Parallel Performance Analysis on Cray Systems

Kevin Roy, *The University of Manchester*

ABSTRACT: *Solving serial performance problems is very different to solving the performance problems on parallel codes. In many cases, the behaviour of an application can depend on the number of processors used. Some parts of the code scale well and others do not; when we look at parallel codes it is these nonscaling parts that are of most interest. Identifying the components and when they become a problem is not often an easy task. In this talk, we present some software that interfaces to existing Cray performance analysis tools and presents clear and easy to view information.*

KEYWORDS: PAT, scaling, performance tools, Cray

1. Introduction

The Need for Analysis

The use of HPC is now common place and an integral part of today's research and industry. The need to solve increasingly larger problems with greater accuracy has driven the HPC market; many research groups have their own facilities and this has been made possible through the availability of commodity clusters. Recent surveys show that the University of Manchester has over 30 clusters currently in use.

This availability has lead to increased numbers of applications in use, particularly in the academic sector. New applications often have the same problems of suboptimal performance and the use of commodity clusters often hides this problem as it can often be easier to use larger system sizes. However this approach can reach a natural limit and the focus then needs to be on improving the application not just the system it is run on. There are many reasons why an application might perform poorly:

- Interconnect to CPU performance of the system does not match what the application needs, e.g., poor interconnects can often leave the CPU idle, waiting for communication to occur.
- The scientific programmer does not always have the set of skills required to scale codes up to very large systems.
- It has become increasingly easy to write programs for today's HPC systems, but in many cases it has become increasingly difficult to get better performance.
- Physical differences between systems can produce very different performances on different architectures.
- The constant change in choice of HPC platform at research institutes coupled with the evolution of microprocessors has meant continual need to ensure performance through code profiling.

As computational scientists we need to overcome these issues through a greater understanding of the hardware and how it interacts with the application, this is where profiling is useful.

Using Tools

Tools are essential for profiling, whether it is simple timing routines placed in a code by the programmer, something more elaborate provided by the hardware vendor or purchased from an external source. Tools enable us to quickly analyse our applications and focus on areas that will provide the greatest improvements.

In section 2, we survey some existing products that are available looking at their usefulness in profiling with particular emphasis in parallel profiling. Section 3 looks at what is missing in these tools and why this paper discusses some new software. The software is described in section 4.

2. Description of Existing Performance Tools

There are a number of tools that available supplied as either freeware, vendor supplied or purchased as a separate product. Here we look at a couple of these products:

- PAT/Apprentice
- OpenSpeedShop/Speedshop
- Gprof
- OPT
- Vampir/ Intel Trace Tools
- Vtune

PAT (Performance Analysis Tool) and Apprentice are provided with Cray systems. These are very good tools and have evolved over many years producing a very stable effective environment however recompilation is needed.

SpeedShop from SGI is in the same position; it is a good effective tool. OpenSpeedShop is also available, as an open source project it is more portable but restricted to IA64 systems.

Gprof is one of the most portable in the list and is available through Linux, but focuses on call graph information rather than drilling down to line level information.

OPT is also a portable, fully supported and an easy to use front end, but those features come at a price as this is a commercial product.

The Intel Trace tools and Vtune are Intel tools and not available across a range of platforms.

Software	Cost	Portable	Graphical	Serial	Parallel
PAT	Y	N	N	Y	Y
Apprentice	Y	N	Y	Y	Y
SpeedShop	N	N	N	Y	S
OpenSpeedshop	N	S	Y	Y	S
Gprof	N	S	N	Y	N
OPT	Y	Y	Y	Y	Y
Vampir/Intel Trace tools	Y	N	Y	N	Y
Vtune	Y	N	Y	Y	S

3. A New Tool

This work started as part of a porting exercise required with a machine upgrade. Many of the applications had not been examined and analysed for a number of years, so the porting process provided an excellent opportunity to examine the codes in great depth, with the aim of providing a more computationally efficient and scalable code.

However the existing tools did not provide information in a convenient manner or more importantly, quick enough. What was required was a simple tool that could analyse simple serial profiles to obtain further insight into serial and parallel performance. This quick insight then provides opportunities to change and alter the code and re-examine, the final goal was to provide a simple mechanism to display results graphically for reports to show code and function improvements. A further key feature of this new tool is portability and cost

In summary the following features were a requirement of the application:

- Portability of application
- Applicability of the application to multiple systems and profiling tools.
- Build on existing tools as much as possible (particularly profiling tools).
- Provide a simple and quick to use interface to provide analysis and insight into the performance and scalability of the application.
- Produce graphical displays of scalability and performance for use in reports.

4. The Software

The software was developed with the QT programming toolkit for reusability and portability of the graphical part of the application. This led to the development of a C++ application.

The development strategy was based around:

1. Getting the performance data into the application.
2. Being able to manipulate and view the data and new and informative ways.
3. Being able to compare multiple runs of a parallel application
4. Being able to visualise the data.

The software was originally designed for the SpeedShop package for SGI Origin machines which was a well understood package. The application has recently been ported to the Cray systems. The format recognised on Cray is PAT which produces information of the form shown in figure 1. This information was read into the application and stored in a data structure that could be

```
Time% | Cum.Time% |        Time |      Calls |Experiment=1
                                              |Function
                                               |PE='HIDE'

 100.0% |    100.0% | 1666.872000 | 5497510788 |Total
|----------------------------------------------------------
--
|  34.5% |     34.5% |  574.526869 |         60 |pdpstrf_
|  20.4% |     54.9% |  339.842347 | 1736599552 |pdrand_
|  15.4% |     70.2% |  255.942290 | 1821932904 |lmul_
|  15.2% |     85.4% |  253.131299 | 1820747972 |ladd_
|   4.8% |     90.2% |   79.594311 |         60 |pdmatgen_
|   3.4% |     93.6% |   56.476741 |    2863492 |MPI_Bcast
|   2.0% |     95.6% |   33.271027 |    7038416
|MPI_Type_commit
|   1.9% |     97.5% |   32.384416 |    2087458 |MPI_Recv
|   1.0% |     98.5% |   16.866056 |   82962768 |jumpit_
```

manipulated expanded, should it be required.

Figure 1 Sample CrayPAT output

User Interface

A user interface needs to be intuitive to use and provide simple access to all the data required. A decision was made to use tabbed displays with the result from a parallel run in a single tab. The other tabs can then be used to store different runs, these can be either

- Runs with a different processor counts to compare scalability
- Runs after optimization (potential saving a history of optimizations), to show the improvements made.
- Using the same processor count but with different decompositions, again to see what parts of the code are affected most by changes in the decomposition.

Figure 2 shows the interface with re-nameable tabs for easy identification of purpose, rows in the display show functions within the profiled program with statistical data (average and variance) across all profiles within the run. The rows open out providing the full data for each profile in the run if required.

The Reports

Much information can be gleaned from the numeric summary data, load balancing and single processor performance problems can be easily identified. In order to gain insight across different runs the graphical reports need to be run. By highlighting routines in the user interface and then choosing a graphical report from the menu system, graphs can be obtained like those in figures 3 and 4.

Figure 3 shows a classical scalability plot for the highlighted routines, figure 4 shows a histogram of performance which can be very useful when comparing profiles of optimized against unoptimized codes or runs where the distribution is changed.

The Output and I/O

The defining purpose of this application is to help provide insight for code optimization and to demonstrate code improvements that have been made; all of which would be impossible without being able to get data into the system and to store data that is produced.

The application uses its own data format for storing profile data. This decision was essential, it simplifies the I/O functions in the application, matches the data type in use in the application and has the potential to expand as new fields are required. It was also needed for portability and neutrality; the application can be used to compare runs across different architectures.

The application also provides a number of different ways of exporting data tables that are used to produce the reports:

- HTML Table
- Excel Table (CSV file)
- LaTeX Table

This is coupled with options to display the reports:

- EPS
- Direct printing
- Jpeg (through QT support)

5. Conclusion

ParProf has been very effective and delivers on all its major features, development time is the only factor stopping this expanding. The screenshots from figures 2, 3 and 4 show it in operation. There is a future in this tool but it still requires development work in a couple of independent directions.

- Separate

Here will be an examination of the effectiveness and future of this tool.

Plus points and minus points

Things for the future. – Perl abstraction layer.

Acknowledgments

The authors would like to thank colleagues in Manchester and collaborators in CSCS for their experiences thoughts and use of Cray supercomputers.

About the Authors

Kevin Roy is team leader of the CSE team at Manchester Computing, which helps provide local and national HPC services. He has worked in HPC for over 7 years on a variety of systems including the recent Cray T3E, XD1, and XT3 systems as well as past CUG conferences in 2001 and as host in 2002. Kevin can be reached at G49.3, Kilburn Building, The University of Manchester, Oxford Road, Manchester, M13 9PL and via email at Kevin.Roy@manchester.ac.uk.