MANCHESTER
1824

The University
of Manchester

Parallel Performance Analysis on Cray Systems

Kevin Roy
University of Manchester

Cray User Group
8-11 May 2006, Lugano, Switzerland

Combining the strengths of UMIST and
The Victoria University of Manchester

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

MANCHESTER
1824

The University of Manchester

Background

Manchester Computing Services

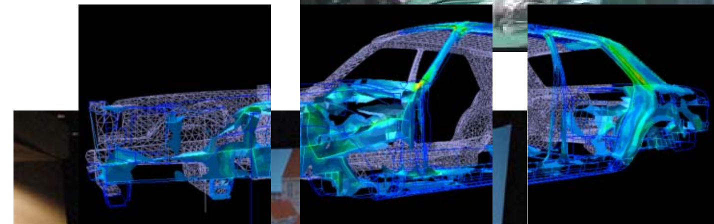
- MIMAS – National Datasets provides services to over 250 institutions.
- CSAR – A national HPC service to UK academia.
- NGS – National Grid Service
- AGSC – UK AccessGrid support centre
- ESNW – eScience North West

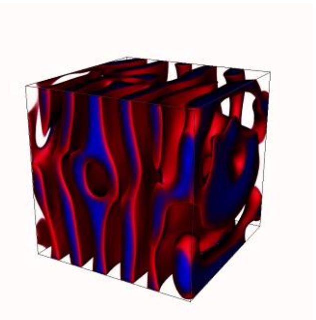
The University of Manchester

MANCHESTER 1824

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

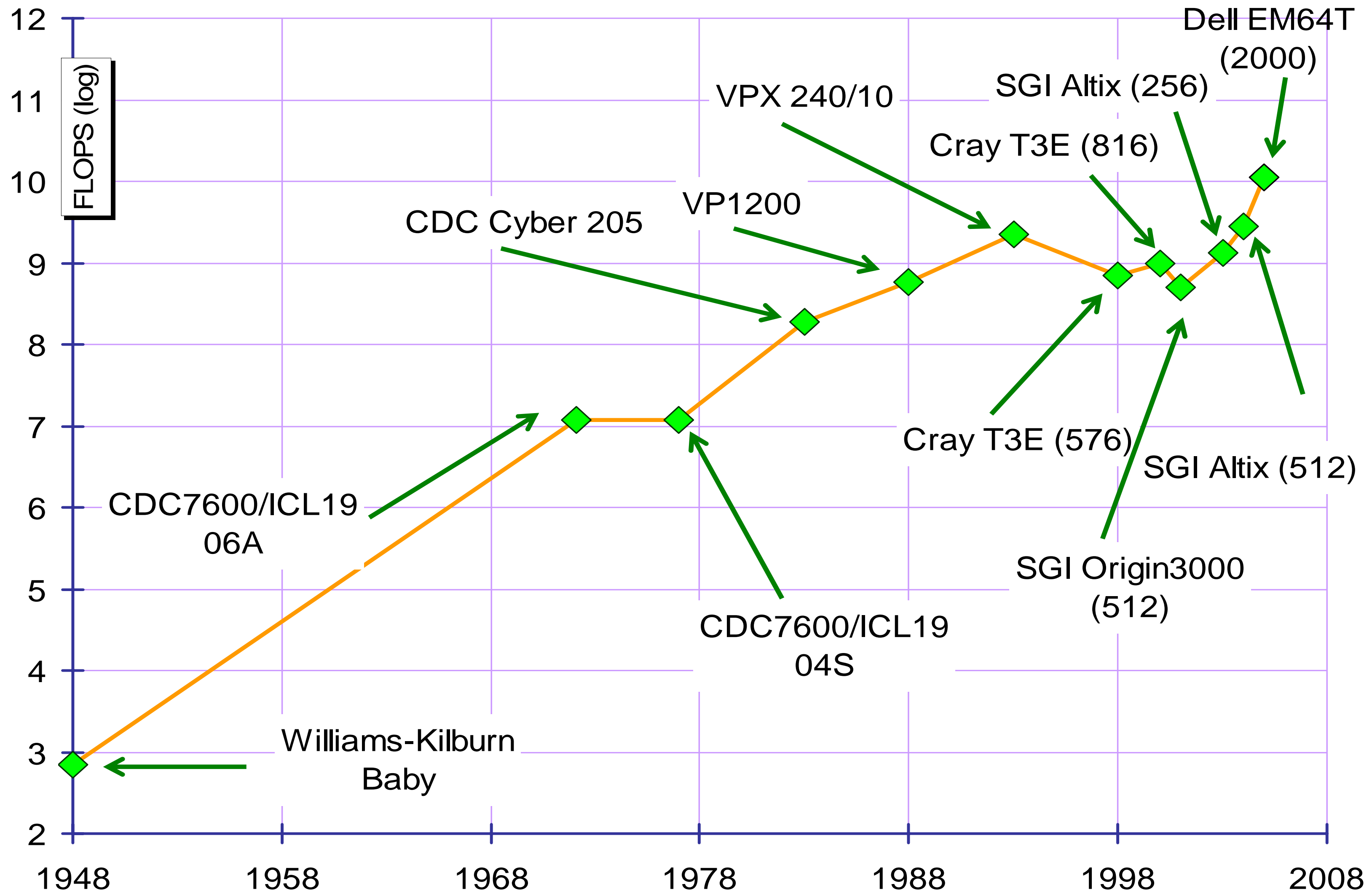




MANCHESTER 1824

The University of Manchester

High Performance Computing services since 1948



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.

- 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

MANCHESTER
1824

The University
of Manchester

The Tool

Combining the strengths of UMIST and
The Victoria University of Manchester

Requirements

- 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

MANCHESTER 1824

The University of Manchester

Reading In Data

- Data comes in from system profiler
- Best option is text based profiles – PAT
- Data is incorporated into internal data structure

```
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_
```

The University of Manchester

MANCHESTER 1824

Data Views

- **Renameable tabbed windows for each run**
- **Function list and summary info**
- **Expandable functions giving information for each profile**
- **Derived statistic information**

profview

File Experiments Tools

1 | 3 | 4 | 5 | 6 | 7 | 8

Function Name	Inc. Secs	Exc. Secs	Inc. Percentage	Exc. Percentage	File	Inc. Diff	Exc. Diff
global_sum_n	6.562	0.000	3.675	0.000	misc_parallel.c,	8.910	0.000
gradcalc	3.220	2.430	1.767	1.367	mean_viscous.c,	6.120	4.680
[illegible]alo_cells	13.670	3.350	7.633	1.867	mean_update.c,	8.190	6.240
halo_vector	7.840	0.215	4.383	0.133	misc_parallel.c,	23.160	0.720
imp_flux2	1.170	0.970	0.667	0.533	mean_osher.c,	0.870	0.600
imp_flux3	1.330	0.910	0.733	0.533	mean_osher.c,	3.030	2.160
inner_product_list_d	0.326	0.037	0.175	0.013	mean_linear.c,	0.240	0.240
inner_product_list_d2	6.570	0.007	3.675	0.000	mean_linear.c,	8.850	0.060
inner_product_list_d2	0.180	0.060	0.100	0.000	mean_linear.c,		
inner_product_list_d2	3.870	0.000	2.200	0.000	mean_linear.c,		
inner_product_list_d2	4.650	0.000	2.600	0.000	mean_linear.c,		
inner_product_list_d2	8.490	0.000	4.700	0.000	mean_linear.c,		
inner_product_list_d2	8.700	0.000	4.900	0.000	mean_linear.c,		
inner_product_list_d2	8.760	0.000	4.900	0.000	mean_linear.c,		
inner_product_list_d2	8.880	0.000	5.000	0.000	mean_linear.c,		
inner_product_list_d2	9.030	0.000	5.000	0.000	mean_linear.c,		
inner_product_list_ls	35.846	0.056	20.038	0.038	2eq_linear.c,	44.670	0.270
inner_product_list_s	97.976	0.086	54.737	0.038	mean_linear.c,	118.500	0.600
loads	0.060	0.004	0.013	0.000	io_input_output.c,	0.120	0.030
local_time_step	1.360	1.090	0.767	0.600	mean_update.c,	2.940	2.490
main	179.085	0.000	99.975	0.000	main.c,	1.410	0.000
mapping	4.600	4.600	2.533	2.533	mesh_mapping.c,	8.160	8.160
memcpy	0.520	0.520	0.300	0.300	bcopy.s,	0.330	0.330
memset	0.070	0.070	0.033	0.033	bzero.s,	0.090	0.090
mmapping	0.195	0.195	0.100	0.100	mesh_extract_datastructure.c,	0.030	0.030
move_mesh	1.796	0.000	0.988	0.000	mean_unsteady.c,	1.290	0.000
mpi_init	0.030	0.000	0.000	0.000	misc_parallel.c,	0.000	0.000

This is my parallel prof tool

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.

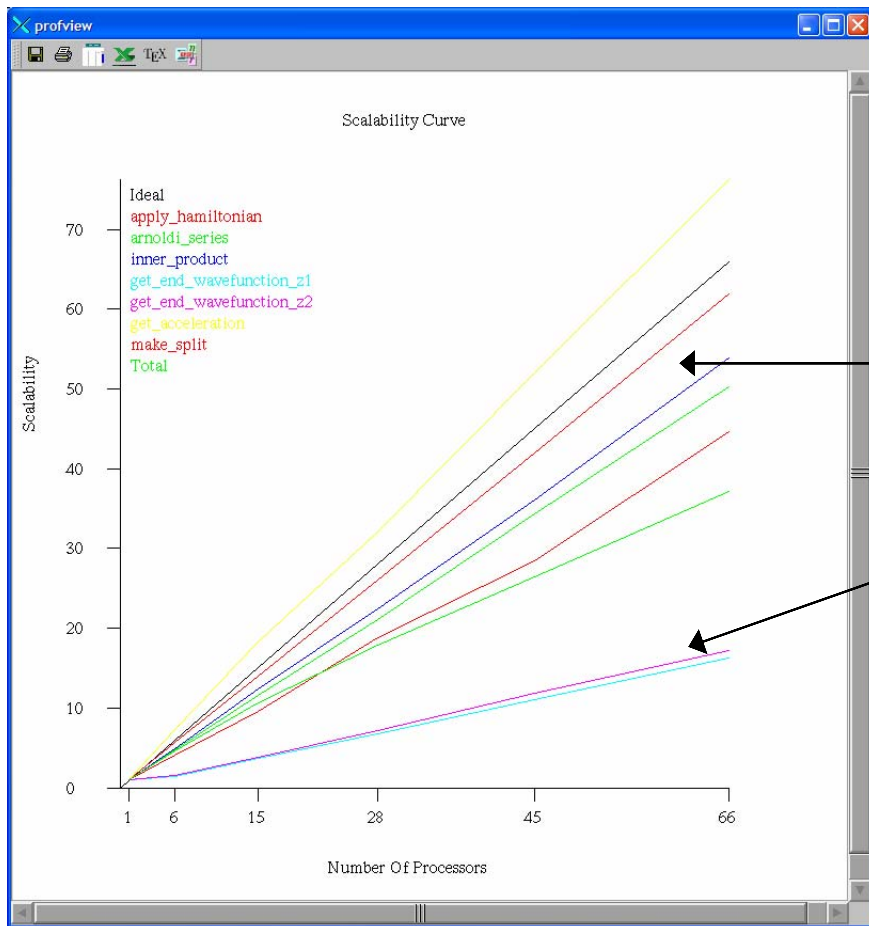
- We don’t see any more information here than profiles provide but accessing the information is quicker.

Generating Graphs

- 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



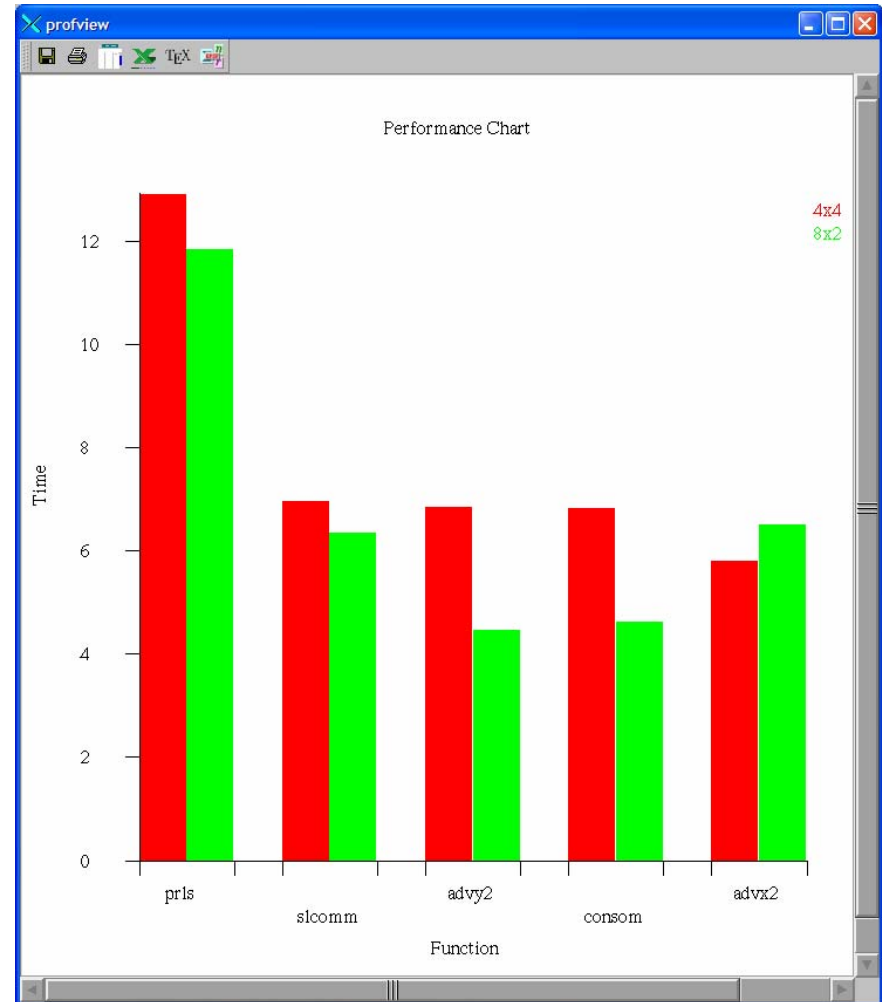
- Data output icons (later)
- Scalability curve
- Legend
- Scalable routines
- Non scaling routines
- From here we see two small routines (on lower processor counts) will dominate at even higher processor counts.

- 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 Plot

- 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

MANCHESTER
1824

The University of Manchester

The Summary

Combining the strengths of UMIST and
The Victoria University of Manchester

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.

Manchester Computing

Combining the strengths of UMIST and
The Victoria University of Manchester