

An Open Source Performance Tools Software Suite for Scientific Computing

Philip J. Mucci
mucci@cs.utk.edu

University of Tennessee, Knoxville.
SiCortex Inc. Maynard, MA.

Tushar Mohan
tmohan@sicortex.com
SiCortex Inc. Maynard, MA.

Abstract

With the rapid replacement of closed, homogeneous, proprietary HPC systems by heterogeneous, Linux-MPI cluster systems, the state of performance monitoring and analysis tools has become a cause for concern. Proprietary systems, despite their drawbacks, provided consistent tools of high quality. Modern Linux cluster systems, on the other hand, benefit from a wide variety of Open Source tools in differing stages of evolution. Recognizing that Linux clusters are here to stay, SiCortex has taken a unique approach of integrating and enhancing Open Source tools into a production-quality suite. Further, as a tribute to the unrewarded Open Source community developers, and for more pragmatic reasons, such as long-term sustainability, changes made to the tools are fed upstream to the original tool developers. In this paper, we present an overview of the SiCortex tools' suite, and some of the challenges and successes we had in the process of realizing it.

Introduction

In this paper, we describe the development of a performance tools suite for the SiCortex technical compute platforms. A unique aspect to this approach is that the entire suite is based on popular Open Source tools, re-engineered to be of production quality. This approach differs greatly from current models where the tool suite is designed outside industry standards or adapted in accordance with vendor traditions to the current cluster model, which essentially ships with only standard Linux tools. Both these approaches are a cause for concern to the end user and further complicate the process of parallel program debugging and optimization. By incorporating the best-of-breed Open Source technology and combining it with state-of-the-art hardware and rigorous quality control standards, SiCortex has developed a new standard for a Linux/HPC performance tool suite against which current and future technical compute clusters will be compared.

The tool suite needs to have the following components:

- Tools to quickly and easily characterize the performance of serial and parallel applications with little or no effort on behalf of the user.
 - ☒ Direct on/off-core hardware performance counter metrics.
 - ☒ Intuitive and meaningful derived metrics from the above, including computational load imbalance.
 - ☒ Full memory utilization information.
 - ☒ MPI statistics including communication load imbalance.
 - ☒ Patterns of I/O usage.
- Low-overhead statistical profiling of unmodified code on the basis of hardware performance counter interrupts, with a breakdown by file, function and line number (if available).
- A tool to trace, visualize and debug MPI across an arbitrary number of processors.
- A simple and easy to use source code instrumentation package for application developers that provides first-order information about performance.

- A comprehensive instrumentation and performance data management system for complex and comparative performance analysis studies across multiple generations of an application.
- Support for automatic instrumentation of code through library replacement, source code parsing as well as compile-time augmentation by the Pathscale and GCC compilers.

In order to make the tools' suite a reality given limited resources, SiCortex formulated a comprehensive strategy, outlined as follows:

- Leverage the best-of-breed Open Source tools, filling in new functionality where, either previously none existed, or where current functionality was insufficient to support the demands of the users and the functionality of the SiCortex hardware.
- Adhere to the Unix philosophy where one tool should perform one task exceedingly well. Maintain orthogonal functionality between tools, avoiding duplication.
- Engineer straightforward, simple (where possible) and consistent user interface semantics for all tools, avoiding complicated and/or unnecessary graphical user interfaces when text based interfaces suffice.
- Wholly observe the Linux Standard Base methodology for the installation of each tool. This means that the standard directory structure is adhered to, in addition to the presence of man pages, additional documentation and data files in the standard locations.
- Pursue value-added extensions to each tool by adding conditional and configurable support for the advanced features provided by the SiCortex hardware, parallel run-time and operating system.
- Where possible, propagate changes to the code back to their original maintainers along with justifications for the modifications.
- Enforce complete interoperability with the rest of the SiCortex software suite, including the MPICH2-based parallel run-time system, the Pathscale optimizing compiler and the Slurm/Maui batch scheduler system.
- Foster a positive working relationship with the Open Source developers, both with SiCortex and amongst themselves, encouraging unification and sharing of code where appropriate, yet maintaining each project's integrity and ability to specialize.

The SiCortex Performance Monitoring Hardware

The SiCortex node chip consists of six MIPS64 cores, each of which has a small PMU very similar to that found in the MIPS5K. The PMU consists of two 32-bit registers each of which can count 19 events in any combination of four domains; interrupt, supervisor, kernel and user. These events span from instruction mix characterization to external cache events like level 1 and level 2 misses. Only two registers per core was rather limiting in terms of the amount of information one could glean from a single run, without multiplexing the counters. Thus the node chip had additional logic added to support the monitoring of many events simultaneously. The SCB or serial control bus, adds a total of 256 highly configurable counters to the chip, partitioned into 128 buckets of two. The counters do not count simultaneously but rather each bucket pair are sampled for a programmable duration and then the bucket counter is incremented. These counters can measure more than just 'core related' events, but also events related to the memory controllers, the DMA engine, the PCI express controller, the L2 cache controller and coherency engine and the link fabric. The SCB further supports additional functionality such as interrupts on overflow, conditional counting and count histograms (by number of cycles), as well as event address sampling. The latter mode operates by, upon each

increment of bucket pair, deposits the corresponding virtual program counter and physical effective address into a pair of co-processor registers for the corresponding core.

The SCB performance counters were designed with global analysis in mind, where a single user takes over the entire node. However, the current suite of tools and the kernel infrastructure, as well as widely used methodology, does not fit well into this model. As we wanted to use the 'standard' suite of tools, as well as kernel infrastructure, a method was designed to partition the SCB into 6 segments of 42 counters each, each segment associated with a corresponding MIPS64 core.

Perfmon2: A Standardized Linux Kernel Infrastructure for Hardware Performance Monitoring

To date there does not exist an infrastructure for hardware performance analysis in the mainline Linux source tree. This introduces severe integration and support problems for HPC vendors in this space, not unique to any particular architecture, with exception of the IA64. For x86 and x86_64 based systems, the PerfCtr infrastructure has been popularized through its support through PAPI, the Performance Application Programming Interface. The patch is slim, robust and heavily tested and runs on almost any kernel revision. However, the PerfCtr interface has proven somewhat limiting given the more advanced hardware monitoring functionality as can be found on certain members of the x86/x86_64 family as well as the IA64 and PPC series and now, the SiCortex node chip. Furthermore, repeated submissions to the Linux Kernel mailing list raised enough objections over the interface, that it became clear that it did not have a future in the mainline kernel. This has not stopped PerfCtr from becoming popular in the HPC community and in fact, many vendors and integrators regularly patch their kernels with PerfCtr in order to support PAPI based tools. Sadly, for most procurement, OS support by the vendor is directly tied to a specific version of kernel, one that is un-patched and provided in binary form. This prevents any and all hardware performance analysis from being done, unless one resorts to using system-wide tools like Oprofile and Vtune, which take over the counter hardware completely and require root privileges to configure. Neither of those are viable options for large, production HPC systems where the user is running in batch mode.

More recently, progress has been made on the Perfmon2 kernel infrastructure, an effort that grew out of the initial implementation of performance monitoring for the IA64. While still in development, this infrastructure has a number of benefits over PerfCtr. The most important of which are:

- Support for non-counting hardware, like trace buffers and address sampling.
- Kernel-based counter multiplexing, time based and event based.
- Generalized model of PMU resources suitable for any architecture.
- Avoid excessive instrumentation of the context switch handler.
- Robust third-party interface to support analysis through ptrace().
- Extensible through insertion of runtime PMU description tables.

Unlike PerfCtr, where the data structures that are passed to the kernel are highly specific to the underlying architecture, Perfmon2 abstracts this into a general model without paying a significant penalty. As a companion to Perfmon2, is libpfm, which is the library that contains the architecture specific details. Using libpfm, one can ask for specific events to be counted or sampled and the library fills in the structure to be passed to the kernel. In this way, the kernel needs very little information about the underlying PMU beyond what is present in the description tables. This keeps the kernel component lightweight and free of architecture specific code.

Perfmon2 has undergone significant review on the various Linux kernel mailing lists and is

currently being evaluated for a merge upstream. Several important components of the necessary infrastructure have already been merged upstream. At the moment, there are no significant opponents to the adoption of this infrastructure other than the frustrating perception that the needs of the HPC community do not reflect the needs of the community as a whole.

The Perfmon2 infrastructure has been ported to the SiCortex platform Linux kernel. The patch has proved robust and stable and flexible enough to handle the different types of PMU hardware found on the SiCortex node chip. The porting process was relatively easy to the MIPS architecture, the only complication arising from figuring out how to gracefully share the SCB counting resources. As a result of the SiCortex porting effort, Perfmon2 now has support for numerous MIPS processors.

Pfmon

Pfmon[9] was originally written by Stefane Eranian as a test driver for the Perfmon2 substrate and libpfm. It's functionality has continued to mature and now it stands as a tool in it's own right. For the SiCortex platform, the pfmon tool serves the purpose of giving access to the address sampling registers found in each core, filled by counting events from the SCB. As mentioned, certain SCB events can trigger the storage of the program counter and effective address of the current event. Pfmon currently provides the interface to this functionality for the situations where a user really wants to trace the addresses of memory events, specifically L1 and L2 cache misses. This tool gives the user a dump of effective virtual addresses (translated from physical by the kernel extension) to standard output. Unfortunately, compiler support does not yet exist for tagging individual loads with their specific data structures, thus mapping these addresses can be a challenge. We hope to develop this infrastructure further in order to provide more high level and useful functionality.

PAPI

At its core, the SiCortex tools suite is built upon two key components, PAPI[6,7] and Monitor. PAPI is the ad-hoc standard for performance monitoring on HPC systems, with well more than a dozen popular tools – Open Source and commercial – supporting it. It repeatedly shows up on large procurements as a necessary port before acceptance, due to the prevalence of quality tools that use it. For SiCortex, porting PAPI meant porting to the new Perfmon2/libpfm infrastructure. An existing port to the IA64 specific version of Perfmon existed, but the interfaces were different enough that it warranted the development of an entirely new PAPI substrate. As a result of this port, bugs were fixed in PAPI as well as many enhancements made:

- Support for additional memory usage information from /proc.
- Vastly improved GNU configure support.
- Additional high performance timer functionality for the SiCortex platforms.
- Out of the box support for other Perfmon2 platforms (x86_64, x86, PPC and MIPS).
- Additional test cases for better code coverage.
- Standardized install targets corresponding to the Linux Standard Base.

Monitor

Monitor is an infrastructure meant to ease the development of tools that use library preloading. As mentioned, the bulk of the SiCortex tools are targeted to work on unmodified, dynamically linked binaries. The Linux run-time linker has the ability to load a shared library before loading the executable or any dependent libraries into memory. As part of this process, the preloaded library can

execute code and/or replace functions found in other libraries. These features are sufficient to implement a host of passive performance analysis tools. Monitor provides a small library that, when preloaded, executes callbacks to another shared library, one provided by the tool. The callbacks occur on the following events:

- Library initialization/finalization.
- Process initialization/finalization.
- Thread initialization/finalization.
- All variants of fork() and exec().
- Normal and abnormal termination.

The basis for the code in Monitor was initially developed as part of HPCToolkit and then migrated out and enhanced by the author for use in other tools. It has, subsequently, been extended to support handling of OpenMP as found in GCC 4.x and in the Pathscale compiler suite. The library has proven exceedingly robust and portable, working out-of-the-box on many combinations of Linux releases and architectures.

Papiex

Papiex forms the basis of all the end-user tools. It was originally written as part of the PAPI project by Mucci, but now has been greatly enhanced by the authors to meet SiCortex's criteria. It parses arguments and, using monitor, preloads tool-specific shared libraries. The driver, and its argument semantics, is shared among six tools – papiex, mpipex, ioex, gptlex, hpcex and tauex, each of which will be described below. A common feature of these tools is that they provide performance measurements of applications with no modifications, re-compilation, or re-linking needed in the user code. The papiex tool reports aggregate totals from processor performance counters and related metrics that indicate overall program performance and/or bottlenecks. If the application has been compiled with passive instrumentation, as present in GCC and the Pathscale compilers, it can generate performance data for each pair of compiler-generated instrumentation points. Papiex can also provide numerous derived metrics such as IPC, computational intensity and MFLOPS. For parallel applications, each thread and/or MPI task has separate output files as well as summary files for the entire application.

The following is a significantly pruned sample papiex output for a parallel application:

```
[output snipped]
MFLOPS Aggregate (wall clock) ..... 7053.30
MFLOPS (per task)..... 136.62
IPC ..... 0.66

Time:
Wall clock (sec) ..... 169.50
Running Time % ..... 98.98

Instructions:
Memory Instructions % ..... 42.16
Est. Int. Arithmetic Instructions %..... 32.45
FP Instructions % ..... 44.03
FP Arithmetic Instructions % ..... 21.22
FMA Instructions % ..... 8.41
Branch Instructions % ..... 4.17
```

Memory:	
Load/Store Ratio	2.79
Flops per L1 D-cache Miss	28.37
L1 D-cache Hit %	97.52
L1 I-cache Hit %	99.96
L2 Miss %	17.89
L2 Bandwidth MB/s	170.48
Memory Bandwidth MB/s	60.99
D-TLB Hit %	84.67
Branch Misprediction %	30.56
Dual Issue %	28.47
Stalls:	
Est. Total Memory Stall %	17.35
Est. Total TLB Stall %	9.62
Est. Mispredicted Branch Stall %	0.84
Dependency (M-stage) Stall %	0.90
Ideal:	
Ideal MFLOPS (max. dual)	248.65
MPI cycles %	10.84
I/O cycles %	0.00

From studying the output one can conclude that better memory-subsystem performance and cache blocking will help the application. Observe the high mix of integer and memory instructions. The high branch mispredictions are not troublesome as the architecture has a short pipeline and each misprediction costs a single cycle.

Ioex

Ioex draws inspiration from a sample tool called IOTrack[10] by Per Ekman, although ioex was written from scratch for SiCortex. Through library replacement, it traps POSIX and MPI I/O calls to provide statistics about the type and amount of time spent in I/O by an application. It can provide statistics such a read/write rates, block sizes and can classify general seek patterns. The tool does this without tracing and thus has a very low overhead. For MPI programs, with an accompanying Perl script, ioex produces a summary report for statistics across all the tasks in the MPI job.

The sample output below is for a 512-task MPI job of *namd* on an *apoa1* molecule. Observe, that a number of processes (shown in square brackets next to the I/O call) open different device files (*/dev/scdma**) for communication. However, only a single task reads the configuration file and writes the program output.

```
[output snipped]
File: /dev/scdma6
  open [86]
                                calls:          1
                                flags:          O_RDWR
-----
File: /dev/scdma7
  open [82]
                                calls:          1
```

```

                                flags:          O_RDWR
-----
File: /proc/sys/kernel/randomize_va_space
  Fopen [512]
                                args:           r
                                calls:          7.96875          7          39
-----
File: apoal.out.vel
  fopen [1]
                                args:           wb
                                calls:          8
  fwrite [1]
                                % I/O time:     97.91
                                MB/s:          195.29
                                bytes:         4.42676e+06
                                bytes/call:    1.10669e+06
                                calls:         4
                                usecs:         22668
                                usecs/call:    5667
-----
[output snipped]
* numbers in [ ] represent the number of processes invoking the I/O call.
* 3 numbers in a row represent the mean, min and max across all processes.
  When a single number is printed, it is the unique value measured.

```

mpiP and mpipex

The mpipex tool measures data related to MPI operations, providing a clear and concise summary report about MPI usage. Mpipex relies on the existence of a modified mpiP[8] library to generate its statistics. However, unlike the traditional mode of usage of mpiP, mpipex does this by run-time pre-loading of the shared mpiP library, and thus avoids re-linking with mpiP. Since, the output of mpipex is identical to mpiP, we are not reproducing it here.

HPCToolkit and hpcex

HPCToolkit[11,12] is a suite developed by Rice University for the analysis of statistical profiles generated by sampling the PC during hardware performance counter interrupts. It consists of a tool to gather the data, as well as a tool to visualize flat profiles by file, function and line. Furthermore, it includes the 'bloop' component that can recover source code structure (like loop nests) from the application binary and associate that with profile data. The pattern of usage of HPCToolkit is somewhat similar to that of gprof. Hpcex has added standardized argument processing as well as data output to HPCToolkit. As the output of hpcex and its accompanying tools is identical to that produced on other systems by HPCToolkit, we are not reproducing it here.

GPTL and gptlex

GPTL provides print-friendly, hierarchical, function-level performance data using PAPI. It can provide counts for both arbitrary user-inserted and compiler-generated instrumentation points. It can also be used to provide a run-time call-graph. Gptlex is an optional, run-timer driver for the library

that allows the user to control the parameters of the instrumentation without modifying the source code.

TAU and tauex

TAU[1,2] is the most comprehensive the performance analysis tool that SiCortex provides. TAU provides for both aggregate performance data as well as traces at virtually any level of detail. This data can be analyzed by paraprof and pprof, tools included with TAU, as well as fed into other performance looks like KOJAK and Vampir-NG. TAU provides a number of ways to gather data, including automated source code instrumentation through the Program Database Toolkit (PDT) as well as library replacement and hand instrumentation. Paraprof also provides for sophisticated visualization of performance data from multiple perspectives, in addition to providing for experiment management. SiCortex has been exceedingly impressed with TAU's feature set however it felt it lacked significantly on the usability front. Thus, in conjunction with a contract with Paratools, TAU's support organization, it made numerous modifications including integration with the tauex driver using the papiex framework. The end result being:

- Users who have instrumented source either by hand or through the PDT need only to link with one library instead of choosing one of dozens that can be built. These libraries represent the various combination of TAU measurement and execution mode. (tracing, aggregate, call-stack vs. MPI, OpenMP, pthreads).
- At run-time, the tauex driver pre-loads the appropriate version of the TAU library depending on the specified options. Sensible defaults are chosen in the absence of those options.
- Traces for MPI and hardware performance counter data are stored in OTF[3], the Open Trace Format resulting in massive improvements in time for processing and analysis.
- Applications automatically instrumented by the GCC or Pathscale compiler suites can use the tauex driver to transparently generate data for TAU.

Vampir-NG

Of particular concern to SiCortex was the lack of a quality Open Source message tracing and visualization engine. MPICH2's Jumpshot and MPE were evaluated and found lacking on numerous aspects, most notably the scalability of the display and the usability of the GUI. Vampir[4,5] was previously the standard for this work, however the product is now wholly owned by Intel and known as the Intel Trace Analyzer. Fortunately, work on Vampir-NG, a successor to Vampir, was progressing nicely at U. Dresden and SiCortex made the decision to approach them through Paratools about offering a version of VNG for the SiCortex platform. It is worth noting that Vampir, unlike the other tools mentioned in this paper, is not Open Source software. Vampir is unique in that it provides the user with the ability to visualize and debug MPI issues that arise during execution. VNG provides a number of enhancements over and above the commercial version of Vampir, now owned by Intel as the Intel Trace Analyzer. These include but are not limited to:

- Vampir-NG has concurrent trace I/O capabilities using the Open Trace Format. With SiCortex's parallel file system, this results in significant advantages over reading the entire trace on one CPU.
- Vampir-NG is a completely new parallel implementation of the classic serial Vampir tool. This

results in super-linear speedups for many visualization tasks. With more CPUs, more memory is available for trace processing. Traditionally, as you expand the number of CPUs and increase the total number of events in each CPU (longer duration of traced execution), more memory was required than was often available to a single CPU. Vampir-NG lets you work with a full trace instead of needing to trim the resolution of the problem and/or relying on periodic summarization. Vampir-NG is the only viable solution to visualizing parallel performance traces at scale.

- Its client/server architecture allows it to quickly browse large data volumes from remote sites. Vampir tended to be quite slow in this regard because the GUI runs natively.
- Vampir obscures dense inter-process communication patterns when the message density exceeds a certain threshold. Vampir-NG, on the other hand, produces an indicator of such a volume and facilitates zooming upon that region and the exploration of individual messages. This is also true for process time-line displays that can show levels of nesting as well as hardware counter information.
- Vampir-NG has a thumbnail of the entire process. This allows you to zoom in to a segment of the overall time-line. Vampir lacks this capability.
- The call-path display in Vampir allows you to click and expand a call-path, but Vampir-NG enhanced this by separating the child and a parent displays, facilitating further exploration of parent-child relationships.

Licensing and sources

The performance tools provided on the SiCortex platform, and discussed in this paper, with the exception of Vampir (and Vampir-NG) are Open Source tools. Users are encouraged to download the sources from the SiCortex website or the website of the original author(s).

Conclusion

The proliferation of Linux clusters without accompanying high-quality performance tools poses a severe impediment to developing high-performance applications. In the past, proprietary closed systems made the development of integrated tools possible. With new, innovative, Open Source tools available, it makes sense to integrate them into a consistent suite, with attention to robustness and usability. We have integrated the tools mentioned in this paper, however wrinkles need to be smoothed. We have met with many successes in integrating our changes upstream, to the wider benefit of the Open Source community. In the future, as can be expected, we will add new features to our existing tool set, while scouring for new tools, with orthogonal functionality, to be blended in.

References

1. Malony A, Shende S. Performance Technology for Complex Parallel and Distributed Systems. *Proc. Third Austrian-Hungarian Workshop on Distributed and Parallel Systems*
2. Malony A, Shende S. TAU: Tuning and Analysis Utilities. *Los Alamos National Laboratory Publication LALP-99-205, November 1999.*
3. Knüpfer A, Brendel R, Brunst H, Mix H, Nagel W E. Introducing the Open Trace Format (OTF). *International Conference on Computational Science (2) 2006: 526-533*
4. Knüpfer A, Brunst H. Nagel W. High Performance Event Trace Visualization. *PDP 2005: 258-263*
5. Brunst H, Kranzlmüller D, Nagel W. Tools for Scalable Parallel Program Analysis - Vampir VNG and DeWiz. *DAPSYS 2004: 93-102*
6. Dongarra J, Malony A, Moore, S, Mucci, P, Shende, S. Performance Instrumentation and Measurement for Terascale Systems. *International Conference on Computational Science 2003, Melbourne, Australia, June 2003.*
7. Dongarra J, London K, Moore S, Mucci P, Terpstra D, You H. Zhou M. Experiences and Lessons Learned with a Portable Interface to Hardware Performance Counters. *Lecture Notes in Computer Science, Springer-Verlag, Heidelberg, Volume 2723, pp. 53-62, January, 2003.*
8. Vetter J S. Dynamic Statistical Profiling of Communication Activity in Distributed Applications. *Proc. ACM SIGMETRICS: Joint International Conference on Measurement and Modeling of Computer Systems, ACM. 2002.*
9. Eranian S. The Perfmon2 Performance Monitoring Interface. *Ottawa Linux Symposium, 2006.*
10. Ekman P, Mucci P. IOTrack. *Fifth Workshop on Linux Clusters for High Performance Computing, October 18-21, 2004.*
11. Strout M M, Mellor-Crummey J, Hovland P. Representation-independent program analysis. *Proceedings of PASTE '05: The 6th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering, pp. 67—74, 2005*
12. Mellor-Crummey J, Fowler R, Marin G, Tallent N. HPCView: A tool for top-down analysis of node performance. *The Journal of Supercomputing, 23, 81-101, 2002.*