Parallel Performance Analysis on Cray Systems

Kevin Roy
University of Manchester
Cray User Group
8-11 May 2006, Lugano, Switzerland
Overview

- **Author**
  - Kevin Roy
  - Kevin.Roy@manchester.ac.uk

- **Background**
  - What we do and why we do it.
  - Why do we need to profile
  - What I wanted to achieve

- **The tool itself**

- **Summary and future directions**
Background
Manchester Computing Services

- MIMAS – National Datasets provides services to over 250 institutions.
- CSAR – A national HPC service to UK academia.
- NGS – National Grid Service
- AGSC – UK AccessGrid support centre
- ESNW – eScience North West
Manchester Computing Research

- Advanced Virtual Prototyping Research Centre (AVPRC)
  - virtual reality in engineering with ‘real-time’ finite element analysis software

- Internationally successful RealityGrid project

- Data intensive computing
  - Supercomputer Data Mining
  - NACTeM, NCeSS, MIMAS, NGS;

- Exploring role of new technologies - FPGA, Cell, Clearspeed, etc

- Manchester Visualization Centre (30 years)
  - AVS/Express (MPE, Parallel Toolkit)
  - Immersive visualization driven by SGI Altix
  - Passive Stereo Lab integrated with AccessGrid

- International grid projects
High Performance Computing services since 1948

- Williams-Kilburn Baby
- CDC7600/ICL19 04S
- CDC7600/ICL19 06A
- CDC Cyber 205
- VP1200
- VPX 240/10
- Cray T3E (576)
- Cray T3E (816)
- SGI Altix (256)
- SGI Altix (512)
- SGI Origin3000 (512)
- Dell EM64T (2000)

FLOPS (log)
Why Profile?

- **Significant investment is made in HPC systems**
  - We should ensure that they are used efficiently

- **Efficient use is affected by numerous factors**
  - Inefficient coding (such as loop orderings).
  - Applicability of the code to the hardware.
  - Quality of the compiler.

- Profiling allows us to find the problem areas in codes and work on those.
Motivation

- Machine upgrade
- Many applications had not been touched for years
- Porting exercise gave opportunity to improve the codes

Problem:
- Large number of applications, short amount of time

Needed to get in-depth information quickly and simply
- Existing tools provided all the necessary information but it was time consuming looking for it.
Motivation

- Wanted to improve scalability of code as well as serial performance.
  - Poor serial performance is easy to spot using profiles
  - Poor scalability is harder to spot and requires analysis of multiple profiles.
- I needed to report on each application worked on
  - Part of any report in evidence of performance improvements
- Good tools are expensive. They didn’t do quite what I wanted either.
- I enjoy writing little applications like this.
  - Fun to start a new application from scratch
  - Opportunity to learn new things
The Tool
What I wanted:

- My application needed to be portable
- Needed to build on system profilers.
- Need access to all the data if needed
- Quick and simple interface
- Needed to compare multiple profiles on different processor counts
- Needed to compare many revisions of the code against each other.
- Needed to compare different processor decompositions
- Sorting
- Graphical performance charts
- Output to external formats for reports
- Data comes in from system profiler
- Best option is text based profiles – PAT
- Data is incorporated into internal data structure

<table>
<thead>
<tr>
<th>Time%</th>
<th>Cum.Time%</th>
<th>Time</th>
<th>Calls</th>
<th>Experiment=1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Function</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PE='HIDE'</td>
</tr>
</tbody>
</table>

<p>| | | | | | |
|        |           |       |       |   |   |</p>
<table>
<thead>
<tr>
<th>100.0%</th>
<th>100.0%</th>
<th>1666.872000</th>
<th>5497510788</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>34.5%</td>
<td>34.5%</td>
<td>574.526869</td>
<td>60</td>
<td>pdpstrf_</td>
</tr>
<tr>
<td>20.4%</td>
<td>54.9%</td>
<td>339.842347</td>
<td>1736599552</td>
<td>pdrand_</td>
</tr>
<tr>
<td>15.4%</td>
<td>70.2%</td>
<td>255.942290</td>
<td>1821932904</td>
<td>lmul_</td>
</tr>
<tr>
<td>15.2%</td>
<td>85.4%</td>
<td>253.131299</td>
<td>1820747972</td>
<td>ladd_</td>
</tr>
<tr>
<td>4.8%</td>
<td>90.2%</td>
<td>79.594311</td>
<td>60</td>
<td>pdmatgen_</td>
</tr>
<tr>
<td>3.4%</td>
<td>93.6%</td>
<td>56.476741</td>
<td>2863492</td>
<td>MPI_Bcast</td>
</tr>
<tr>
<td>2.0%</td>
<td>95.6%</td>
<td>33.271027</td>
<td>7038416</td>
<td>MPI_Type_commit</td>
</tr>
<tr>
<td>1.9%</td>
<td>97.5%</td>
<td>32.384416</td>
<td>2087458</td>
<td>MPI_Recv</td>
</tr>
<tr>
<td>1.0%</td>
<td>98.5%</td>
<td>16.866056</td>
<td>82962768</td>
<td>jumpit_</td>
</tr>
</tbody>
</table>
- Renameable tabbed windows for each run
- Function list and summary info
- Expandable functions giving information for each profile
- Derived statistic information
Interpretation

- From here we can see load balancing
  - high differences between max and min times per function
- We can also easily identify poor serial performance
- Fully sortable to aid discovery (standard feature in QT list views)
- Can quickly skip between different runs using the tabs
- Collections of profiles can be saved for easier retrieval.

- We don’t see any more information here than profiles provide but accessing the information is quicker.
The graphs are the most essential part of this.

- My previous methods involved manually copying data to excel or Matlab to generate the graph.
- Slow and laborious

I identified two key things I wanted

- Scalability plots at a function level
- Performance plots (potentially the most wide ranging in terms of use).
Scalability Plot

- Data output icons (later)
- Scalability curve
- Legend
- Scalable routines
- Non scaling routines
- From here we see two small routines (on lower processor counts) will dominate at even higher processor counts.
From the previous plot we can extrapolate the effects at higher processor counts.

- Perhaps the full system is not available yet.

If 66 processors are sufficient to run the code on then all data views give all the necessary information.

If we need to improve scalability we should start looking at the lower routines as they will become dominant.
- Performance chart

- Here we compare a 4x4 decomposition against a 8x2 decomposition.

- We can quickly analyse the behaviour

- Also useful in comparing different revisions of the code
Exporting Data

- Viewing the data and graphs gives the ability to assess and places to target for optimization.
- Exporting the graphs is necessary for reports and showing others (including code owners).
- Data can be exported to
  - HTML table of selected data
  - Excel file (CSV) of selected data
  - LateX table of selected data
  - Text output of selected data
  - EPS
  - JPEG
  - Printer
The Summary
Conclusion

- This served its purpose and more!

- I would still like to add more features time permitting to improve functionality and portability
  - Split up GUI from code to read in profile
  - Add line level data
  - Automatic analysis and problem highlighting
  - Support for other tools on other systems (probably as needed).
  - Support for other output formats (e.g., generate Matlab program to draw graphs).
  - Add a hierarchy of tabs (would allow comparison of runs with different inputs or on different systems).
  - Now I’m getting close to being able to store and retrieve information on every run of every code with every input set on every compiler on every system that I have run it on.
Combining the strengths of UMIST and The Victoria University of Manchester