Lawrence Berkeley National Laboratory
Lawrence Berkeley National Laboratory

Title
Performance monitoring of parallel scientific applications

Permalink
https://escholarship.org/uc/item/55t52673

Author
Skinner, David

Publication Date
2005-05-01

eScholarship.org Powered by the California Digital Library
University of California

Performance Monitoring of Parallel Scientific Applications

David Skinner
National Energy Research Scientific Computing Center
Lawrence Berkeley National Laboratory

Abstract.

This paper introduces an infrastructure for efficiently collecting performance profiles from parallel HPC codes. Integrated Performance Monitoring (IPM) brings together multiple sources of performance metrics into a single profile that characterizes the overall performance and resource usage of the application. IPM maintains low overhead by using a unique hashing approach which allows a fixed memory footprint and minimal CPU usage. IPM is open source, relies on portable software technologies and is scalable to thousands of tasks.

Introduction and Motivation

The creation of a portable, scalable, and low overhead profiling infrastructure for HPC applications is driven by the need to both optimize HPC applications on a given architecture and to characterize the performance of applications across architectures in order to make optimal matches of HPC workload and HPC architecture.

Overview and Methodology:

The basic idea of IPM is to provide an easy to use means of collecting performance data from HPC codes in a production environment. On most HPC platforms there is no parallel layer for HPM. This is unfortunate and IPM partly remedies this problem.
In order to do this the IPM infrastructure must be parallel aware, being able to coordinate the performance data from multiple concurrent sources in a parallel job.

Since most of the workload we are interested in uses MPI for parallelism we have focused on implementing IPM through the name shifted profiling interface to MPI. The use of the profiling interface to MPI is of widely recognized value in profiling MPI codes. The name-shifted or (PMPI) interface allows each MPI call to be "wrapped" by user code or tools that time or otherwise profile the communication performance. The principal mechanism by which IPM attaches to the parallel job is through this PMPI interface. We have also implemented IPM for non MPI codes to a lesser degree.

Once the application is running IPM quietly (using little CPU or memory) collects a performance profile which is written when the application terminates. In subsequent sections a more detailed description will be given of what is meant by a profile.

IPM is a portable profiling infrastructure for parallel codes. It provides a low-overhead performance summary of the computation and communication in a parallel program. The amount of detailed reported is selectable at runtime via

environment variables or through the MPI_Pcontrol interface. IPM has extremely low overhead, is scalable and easy to use requiring no source code modification.

IPM is portable running on IBM SP's, Linux clusters (MPICH/MVAPICH), SGI Altix, NEC SX6, and the Earth Simulator.

IPM brings together several types of information important to developers and users of parallel HPC codes. The information is gathered in a way the tries to minimize the impact on the running code, maintaining a small fixed memory footprint and using minimal amounts of CPU. When the profile is generated the data from individual tasks is aggregated in a scalable way. The time spent aggregating depends on the amount of detail collected and the number of tasks. On an IBM SP applications running on 4096 tasks can be aggregated in approximately 5 min. This is a one time cost occurring when the application terminates.

The 'integrated' in IPM is multi-faceted. It in part refers to binding the above information together through a common interface and also the integration of the records from all the parallel tasks into a single report. On some platforms IPM can be integrated into the execution environment of a parallel computer. In this way IPM profiling is available either automatically or with very little effort. The final level of integration is the collection of individual performance profiles into a database which synthesizes the performance reports via a web interface. This web interface can be used by all those concerned with parallel code performance, namely HPC center users and HPC center staff and managers. Optionally the IPM logfiles are available to be parsed in a variety of tabular and graphical ways. Examples of parsed IPM data and the logfile format are given in subsequent sections.

IPM is Open Source software, available under the GNU LGPL at

http://www.nersc.gov/projects/ipm/

Application Chronology and Context:

The most detailed characterization possible of a parallel code is a complete chronology of all performance **events** occurring during its execution. In this work, we define performance events to be intervals of time during which something of interest is happening and the performance **metrics** associated with that interval. Performance metrics could be on-chip hardware performance counters, switch counters, or resource utilization statistics from the operating system.

An application **trace** is a list of time stamped performance events

An application **profile** is a summary description of performance events

If it were possible to accumulate performance traces for HPC applications at large scale this would provide an exhaustive description of all performance events encountered. In practice tracing is of limited feasibility at large scale. The extensive properties of traces become burdensome both in time and concurrency as data volume, memory foot print, and CPU required to accommodate tracing do not scale.

Additionally the extensive properties of a trace often do not lead to increased understanding of the application performance. In the design of IPM we considered application **context** to be generally more important than application **chronology**. By context we mean that the profile records enough unique features of the performance event so that it may be identified in the code, algorithm, or overall operation of the program.

There are cases where event chronology is important such as intermittent performance glitches. IPM does support a limited type of "snapshot" tracing which is not described in this work, but is detailed in the IPM documentation.

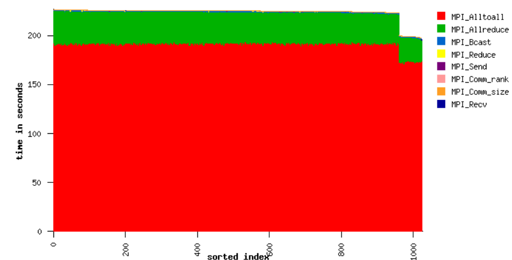
In order to improve contextual clarity of performance events, IPM optionally supports the idea of regioning. Users may introduce MPI_Pcontrol statements that tag certain code regions with a specified name. That name becomes part of the contextual details which identify the performance event.

Along with the contextual information (e.g., MPI function calls and their arguments, code regions, callsites) IPM stores performance information such as HPM counters, timings, and statistics. The hashing strategy used is described in a subsequent section.

If the number of unique performance event contexts exceeds what is manageable within ~1MB a failover strategy is used that decimates contextual information for events which rank low in total time.

Load Balance:

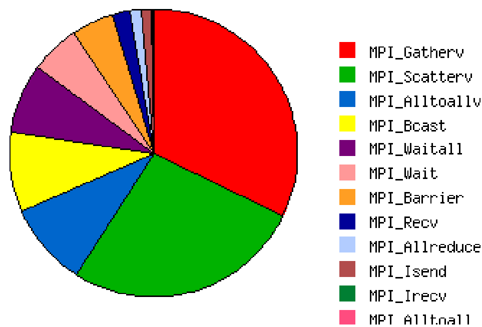
At the coarsest level of detail one may detect load imbalance in a parallel application by tracking performance metrics across tasks. For performance metrics such as %communication time, GFLOPs, memory usage etc. a non-uniform decomposition of the problem across processors leads to discontinuities. This is particularly the case for strongly synchronizing parallel codes where all tasks must wait until each task completes some section of the code. An example of load imbalance detected through IPM is shown below for a 1024 way code. On the x axis are all the tasks/ranks sorted by amount of communication time. On the y axis is the metric of time spent in each MPI call. For this code most time is in MPI_Allreduce and MPI_Alltoall. The discontinuity shows that 10% of the tasks spend ~25 seconds less time in these MPI functions than the other ~900 tasks. From this profile it is likely that these ~100 tasks have more compute load than the rest and arriving at the MPI_Allreduce "late" cause the other ~900 tasks to stall. This can be confirmed by examination of the HPM counters which show a similar but reversed discontinuity in the number of FLOPs executed by each task. The impact on the parallel efficiency and scalability from this type of load imbalance can be enormous.



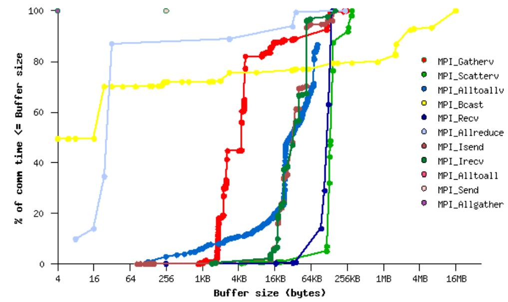
IPM makes detection and detailed quantification of load imbalance relatively easy. From its use so far IPM has had the greatest impact on improving the performance of NERSC codes by treating load balance issues.

Communication Type and Volume:

A high level characterization of a parallel application can be obtained by examining the bulk time spent in different parts of the MPI library. By default IPM gives a breakdown of MPI, HPM, and switch events in tabular and graphical form. The following figure is output from the IPM parser that shows the relative amount of time spent in each MPI call across all tasks in a parallel code.



More detailed analysis is of course needed to make progress in understanding and optimizing a parallel code. Message size distributions are the next most specific characterization of the parallelism in a code. By examining the time spent in each MPI call as a function of buffer size one can easily identify the message sizes critical to the performance of the code.



These graphs and the ones that follow are largely taken from the parsed output of IPM logfiles. It is not easy to show the full display without including a screen shot which does not easily fit in the format of a paper. Examples of the full output from IPM profiling from real HPC applications is given at

http://www.nersc.gov/~dskinner/ipm/ex1

http://www.nersc.gov/~dskinner/ipm/ex2

http://www.nersc.gov/~dskinner/ipm/ex3

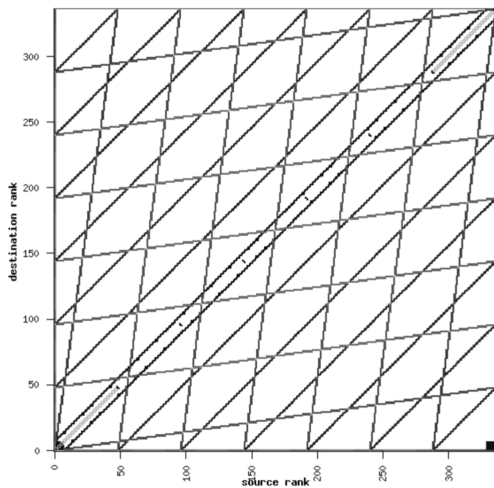
Communication Topology:

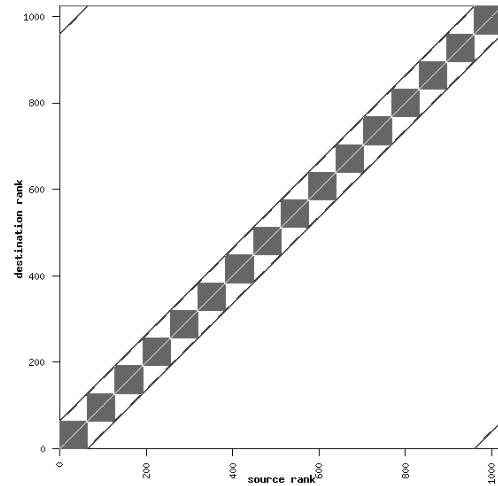
On aspect of parallel code performance that is often overlooked is the topological nature of the messages sent and received between each task. Many HPC hardware resources have some inherent hierarchy of latencies and bandwidth, e.g. SMP clusters. As such the placement of the tasks in the machine should be done in a way to optimize the usage of fast resources over slow ones. This is especially important for architectures such as BlueGene which have large toroidal interconnects.

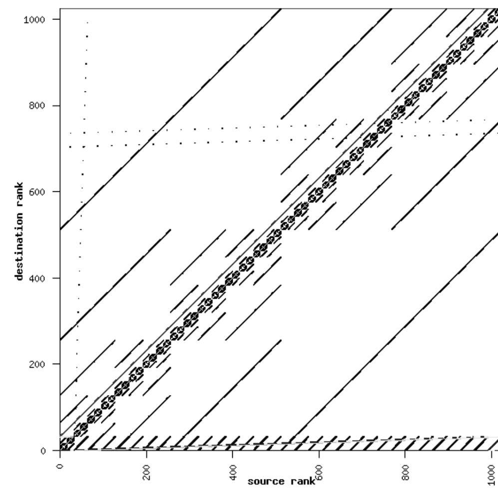
Three example point-to-point communication topologies are shown in the figures below. The x axis is the source rank (task) and the y axis is the destination rank. The shading is darker

in proportion to the volume of data exchanged between the two tasks. It is also possible to render this data in terms of communication time or data movement rate.

A wide variety of application topologies have been observed in the NERSC workload. More detailed analysis of this data is underway but will take time to complete. In the short term it is possible to immediately exploit this information in the creation of optimal task-processor geometries for a given machine. In the long term understanding of communication topologies important to the HPC community may contribute to better understanding the architectural requirements of future machines.



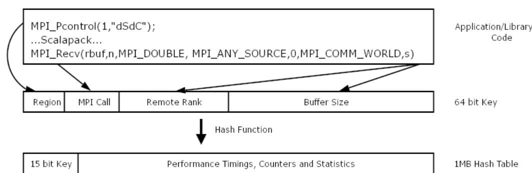




Hashing Methodology:

IPM relies on a hash function to map performance events into a fixed size table. Currently the hashing methodology used is double open-address hashing based on a large prime number. The core idea is using hashing is to be able to efficiently and quickly map performance events onto a data structure which is not extensive in time or the number of events.

The number of CPU cycles required to compute the hash is quite small, relying on three bit-shift and XOR mask operations. A depiction of the hashing process is given in the following figure:



As performance events are collected, keys are computed and statistics recorded. For each unique key we record the number of events, the total time spent in that type of event, as well as the minimum and maximum time spent on any one event of that type.

The final outcome from this is a list of performance events which detail the execution of the code. This information is written to a logfile described in the next section.

XML and Data Analytics:

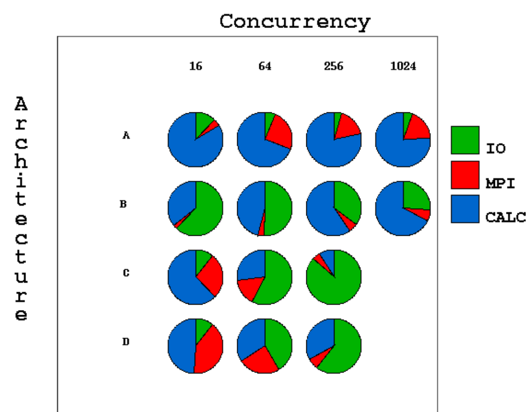
All too often HPC architectural comparisons are made in the absence of "apples to apples" performance metrics. The information complexity of describing "How code X performs on architectures A and B" is remarkable high when one considers the numerous details that contribute to application performance, e.g., memory per task, tasks per node, environment variables, etc.

The IPM logfile records more that just the performance event hash table. The environment variables, hostnames, executable information, etc. are also recorded and provide a more robust way of answering questions after the fact about how a code performs.

By using a regular and extensible syntax for performance profiles those profiles become a more comparable and sharable information product that can serve as a reference point in making performance comparisons. The syntax of the IPM logfile format is described in the IPM documentation. It follows an XML approach to describing the execution environment, task topology, and the hash table of performance events.

By virtue of its portability and by using a standardized schema based on XML for performance profiles, IPM enables clear cross architecture comparisons within the context of the performance of a specific application. As it has been wisely recognized that "performance in the absence of a workload is meaningless" the IPM logfile format seeks to bring performance and workload together in a formalized systematic way.

The figure below shows the parallel scaling of the relative amount of time spent in file I/O, MPI, and calculation across four modern HPC architectures.



Application Complexity:

Future Work:

Conclusions:

References:

Rolf Rabenseifner, Peter Gottschling, Wolfgang E. Nagel, and Stephan Seidl:

"Effective Performance Problem Detection of MPI Programs on MPP Systems: From the Global View to the Details." ParCo99

Jeffery Vetter and Dong Ahn Lawrence Livermore National Laboratory) Scalable Analysis Techniques for Microprocessor Performance Counter Metrics, SC2002

http://www.llnl.gov/CASC/mpip/