.univie.ac.at/project/scalea/
Tprof:	http://www16.boulder.ibm.com/pseries/en_US/aixbman/prftungd/cpuperf4.htm
Vprof:	http://hpcn.ca.sandia.gov/~cljanss/perf/vprof/
Vtune:	http://www.intel.com/software/products/vtune/vlin/index.htm
Xprofiler:	http://publib.boulder.ibm.com/clresctr/windows/public/pebooks.html

Tracers

mpitrace:	http://oss.software.ibm.com/developerworks/opensource/dpcl/
Performeter:	http://icl.cs.utk.edu/papi/
RootCause/Aprobe:	http://www.ocsystems.com/prod_rootcause.html
Sigma:	http://www.alphaworks.ibm.com/tech/sigma

Visualizers (GUI)

Intel Trace Tools:	http://www.intel.com/software/products/cluster/tanalyzer/index.htm
Jumpshot / MPE:	http://www-unix.mcs.anl.gov/perfvis/software/viewers/
Opt:	http://www.allinea.com/
OptiPath:	http://www.pathscale.com/optipath.html
Pablo/SvPablo:	http://www-pablo.cs.uiuc.edu/
Paradyn:	http://www.paradyn.org
ParaGraph/MPICL:	http://www.csar.uiuc.edu/software/paragraph/index.html
Paraver/Dimemas:	http://www.cepba.upc.es/paraver/
SeeWithin/Pro:	http://www.mpi-softtech.com/products/cluster/seewithinpro/
Vampir/GuideView:	http://www.pallas.com/e/products/vampir/
xmpi:	http://www.lam-mpi.org/software/xmpi/

Communication Libraries

MPI (MPI-2):	http://www.mpi-forum.org/docs/docs.html
PVM:	http://www.csm.ornl.gov/pvm/pvm_home.html
OpenMP:	http://www.openmp.org

Networking Software

Globus:	http://www.globus.org/toolkit/
Condor:	http://www.cs.wisc.edu/condor/

Data Visualization Tools

GRIZ:	http://www.llnl.gov/eng/mdg/Codes/Griz/body_griz.html
ParaView:	http://www.paraview.org
VisIt:	http://www.llnl.gov/visit/
Tecplot:	http://www.tecplot.com/
EnSight:	http://www.ceintl.com/
Maya:	http://www.aliaswavefront.com/eng/products-services/maya/index.shtml
Chromium:	http://chromium.sourceforge.net/

Other Parallel Tools

UPC:	http://upc.nersc.gov/
Parallel Matlab:	http://supertech.lcs.mit.edu/~cly/survey.html
AMPI:	http://charm.cs.uiuc.edu/research/ampi/
JXTA:	http://jxta-grid.jxta.org/
SOAP:	http://www.w3.org/2000/xp/Group/
EJB:	http://java.sun.com/products/ejb/
Eclipse:	http://www.eclipse.org/

Appendix: Tools Table

By Michael Collette

Legend: +:Yes, -:No, ?:May Note: a * indicates that this value is expected to change soon Dec 2004 Revision 1.1

Category	Platforms											Languages						Parallelism				
Tool	IBM-AIX 32-bit	IBM-AIX 64-bit	x86-Intel-Linux	IA-64-Intel-Linux	Alpha-Tru64	SGI-Irix	Sun-Sparc	AMD-Opteron	Win NT	Win 2k	Win XP	C	C++	F77	F90	C/C++ Mixed	C/F77 Mixed	MPI	MPI-2	MPICH	PVM	SH MEM
Debuggers																						
DDT	+	+	+	+	-	+	+	+	-	-	-	+	+	+	+	+	+	+	+	+	+	+
décor	+	+	?	?	-	-	-	-	-	-	-	+	+	+	+	+	+	+	+	+	-	+
gdb & DDD	+	+	+	+	+	+	+	+	-	-	-	+	+	+	+	+	+	-	-	-	-	+
Guard	+	-	+	-	-	-	+	-	-	-	-	+	+	+	+	+	+	+	-	+	+	+
IDB	-	-	+	+	-	-	-	-	-	-	-	+	+	+	+	+	+	+	+	+	-	+
Ladebug	-	-	-	-	+	-	-	-	-	-	-	+	+	+	+	+	+	+	+	+	-	+
pdbx	+	+	-	-	-	-	-	-	-	-	-	+	+	+	+	+	+	+	+	+	-	+
pgdbg	-	-	+	-	-	-	-	+	-	-	-	+	+	+	+	+	+	+	-	+	-	-
TotalView	+	+	+	+	+	+	+	+	-	-	-	+	+	+	+	+	+	+	+	+	+	+
Memory Checkers																						
ccmalloc	+	?	+	+	+	+	+	+	-	-	-	+	+	-	-	+	-	?	?	?	?	?
dmalloc	+	?	+	+	+	+	+	+	-	-	-	+	?	-	-	?	-	?	?	?	?	+
Electric Fence	-	-	+	+	+	-	+	-	+	+	+	+	+	-	-	+	-	+	+	+	+	+
Application Saver	+	+	+	-	-	-	+	-	+	+	+	+	+	-	-	+	-	+	-	-	-	+
Insure++	+	+	+	+	+	+	+	+	+	+	+	+	+	-	-	+	-	+	+	+	+	+
MemCheck Deluxe	-	-	+	+	-	-	-	-	-	+	-	+	+	-	-	+	-	?	?	?	?	?
MemUsage (ours)	+	+	+	+	+	-	+	-	-	-	-	+	+	+	+	+	+	+	+	+	-	+
Mpatrol	+	-	+	-	+	+	+	-	+	-	-	+	+	-	-	+	-	?	?	?	?	+
Mprof	-	-	-	-	-	+	+	-	-	-	-	+	-	-	-	-	-	?	?	?	?	?
Purify	-*	-*	+	+	-	+	+	+	+	+	+	+	+	-	-	+	-	+	+	+	+	+
SmartHeap	+	+	+	+	+	+	+	-	+	+	+	+	+	-	-	+	-	-	-	-	-	-
Third Degree	-	-	-	-	+	-	-	-	-	-	-	+	+	+	+	+	+	+	+	+	+	+
Valgrind	-	-	+	-	-	-	-	-	-	-	-	+	+	+	+	+	+	-	-	-	-	-
ZeroFault	+	-	-	-	-	-	-	-	-	-	-	+	+	+	+	+	+	+	+	+	+	-
Profiling APIs																						
DCPI	-	-	-	-	+	-	-	-	-	-	-	+	+	+	+	+	+	+	+	+	-	+
DPCL	+	+	-	-	-	-	-	-	-	-	-	-	+	-	-	-	-	+	+	+	+	+
Dyninst	+	+	+	+	+	+	+	-	?	+	+	+	+	+	+	+	?	+	-	+	-	-
PAPI	+	+	+	+	+	+	+	+	-	-	-	+	+	+	+	+	+	+	+	+	+	+
PMAPI	+	+	-	-	-	-	-	-	-	-	-	+	+	+	+	+	+	+	+	+	+	+
Profiling Toolkits																						
AIMS	+	-	+	-	+	+	+	-	-	-	-	+	-	+	-	-	+	+	-	+	+	-
HPCToolkit	-	-	+	+	+	+	-	+	-	-	-	+	+	+	+	+	+	+	+	+	+	+
HPM Toolkit	+	+	-	-	-	-	-	-	-	-	-	+	+	+	+	+	+	+	+	+	-	+
KOJAK	+	-	+	-	-	+	+	-	-	-	-	+	+	+	+	+	+	+	-*	+	-	-
PE Benchmarker	+	+	-	-	-	-	-	-	-	-	-	+	+	+	+	+	+	+	+	-	-	-
PerfSuite	-	-	+	+	-	-	-	-	-	-	-	+	+	+	+	+	+	+	+	+	+	-
SpeedShop	-	-	-	-	-	+	-	-	-	-	-	+	+	+	+	+	+	+	+	+	-	+
TAU	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
Profilers																						
DEEP/MPI	+	-	+	-	-	+	+	-	+	-	+	+	-	+	+	+	-	+	+	+	+	-
Dynaprof	+	+	+	+	-	+	+	-	-	-	-	+	+	+	+	+	+	+	+	+	+	+
gprof	+	+	+	+	+	+	+	+	-	-	-	+	+	+	+	+	+	+	+	+	+	+
Hiprof	-	-	-	-	+	-	-	-	-	-	-	+	+	+	+	+	+	+	-	+	-	+
IPM	+	+	+	+	-	-	-	-	-	-	-	+	-	+	+	-	+	+	-	+	-	+
MPIP (ToolGear)	+	+	+	+	+	-	-	+	-	-	-	+	+	+	+	+	+	+	+	+	-	-
MPX (ToolGear)	+	-	-	-	-	-	-	-	-	-	-	+	+	+	+	+	+	+	-	-	-	-
PapiEx	-	-	+	+	+	+	+	+	-	-	-	+	+	+	+	+	+	+	+	+	+	+
Perfex	-	-	-	-	-	+	-	-	-	-	-	+	-	+	+	-	+	+	+	+	+	+
pgprof	-	-	+	-	-	-	-	+	-	-	-	+	+	+	+	+	+	+	-	+	-	-
Pixie	-	-	-	-	+	-	-	-	-	-	-	+	+	+	+	+	+	+	+	+	-	+
prof	+	+	+	+	+	+	+	+	-	-	-	+	+	+	+	+	+	+	+	+	+	+
SCALEA	-	-	-	-	-	-	+	-	-	-	-	-	-	+	+	-	-	+	-	+	+	+
Tprof	+	+	-	-	-	-	-	-	-	-	-	+	+	+	+	+	+	-	-	-	-	-
Vprof	+	+	+	+	+	+	+	+	-	-	-	+	+	+	+	+	+	+	+	+	+	+
Vtune	-	-	+	+	-	-	-	-	+	+	+	+	+	+	+	+	+	-*	-*	-*	-	-
Xprofiler	+	+	-	-	-	-	-	-	-	-	-	+	+	+	+	+	+	+	+	+	+	+
Tracers																						
mpitrace	+	+	-	-	-	-	-	-	-	-	-	+	+	+	+	+	+	+	-	-	-	-
Performeter	+	+	+	+	+	+	+	+	-	-	-	+	+	+	+	+	+	+	+	+	+	+
RootCause/Aprobe	+	-	+	-	-	-	+	-	+	+	+	+	+	?	?	+	?	?	?	?	?	?
Sigma	+	-	-*	-	-	-	-	-	-	-	-	+	-	+	+	-	+	-	-	-	-	-*
Visualizers (GUI)																						
Intel Trace Tools	+	+	+	+	+	+	+	-	-	-	-	+	+	+	+	+	+	+	+	+	-	+
Jumpshot / MPE	+	+	+	+	+	+	+	+	+	+	+	+	-	+	+	-	+	+	+	+	-	-
Opt	+	+	+	+	-	-	-	+	-	-	-	+	+	+	+	+	+	+	+	+	-	+
OptiPath	-	-	+	+	-	-	-	+	-	-	-	+	+	+	+	+	+	+	+	+	-	+
Pablo/SvPablo	+	+	+	+	+	+	+	?	-	-	-	+	-*	+	+	-*	+	+	+	+	-	+
Paradyn	+	+	+	-	-	-	+	-	?	+	+	+	+	+	+	+	?	+	-	+	-	-
ParaGraph/MPICL	+	-	+	-	+	+	+	-	-	-	-	+	-	+	+	-	+	+	-	+	+	-
Paraver/Dimemas	+	+	+	+	+	+	-	-	-	-	-	+	+	+	+	+	+	+	+	+	+	+
SeeWithin/Pro	-	-	+	+	-	-	-	+	+	+	+	+	-	+	+	-	+	+	-*	+	-	+
Vampir/GuideView	+	+	+	+	+	-	-	-	-	-	-	+	+	+	+	+	+	+	+	+	-	+
xmpi	+	+	+	+	+	+	+	+	-	-	-	+	+	-	-	+	-	+	-	-	-	-

Appendix: Tools Table

By Michael Collette

Legend: +:Yes, -:No, ?:May Note: a * indicates that this value is expected to change soon Dec 2004Revision 1.1

Category	Parallelism (cont)			Comments									
Tool	Posix pthrds	OpenMP	DMP SMP Mix	Free	Vendor	Versn	Date	On-site	Special Features	How to apply	Issues	Misc.
Debuggers												
DDT	+	+	+	-	Allinea	1.8	Oct-04	+	3D data visualization	GUI or CL	Newcommer	Were Streamline Cm.
décor	+	+	+	+	LLNL created	1.0	Nov-00	+	Uses lite corefiles	CL	Lacks support	
gdb & DDD	+	+	-	+	GNU	3.3.9	Jun-04	+	For Python, Perl, Java	GUI or CL	Easilly overwhelmed	
Guard	+	+	+	-	Guardsoft	1.0	Jun-01	-	Relative debugger	CL	No GUI	Uses gdb
IDB	+	+	+	-	Intel	8.1-3	May-04	-	dbx or gdb modes	GUI or CL	No longer parallel	
Ladebug	+	+	+	-	HP	4.0-68	Jun-04	-	Kernel debugging	GUI or CL	Alpha only (retired)	
pdbx	+	+	+	-	IBM	4.1	Jul-04	+	Low overhead	CL	Had GUI xpdbx	Was pedb
pgdbg	+	+	+	-	Portland Group	5.2.2	Aug-04	+	MPI queue display	GUI or CL	Max 64-prc/16-thr	With PG compilers
TotalView	+	+	+	-	Etnus	6.5	Jun-04	+	Memory checking	GUI or CL	Costly	On-site help available
Memory Checkers												
ccmalloc	?	?	?	+	GNU - GPL	0.4	Feb-03	-	Works on striped exe	Relink	Not for FORTRAN	
dmalloc	+	+	?	+	Gray Watson	5.3	Jan-04	+	Threaded support	Relink	Needs static link	Good w/threads
Electric Fence	+	+	+	+	Bruce Perens	2.4.10	Jul-04	+	Uses virtual memory	Relink w/lib	Not well w/threads	Too limited
Application Saver	+	+	+	-	Veritas Software	1.0	May-04	+	Memory management	Modify source	Not for FORTRAN	Was Great Circle
Insure++	+	+	+	-	ParaSoft	7.0	Sep-04	+	Code coverage info	Recompl tool	May need alter code	
MemCheck Deluxe	?	?	?	+	SoftPixel	1.2.3	Jun-04	-	Max & min alloc info	Relink	Not for FORTRAN	
MemUsage (ours)	+	+	+	+	LLNL - Bdiv	1.0.4	Aug-04	+	Info available to code	Modify source	Messaging overhead	
Mpatrol	+	+	?	+	GNU - GPL	1.4.8	Jan-02	-	Can mimick failures	Include headr	Not for FORTRAN	
Mprof	?	?	?	+	Ben Zorn	3.0	May-02	-	Simple, low overhead	Relink w/lib	Only C code	
Purify	+	+	+	-	IBM Rational	6.14	Apr-04	+	Highly regarded	Dynamic	Sun version is best	
SmartHeap	+	-	-	-	MicroQuill	7.3	Jun-04	-	Works on opt codes	Relink w/lib	Max 72 procs	For SMP systems
Third Degree	+	+	+	-	HP (Compaq)	5.4	Oct-03	+	Readable text output	Rebuilds exec	Alpha only (retired)	
Valgrind	+	-	-	+	GNU (GPL)	2.2.0	Aug-04	+	Other 'skins' available	Dynamic	Serializes threads	
ZeroFault	+	+	+	-	Kernel Group	4.6	Jun-04	+	Apply to procs subset	Dynamic	Can miss errors	
Profiling APIs												
DCPI	+	+	+	-	HP	4.0.1	Jun-04	+	Low overhead	Dynamic	Alpha only (retired)	
DPCL	+	+	+	+	IBM	3.2.6	May-03	+	Infrastructure for tools	Dynamic	Only C++ code	
Dyninst	+	?	?	+	U of Wisconsin	4.1	Apr-04	-	Apply to running job	Dynamic	Scales slow	
PAPI	+	+	+	+	U of Tennessee	3.0	Oct-04	+	Two interface levels	Modify source	Platform variability	
PMAPI	+	+	+	-	IBM	1.2.2	Jul-01	+	Kernel extensions	Modify source	PAPI more advanced	
Profiling Toolkits												
AIMS	-	-	-	+	NASA Ames	3.7.2	Jun-98	-	Automatd instrumntatn	Recompile	debug (-g) exec only	
HPCToolkit	+	+	+	+	Rice University	N/A	Mar-04	-	Automate via scripts	Dynamic	Analysis pre node	
HPM Toolkit	+	+	+	-	IBM/AlphaWorks	2.5.4	Mar-04	+	Includes threads tools	Relink	Only AIX platforms	Now IBM HPC Toolkit
KOJAK	+	+	+	+	U of Tennessee	2.0b2	Sep-04	-	Automated analysis	Recompile	Lacks maturity	
PE Benchmarker	+	+	+	-	IBM/AlphaWorks	3.2	Dec-01	+	File conversion utils	Dynamic	No native viewer	
PerfSuite	+	+	+	+	NCSA/UIUC	0.6.1b5	Jun-04	-	View multiple results	Recompile	In beta development	
SpeedShop	?	?	?	-	SGI	1.4.6	Apr-04	+	Memory tools too	Dynamic	pthreads limitations	Linux version in 2006
TAU	+	+	+	+	Univ. Oregon	2.14.1	Oct-04	+	Java & Python	Relink / parse	Steep learning curve	
Profilers												
DEEP/MPI	+	+	+	-	Crescent Bay	2.1a	Jul-02	-	Debugger mode	Recompile	No C++ support	Uses PAPI
Dynaprof	+	+	+	+	Phillip Mucci/UTK	0.9	Nov-04	-	Runtime instrumentatn	Dynamic	Lacks some featurs	PAPI & wallclock only
gprof	+	+	+	+	Part of Unix OS	N/A	N/A	+	Very simple	Recompile	No wallclock time	
Hiprof	+	+	-	+	Part of Tru64	N/A	N/A	+	Directly measure time	Rebuilds exec	Alpha only (retired)	
IPM	+	+	+	+	NERSC	0.8	Jun-04	-	Systmwide MPI profile	Dynamic	Over entire run only	
MPIP (ToolGear)	-	-	-	+	LLNL	2.7	Aug-04	+	Easy MPI profiler	Relink	Only subset of MPI-2	
MPX (ToolGear)	+	+	+	+	LLNL	1.3	Sep-04	+	Expands PAPI events	GUI insertion	Provides estimates	
PapiEx	+	+	+	+	U of Tennessee	3.0	Oct-04	+	Counter multiplexing	Relink	Need shared lib	
Perfex	+	+	+	-	SGI	N/A	N/A	-	Exact event counts	Dynamic	SGI R10000 only	
pgprof	+	+	+	+	Portland Group	5.2.2	Aug-04	+	Measures scalability	GUI or CL	Max 64-procs/16-threads	
Pixie	+	+	-	+	Part of Tru64	N/A	N/A	+	Feedback to compiler	Rebuilds exec	Alpha only (retired)	
prof	+	+	+	+	Part of Unix OS	N/A	N/A	+	Very simple	Recompile	No wallclock time	
SCALEA	+	+	+	+	Askalon Project	1.0.3	Jul-03	-	Also on grid computing	Instrumnt tool	Only FORTRAN	
Tprof	+	+	-	-	IBM	N/A	N/A	+	See system processes	Dynamic	No MPI support	
Vprof	+	+	+	+	Sandia Nat'l Lab	0.12	Mar-02	-	Simple PAPI tool	Relink	Lacks support	
Vtune	+	-	-	-	Intel	2.0	May-04	+	Automated analysis	GUI	No MPI support yet	
Xprofiler	+	+	+	-	IBM	N/A	Sep-04	+	GUI for gprof	Recompile	Easilly overwhelmed	
Tracers												
mpitrace	-	-	-	-	IBM	N/A	N/A	+	Very low overhead	Relink	Not thread-safe	
Performeter	+	+	+	+	U of Tennessee	3.0	Oct-04	+	Real-time monitoring	Modify source	Coarse-grained only	With PAPI
RootCause/Aprobe	?	?	?	-	OC Systems	2.1.3	Jun-03	-	Leaks/trace/coverage	GUI, dynamic	Not parallel-aware	
Sigma	-*	-*	-	-	IBM/AlphaWorks	2.1.1	Dec-03	-	Tune memory system	Dynamic	Single proc only	
Visualizers (GUI)												
Intel Trace Tools	-	-	-	-	Intel	4.0.2.1	Jul-04	-	Fully featured GUI	Relink	Multiplatform future?	Based on Vampir
Jumpshot / MPE	-	-	-	+	Argonne Natl Lab	4.0	Sep-04	+	Est. MPI Overhead	Relink	Lacks some featurs	SLOG2 trace files
Opt	+	+	+	-	Allinea	1.0	Jan-05	-	Interoperable with DDT	Dynamic	Still being designed	Available early 2005
OptiPath	-	-	-	-	PathScale	1.0	Jan-05	-	Suggests improvemnts	Dynamic	Still being designed	Available early 2005
Pablo/SvPablo	+	+	+	+	UIUC	5.2	Nov-02	-	Interactiv instrumenting	GUI insertion	No C++ support yet	
Paradyn	+	?	?	+	U of Wisconsin	4.1	Apr-04	+	Uses Dyninst	Dynamic	Scales slow	
ParaGraph/MPICL	-	-	-	+	UIUC	N/A	Oct-99	-	Varying perspectives	Modify source	Lacks support	
Paraver/Dimemas	+	+	+	-	CEPBA - UPC	3.3	Nov-03	+	Works on BlueGene/L	Dynamic/rlnk	Steep learning curve	
SeeWithin/Pro	-	-	-	-	Verari Systems	1.2	Feb-04	-	Cook-book analysis	Dynamic	Newcommer	
Vampir/GuideView	+	+	+	-	Intel/Pallas	4.0	Aug-03	+	Support for threads	Relink	Multiplatform future?	Replaced with ITA
xmpi	-	-	-	+	Indiana Universty	2.2	Mar-04	-	Educational tool	Dynamic	Needs LAM/MPI	

University of California
Lawrence Livermore National Laboratory
Technical Information Department
Livermore, CA 94551