



# Single Node Performance Analysis of Applications on HPCx

J. Mark Bull

*Edinburgh Parallel Computing Centre, University of Edinburgh, JCMB, The King's Buildings, Mayfield Road, Edinburgh, EH9 3JZ, UK.*

*email: [m.bull@epcc.ed.ac.uk](mailto:m.bull@epcc.ed.ac.uk)*

## Abstract

This report analyses the performance of a range of application codes on an IBM p575, which forms a single node of the Phase 3 HPCx system. We find that most codes run at between 8% and 20% of the nominal peak floating point performance of the system. A small number of codes, which heavily utilize tuned libraries, run at between 20% and 50% of peak. For each code we also collected and analysed a range of other performance metrics derived from hardware counters on the Power5 processor. We also investigate the performance impact of enabling simultaneous multithreading SMT: the performance gain varied from a 29% slowdown to a 44% speedup. We gain some interesting insights into the performance of the set of codes, but also expose some of the shortcomings of this approach.

## 1. Introduction

In this report we analyse the measured performance of a set of 19 application codes on a single node of the HPCx Phase 3 installation. We concentrate primarily on the floating point performance of these codes, and how this relates to the peak floating point performance of the hardware. In addition to measuring floating point performance, we also use the hardware counter facilities of the Power5 processors to record a range of other metrics for each code. We then attempt to gain some further insight into the performance characteristics of the codes by analyzing this additional data.

In this study, we used parallel versions of all the codes, and collected data (with a few exceptions, see later) from runs using all 16 processors of a p575 shared memory node. We consider this preferable to running sequential codes for the following reason: a single CPU of a p575 node has access to more than  $1/16^{\text{th}}$  of the Level 2 and 3 cache memory and more than  $1/16^{\text{th}}$  of the memory bandwidth. The vast majority of production work on the system is carried out with parallel codes where all 16 processors of each node are utilized. Sequential codes running in isolation can therefore achieve inflated performance which is unrepresentative of the way the system is used in practice.

We also investigate the effectiveness of simultaneous multithreading (SMT) on the applications. Each application was also run using double the number of processes, but on the same number of processors with SMT enabled. We are then able to use other metrics to gain some insight into why some applications benefit from SMT and others do not.

The remainder of this report is organized as follows: Section 2 lists the application and benchmark codes which are used in this study, and Section 3 gives an overview of the p575 architecture. In Section 4 we present the results of the analysis, and in Section 5 draw conclusions and suggest some further work which might be carried out.

## **2. Applications and Benchmarks**

In this section we give a brief description of the 19 application and benchmark codes used in this study.

### **AIMPRO**

The AIMPRO code is used to calculate the structural, electrical, optical and mechanical properties of molecules and solids using spin-polarised self-consistent local density functional pseudopotential techniques. Calculations can be carried out on very large atomic systems containing up to 800 atoms. Applications range from studying macro-molecules and nanocrystals to the simulation of bulk materials.

### **CASTEP**

The materials science code CASTEP performs quantum mechanical (density functional) calculations of bulk materials, surfaces and interacting molecules. The code expands electronic wave-functions in plane waves with many associated transformations between real and reciprocal space during a direct minimization of the electronic energy. The code uses three-dimensional Fast Fourier Transforms (FFTs) on a distributed grid, performed using MPI\_AllToAllV combined with serial FFTs. These electronic calculations occur in all the various scientific applications of the code.

### **CENTORI**

CENTORI is a global fluid code developed by UKAEA Culham used to model turbulence evolution and transport in fusion reactors. These models represent a tokamak plasma using eight or nine 3D fields varying in space and time. Non-linear balance equations of particle number, momentum and energy for each species and a reduced set of Maxwell equations are derived by suitable averaging of the more exact kinetic description.

### **CRYSTAL**

The CRYSTAL code uses a periodic Hartree-Fock or density functional Kohn-Sham Hamiltonian and various hybrid approximations in the calculation of wave-functions and properties of crystalline systems. The wave-functions are expanded in atom centred Gaussian type orbitals (GTOs) providing a highly efficient and numerically precise solution with no shape approximation to the density or potential. The code is developed within a long-standing collaboration between the Theoretical Chemistry Group at the University of Torino, Italy, and the Computational Materials Science Group at Daresbury Laboratory.

## **DL\_POLY**

DL\_POLY 3 is a general-purpose molecular dynamics simulation package designed to cater for a wide range of possible scientific applications and computer platforms, especially parallel hardware. It supports a wide range of application areas, including ionic solids, solutions, metals, zeolites, surfaces and interfaces, complex systems (e.g. liquid crystals), minerals, bio-systems, and those in spectroscopy.

## **GAMESS-UK**

The GAMESS-UK package is a general-purpose molecular quantum chemistry code. It originated from the GAMESS code developed by Michel Dupuis et al., and has been extended over the past two decades by contributions from the CCP1 community, and from a number of Theoretical Chemistry Groups, including those at Utrecht University (NL), at the University of Wuppertal (BRD), and at Humboldt University in Berlin (BRD).

## **H2MOL**

H2MOL is a code from the Multi-photon and Electron Collision Consortium (CCP2) written by Ken Taylor & Daniel Dundas, Queens University Belfast. It solves the time-dependent Schrödinger equation to produce estimates of energy distributions for laser-driven dissociative ionization of the H<sub>2</sub> molecule. A cylindrical computational grid is defined with Phi, Rho and Z co-ordinates, with the Z domain distributed amongst an array of processors arranged logically in a triangular grid (to take advantage of symmetry). The Z variables are modelled using a finite-difference mesh whose points are equally spaced.

## **LAMMPS**

LAMMPS is a molecular dynamics package which solves classical physics equations and is able to simulate a wide range of materials including atomic, molecular, metallic and hybrid systems.

## **LUDWIG**

LUDWIG is a general-purpose parallel Lattice-Boltzmann code, capable of simulating the hydrodynamics of complex fluids in 3-D. LUDWIG is capable of handling multicomponent fluids, amphiphilic systems, and flow in porous media as well as colloidal particles and polymers.

## **MDCASK**

MDCASK is a molecular dynamics code which was originally developed to study radiation damage in metals. It operates by calculating the energies of and forces on, and determining the motions of, the atoms in the system which is characterised by specific interatomic potentials and boundary conditions.

## **NAMD**

NAMD is a parallel, object-oriented molecular dynamics code designed for high-performance simulation of large biomolecular systems. The code has been developed

by the Theoretical and Computational Biophysics Group, in the Beckman Institute for Advanced Science and Technology at the University of Illinois at Urbana-Champaign.

## **NEWT**

NEWT is a finite volume, unstructured mesh code for the solution of CFD problems associated with flow in turbomachinery. The parallel version PNEWT was used for this study.

## **PCHAN**

The UK Turbulence Consortium (UKTC) is at the forefront of simulating turbulent flows by Direct Numerical Simulation (DNS), a direct first principles solution of the Navier-Stokes equations which is become more and more feasible for realistic turbulent flows as the capabilities of computer systems increase. UKTC has developed a parallel code to solve problems associated with shock/boundary-layer interaction. The SBLI code is a sophisticated DNS code that incorporates a number of advanced features: namely high-order central differencing; a shock-preserving advection scheme from the total variation diminishing (TVD) family; entropy splitting of the Euler terms and the stable boundary scheme. The PCHAN benchmark is a simple turbulent channel flow benchmark using the SBLI code.

## **PMEMD**

AMBER (Assisted Model Building Energy Refinement) is a parallel molecular dynamics suite designed specifically for bio-molecular systems. The code has been developed by a collaboration between The Scripps Research Institute, the University of Utah, NIEHS, Penn State and SUNY-Stony Brook. PMEMD is one of the key computational codes from the AMBER suite.

## **POLCOMS**

The Proudman Oceanographic Laboratory Coastal Ocean Modeling System (POLCOMS) has been developed to tackle multi-disciplinary studies in coastal/shelf environments. The central core is a sophisticated 3-dimensional hydrodynamic model that provides realistic physical forcing to interact with, and transport, environmental parameters. The hydrodynamic model is a 4-dimensional finite difference model based on a latitude-longitude Arakawa B-grid in the horizontal and S-coordinates in the vertical. Conservative monotonic PPM advection routines are used to ensure strong frontal gradients. Vertical mixing is through turbulence closure (Mellor-Yamada level 2.5).

## **PRMAT**

The parallel R-matrix program PRMAT has been optimised to treat open d-shell atoms and ions as well as intermediate energy scattering problems. Several new facilities have been incorporated into PRMAT to improve its efficiency for these calculations. In electron-ion scattering calculations it is necessary to perform separate computations at a very large number of scattering energies in order to compute the thermally averaged collision strengths required in applications. The propagation stage of the calculation is the most computationally intensive part of the calculation, due to the tens of thousands of scattering energies that are often required to yield accurate

results. Around 2/3 of the processors form processor pipelines to propagate the R-matrices, and therefore take advantage of the near-optimal performance of serial BLAS 3 routines (ESSL). This operation is the computational rate-determining step. The remaining processors are assigned to the asymptotic region calculation, based on the final R-matrices emerging from the processor pipelines.

## **SANDER**

SANDER is another of the key computational codes from the AMBER suite (see PMEMD above). It predates PMEMD, and uses a less efficient algorithm, but offers more functionality.

## **SIESTA**

Siesta (Spanish Initiative for Electronic Simulations with Thousands of Atoms) performs electronic structure calculations and *ab initio* molecular dynamics simulations of molecules and solids. It uses the standard Kohn-Sham self-consistent density functional method.

## **VASP**

VASP (Vienna Ab-Initio Simulation Package) is a package for performing ab-initio quantum-mechanical molecular dynamics (MD) using pseudopotentials and a plane wave basis set. The approach implemented in VASP is based on a finite-temperature local-density approximation (with the free energy as variational quantity) and an exact evaluation of the instantaneous electronic ground state at each MD-step using efficient matrix diagonalization schemes and an efficient Pulay mixing.

# **3. Hardware**

## **IBM p575**

Phase 3 of the HPCx system contains 160 compute nodes, each of which is an IBM p575 containing 16 processors. The p575 compute nodes utilise IBM Power5 processors. The Power5 is a 64-bit RISC processor implementing the PowerPC instruction set architecture. It has a 1.5 GHz clock rate, and has a 5-way super-scalar architecture with a 20 cycle pipeline. There are two floating point multiply-add units each of which can deliver one result per clock cycle, giving a theoretical peak performance of 6.0 Gflop/s. There is one divide and one square root unit, which are not pipelined.

The processor has 120 integer and 120 floating-point registers. There is extensive hardware support for branch prediction, and both out-of-order and speculative execution of instructions. There is a hardware prefetch facility: loads to successive cache lines trigger prefetching into the level 1 cache. Up to 8 prefetch streams can be active concurrently.

The level 1 cache is split into a 32 Kbyte data cache and a 64 Kbyte instruction cache. The level 1 data cache has 128-byte lines, is 2-way set associative and write-through. Each chip contains two processors with their own level 1 caches and a shared level 2 cache. The level 2 cache is a 1.9 Mbyte combined data and instruction cache, with 128 byte lines and is 10-way set associative and write-back.

The Power5 processors support *simultaneous multithreading* (SMT). With SMT, each processor can support two instruction streams, allowing the simultaneous execution of two threads. These streams appear as logical processors, two per physical processor, four per chip. However, these logical processors share the physical processor's level 1 instruction and data cache, floating-point, and other functional units. Enabling SMT will also cause the level 2 and 3 caches to be shared among the four logical processors.

Each chip is packaged, together with a level 3 cache, into a Dual-Core Module (DCM). The level 3 cache is 36 MBytes and is shared between the 2 processors, equivalent to 18 Mbytes per processor. It has 256 byte lines, and is 12-way set associative and write-back.

Each eServer node contains 8 DCMs (16 processors) and has 32 Gbytes of main memory. Inter node communication is provided by an IBM's High Performance Switch (HPS). Each eServer node (frame) has two network adapters and there are two links per adapter, making a total of four links between each of the frames and the switch network.

## hpmcount

`hpmcount` is a utility which gives access to the hardware counters on the IBM Power5 processor. There are six hardware counter registers on the processor: two of which always count the same events: processors run cycles and completed instructions. The other four registers can count a wide variety of possible events, many of which are not especially useful for performance analysis. In this study we used the AIX version of `hpmcount`, which only allows access to eight groups of events. The ACTC version, which was not available at the time, but which is now installed on HPCx, gives access to 148 event groups.

## 4. Results

### Methodology

Each benchmark or application was run on 16 processors (with a few exceptions, see below). For each code, five separate runs were made, instrumented with `hpmcount` and recording events from counter groups 1, 2, 3, 4, and 8. (**Note:** For reasons related to symmetry in the problem geometry, H2MOL can only execute on processor numbers of the form  $n(n+1)/2$ . Therefore instead of 16 processors, runs were made on 15 processors. In addition, the NEWT and PRMAT runs were made on 32 processors (2 nodes) and the AIMPRO runs on 64 processors (4 nodes), due to restriction on problem size and geometry. In these case, the data was scaled to single node appropriately. ) Table 1 shows the raw data which we obtained for each application from `hpmcount`.

For each code, a dataset and/or input parameters were chosen to yield a problem size which is small enough to fit in the 27Gb of available memory on the node, but large enough to be representative of real usage of the application codes, and large enough so that communication overheads do not dominate the execution time.

For each application (except GAMESS-UK, which failed to execute correctly), we performed a second set of runs where the number of processes (i.e. MPI tasks) was doubled (or increased to 28 in the case of H2MOL) and SMT enabled, with the number of physical processors used remaining the same.

Wall clock time	Wall clock time in secs
Time in user mode	Time in user mode in secs
Time in system mode	Time in system mode in secs
Maximum resident set size	Maximum resident set size in Mbytes
PM_FPU_FIN	FPU produced result
PM_FPU_1FLOP	FPU executed 1-flop instruction
PM_CYC	Processor cycles
PM_FPU_STF	FPU executed floating point store instruction
PM_LSU_LDF	LSU executed floating point load instruction
PM_INST_DISP	Instructions dispatched
PM_INST_CMPL	Instructions completed
PM_LD_MISS_L1	L1 cache load misses
PM_ST_MISS_L1	L1 cache store misses
PM_DTLB_MISS	Data TLB misses
PM_ST_REF_L1	L1 cache store references
PM_LD_REF_L1	L1 cache load references
PM_DATA_FROM_L3	Data loaded from L3
PM_DATA_FROM_LMEM	Data loaded from local memory
PM_DATA_FROM_L2MISS	Data loaded missed L2
PM_DATA_FROM_RMEM	Data loaded from remote memory

Table 1: Raw data obtained from `hpmcount`

## Floating point performance

To derive the floating point operation (flop) rate from the raw data, we must recognise that the `PM_FPU_FIN` event includes floating point stores, but only counts fused multiply-adds (FMAs) as a single operation. Non-FMA flops are counted by `PM_FPU_1FLOP`, and floating point stores by `PM_FPU_STF`, so the actual flop count is given by the expression

$$2 * (PM\_FPU\_FIN - PM\_FPU\_STF) - PM\_FPU\_1FLOP$$

For each code, the total flop rate for all 16 processors was calculated, and expressed as a percentage of the peak flop rate for the p575, which is 96 Gflop/s. This data is shown in Figure 1.

The highest performance (just over 50% of peak) is obtained by AIMPRO and the lowest, 1.5% of peak, by NEWT. A small group of codes achieve between 20% and 50% of peak performance. The codes in this group typically make extensive use of tuned libraries (either dense linear algebra or FFTs) for a significant portion of the computation. Most other codes achieve between 8% and 20% of peak performance.

It must be noted that the percentage of peak performance is not necessarily a good measure of the quality of an application, and should not be interpreted as such. Flop rates are a poor measure the quality of science achieved, and can vary significantly within a single application depending on the input data and chosen parameters. Furthermore, some types of application are intrinsically harder for modern microprocessors to achieve high flop rates on than others.

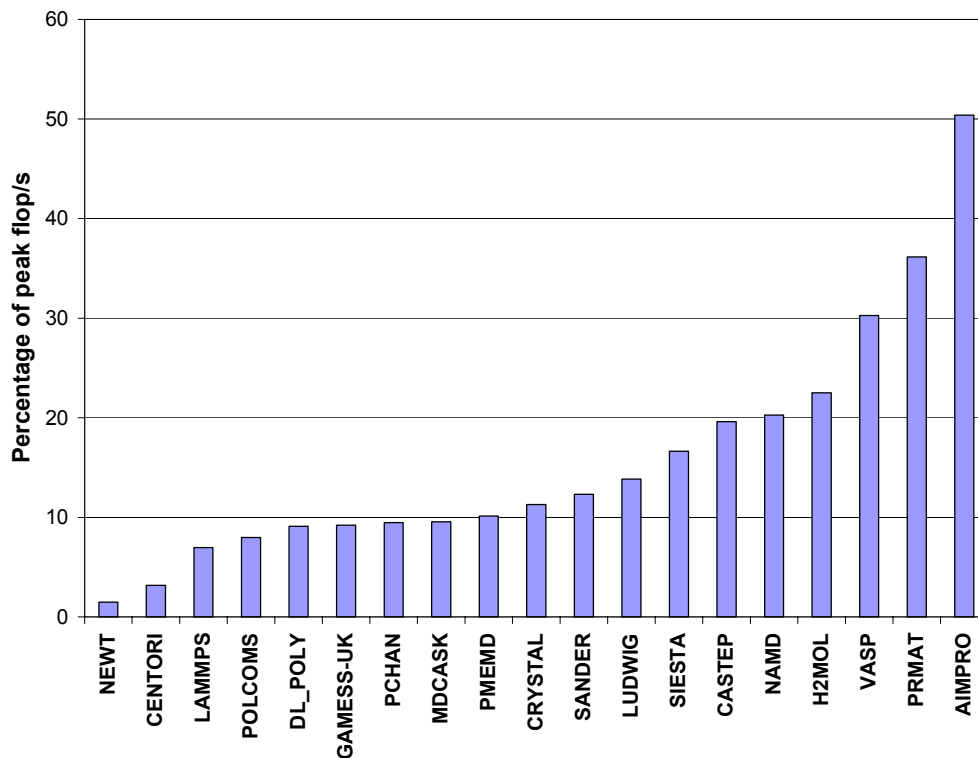


Figure 1: Percentage of peak performance achieved

## Other metrics

In addition to the flop rate, a number of other metrics can be derived from the raw hardware event counts. These include:

- **Floating point multiply-add percentage** The percentage of flops which are executed in fused multiply-add instructions.
- **Flops per floating point load/store (Computational intensity)** The ratio of floating point operations to the total number of floating point load and store instructions.
- **Floating point load/stores per cycle** The average number of floating point load and store instructions per cycle
- **Level 1 cache references per nanosecond** The rate at which load and store instructions are issued.
- **Level 1 cache hit ratio** The fraction of accesses to Level 1 cache resulting in a hit.
- **Level 1 cache hits per nanosecond** The rate at which Level 1 cache hits occur.
- **Level 2 cache hit ratio** The fraction of accesses to Level 1 cache resulting in a hit.
- **Level 2 cache hits per microsecond** The rate at which Level 2 cache hits occur.

- **Level 3 cache hit ratio** The fraction of accesses to Level 3 cache resulting in a hit.
- **Level 3 cache hits per microsecond** The rate at which Level 3 cache hits occur.
- **Level 3 cache misses per microsecond** The rate at which Level 3 cache misses occur.
- **Main memory accesses per microsecond.** The rate at which main memory accesses (local and remote) occur.
- **TLB misses per microsecond** The rate at which TLB misses occur.
- **Instructions per nanosecond** The rate of all instructions issued.
- **Percentage of instructions which are floating point** The ratio of floating point instructions to all instructions, expressed as a percentage .
- **Percentage of load/stores which are floating point** The ratio of floating point load stores to all load/stores, expressed as a percentage .

In order to assess the importance of these metrics in determining the flop rate, we computed the correlation coefficient between each metric and the flop rate across the 19 codes. These correlation coefficients are shown in Table 1, together with the minimum, mean and maximum values of the metrics across the 19 codes. For this size of dataset (19 variables, 17 degrees of freedom), a correlation coefficient with absolute value of greater than 0.456 is significant at the 95% level, and a correlation coefficient with absolute value of greater than 0.575 is significant at the 99% level.

The metrics with the highest correlation to flop rate are **flops per floating point load/store** (called *computational intensity* in `hpmcount` output) and **floating point load stores per cycle**. This is not surprising, since the flop rate can be written as the product of **flops per floating point load/store** with **floating point load/store per cycle**. Figure 4 shows how these two metrics combine to give the resulting flop rates.

Metric	Min	Mean	Max	Correlation
Flops per FP load/store (CI)	0.72	2.00	4.18	0.88
Floating point load/stores per cycle	0.08	0.30	0.49	0.79
Instructions per nanosecond	0.71	1.74	2.68	0.77
Percentage of instructions which are FP	21.61	54.75	88.42	0.75
Percentage of load/stores which are FP	33.48	63.98	91.60	0.66
Percentage of flops in FMAs	17.47	73.04	99.69	0.66
Level 1 cache hits per nanosecond	0.23	0.61	0.93	0.65
Level 1 cache references per nanosecond	0.28	0.66	0.95	0.61
Level 1 cache hit ratio	81.15	90.99	98.00	0.44
TLB misses per microsecond	0.03	0.27	0.80	0.04
Level 3 cache misses per microsecond	0.01	0.42	3.11	-0.02
Memory accesses per microsecond	0.004	0.39	3.11	-0.03
Level 3 cache hits per microsecond	0.09	1.24	4.44	-0.09
Level 2 cache hit ratio	91.27	96.65	99.19	-0.11
Level 3 cache hit ratio	8.72	78.23	98.36	-0.12
Level 2 cache hits per microsecond	17.81	53.99	131.17	-0.13

Table 2: Minimum, average and maximum values of performance metrics, and correlation coefficients of performance metrics with flop rate.

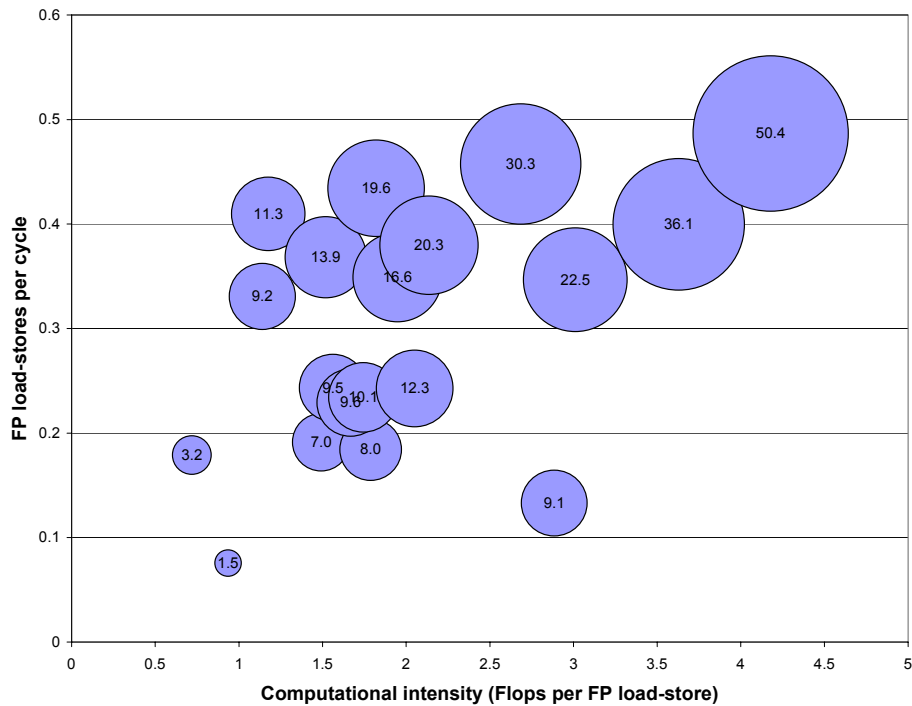


Figure 2. Scatter plot of floating point load/stores per cycle against computational intensity. The size of each circle and the number it contains is the percentage of peak flop rate.

Codes which obtain a high percentage of the peak flop rate have a high computational intensity (between 2.5 and 4) and also perform a high number of floating point load/stores per cycle (between 0.3 and 0.5). Many of the other codes have a computational intensity of between 1 and 2.5, and these appear to fall into two groups, one with a high number of floating point load/stores per cycle (between 0.3 and 0.5) and one with a lower number of floating point load/stores per cycle (between 0.15 and 0.3). Two codes (NEWT and CENTORI) have a computational intensity of less than 1. DL\_POLY is alone in having a high computational intensity, but a low floating point load/store rate.

Despite the superscalar properties of the Power5 processors, it is also clear that to achieve a high percentage of the peak flop rate, the application must not be doing too much non-floating point work. There are significant correlations between the percentage of peak flop rate and both the **percentage of instructions which are floating point** and **percentage of load/stores which are floating point**. Figure 3 shows a scatter plot of percentage of peak flop rate against the percentage of load/stores which are floating point. Codes with low values of both these metrics are likely to be doing significant numbers of address manipulations, either via indirect array indexing or pointer chasing.

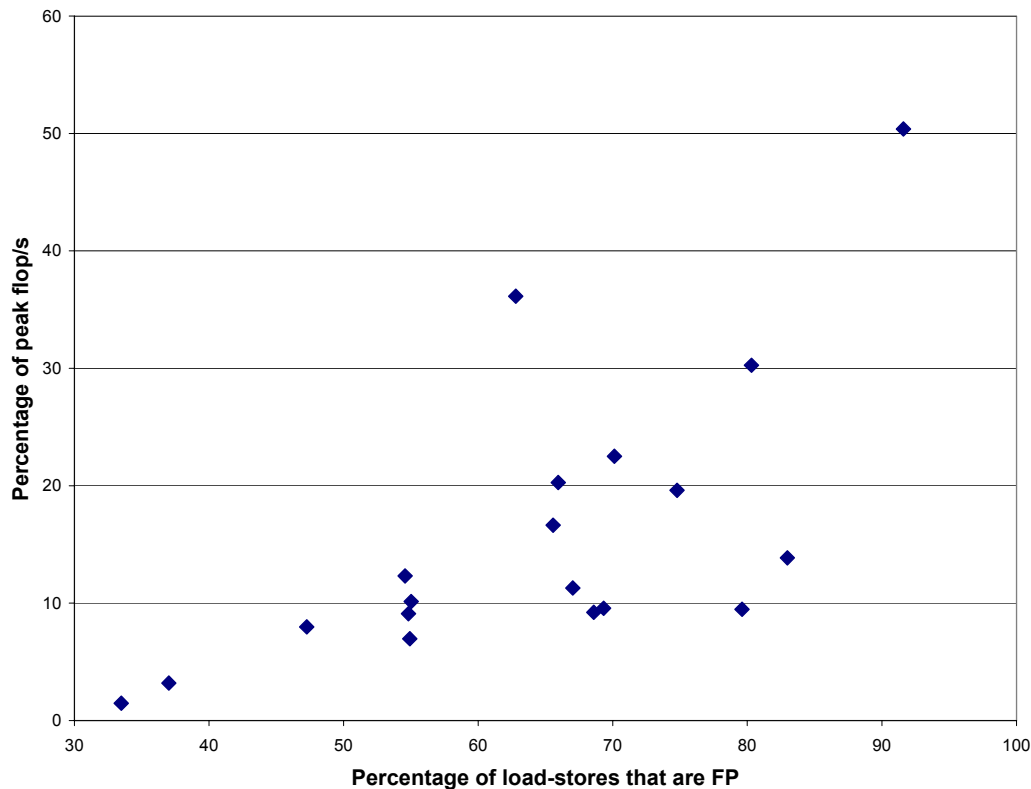


Figure 3. Scatter plot of percentage of peak flop/s against percentage of load/stores that are floating point.

Since the peak performance of the Power5 processor gains a factor of two from the presence of fused multiply-add instructions, the percentage of flops computed in such instructions is also a factor in determining whether an application attains high flop rates. Figure 4 shows a scatter plot of percentage of peak flop rate against the percentage of floating point instructions in FMAs. This suggests that, on the whole the compiler, is good at generating FMA instructions: with one exception (MDCASK) application do more than 50% of their flops in FMAs. Codes with high floating point rates typically execute more than 90% of their flops in FMAs.

It is perhaps surprising that none of the metrics related to the memory system shows a significant correlation with percentage of peak flop rate. The Level 1 cache hit ratio is almost significant at the 95% level: Figure 5 shows a scatter plot of percentage of peak flop rate against the Level 1 cache hit ratio. While the application with low flop rates do have low Level 1 cache hit ratios, it is hard to discern other patterns in this data.

A possible explanation for the lack of other significant correlations is as follows: cache hit ratios are only important if the number of cache hits is large: some applications have working sets which are cache resident. Furthermore, with hardware prefetching, the number of cache hits and misses at a given level is not necessarily a good measure of the length of time the CPU will stall for the data.

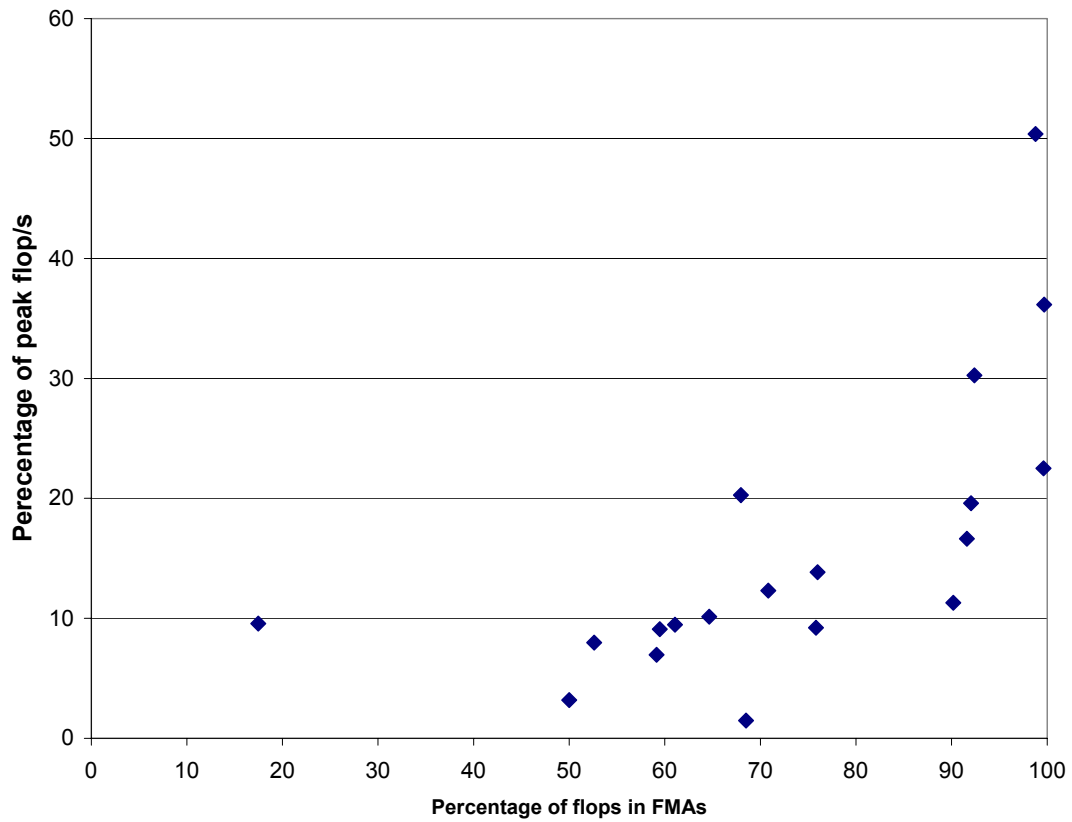


Figure 4. Scatter plot of percentage of peak flop/s against percentage of flops in FMAs

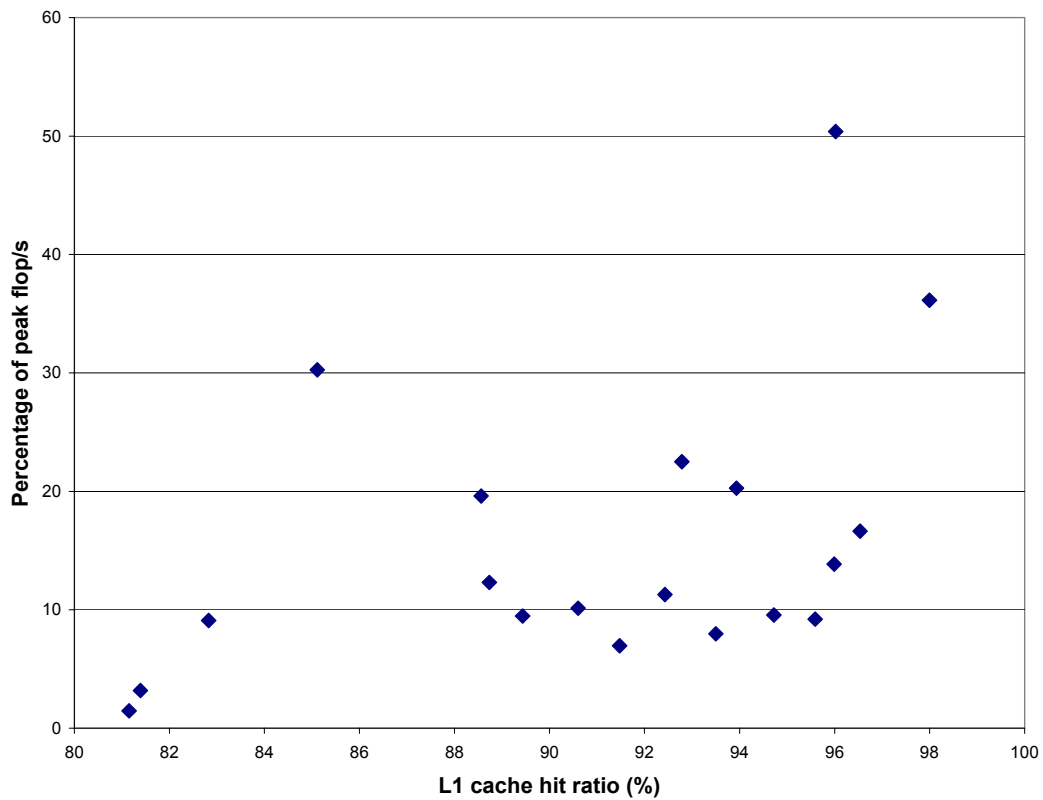


Figure 5. Scatter plot of percentage of peak flop/s against Level 1 cache hit ratio

Note that we have not used bandwidth as a metric: the reason for this is that the Power5 hardware counters do not supply enough information to determine the actual consumed bandwidths in the memory system. The reason for this is the presence of hardware prefetching. For example, a Level 2 hit event will be recorded if either the data was already resident in Level 2 cache, or if it was resident lower in the memory hierarchy and was successfully prefetched to Level 2 before the matching load/store operation. Thus a Level 2 hit event may or may not imply that memory bandwidth between Level 2 and Level 3 (or possibly main memory) has been consumed. There is also no way to discover whether prefetched data is used or not.

### Simultaneous multithreading

For each application, we compared the total execution time for the runs with and without SMT enabled. Figure 6 shows the speedup for each application as a result of enabling SMT: a number less than one indicated that the code ran slower with SMT than without. Since the benchmark cases were chosen such that communication is not a significant overhead, we can be reasonably confident that the increased communication overheads resulting from doubling the number of processes are not important in determining the outcome.

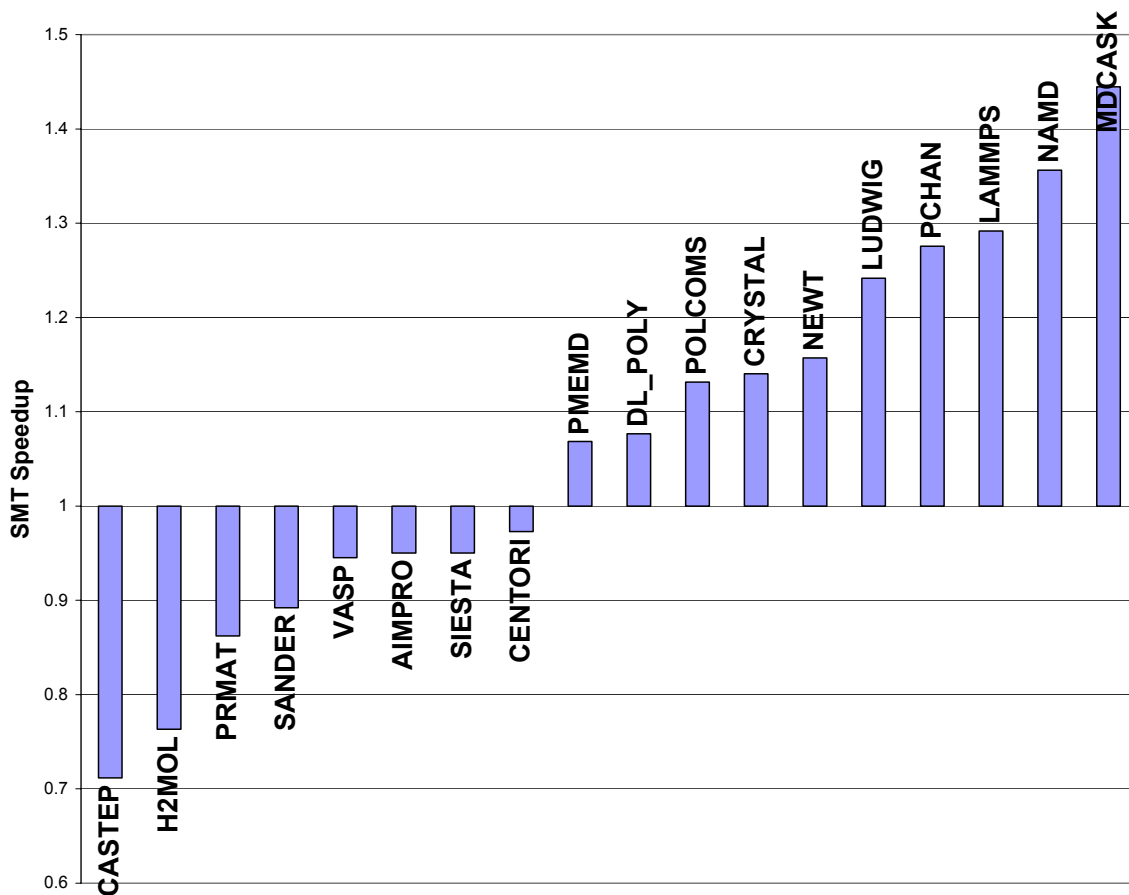


Figure 6: Speedup resulting from enabling SMT

The results show a range of outcomes, from a 27% slowdown for CASTEP to a 44% speedup for MDCASK. The geometric mean speedup is 1.06 and the geometric mean speedup of application which do benefit from SMT is 1.22.

We also computed correlations between SMT speedup and the other metrics for the applications. Few of these are significant, but those that are give us some insight. The most significant correlations are with the floating point multiply-add percentage (-0.70), the percentage of peak flop rate (-0.45), computational intensity (-0.44) and the Level 1 cache references per nanosecond (-0.43). Figures 7, 8 and 9 show the scatter plots of SMT speedup against floating point multiply-add percentage, the percentage of peak flop rate, and the Level 1 cache references per nanosecond respectively.

It is clear that applications with a high FMA percentage, a high peak flop rate or a high number of L1 references per nanosecond, typically do not benefit from SMT. The likely explanation is that these applications are already close to saturating CPU resources: either the floating point units and/or the bandwidth to L1 cache. Doubling the number of threads causes contention for these resources and the resulting slowdown in performance. Only applications where there are spare resources benefit from the introduction of the extra threads, which can potentially utilize some of the spare capacity.

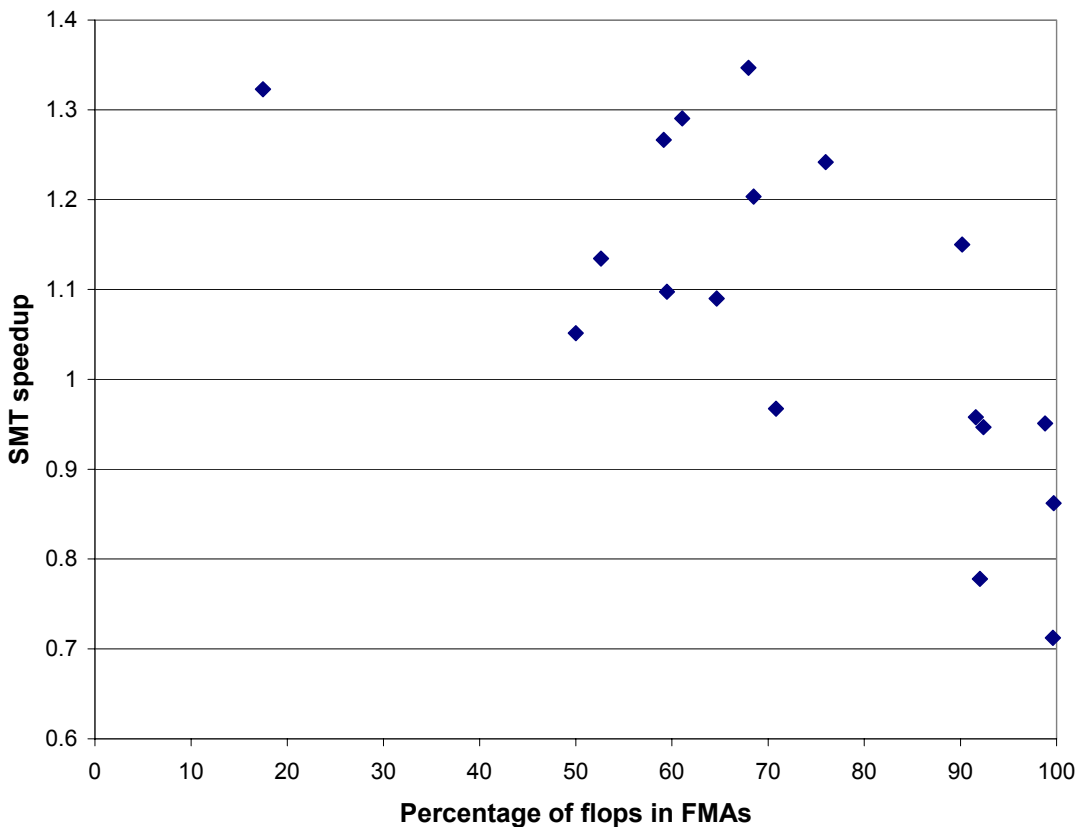


Figure 7. Scatter plot of SMT speedup against percentage of flops in FMAs

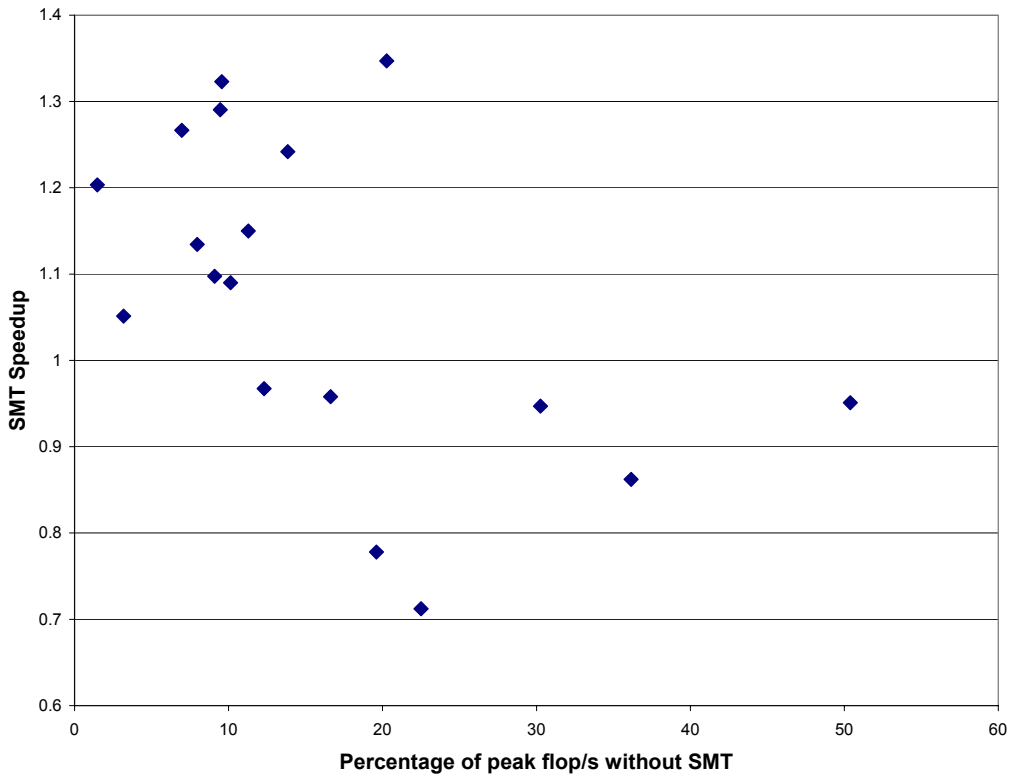


Figure 8. Scatter plot of SMT speedup against percentage of peak flop/s

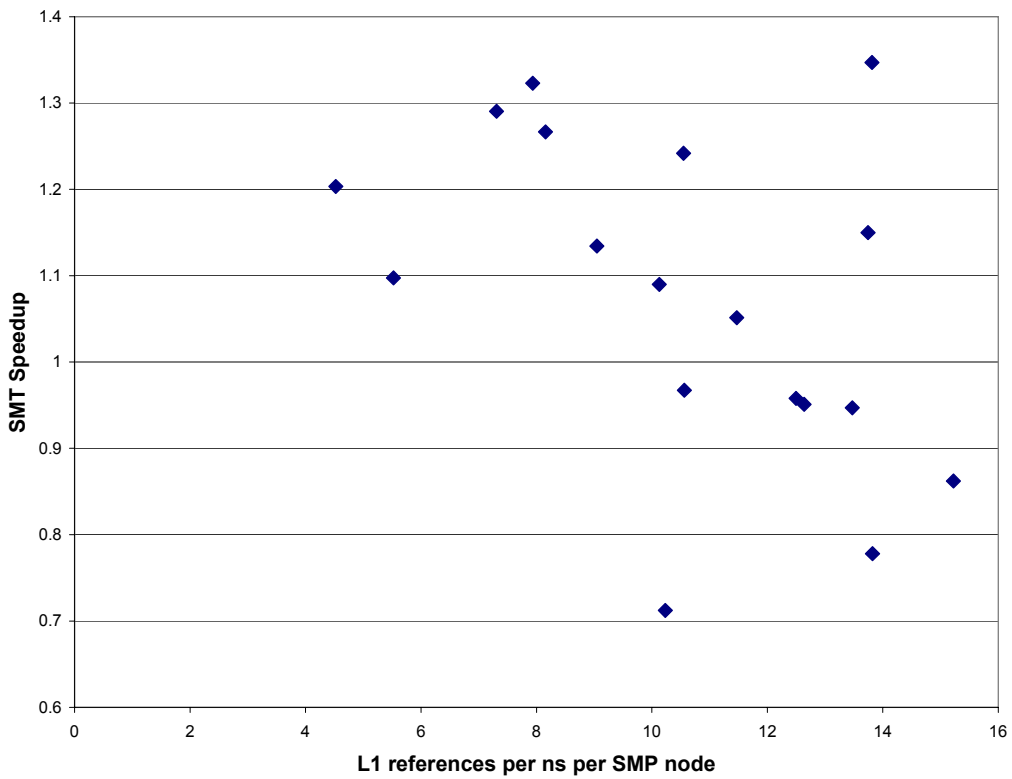


Figure 9. Scatter plot of SMT speedup against Level 1 cache references per nanosecond

## 5. Conclusions and Future Work

We have reported the single node performance for a range of application and benchmark codes on HPCx. Typically, these codes achieve between 8% and 20% of the nominal peak performance of the system. However, a small number of codes that rely heavily on linear algebra or FFT libraries achieve up to 50% of peak performance.

We also measure a number of other metrics for these codes, derived from hardware performance counter data, which give some interesting insights into the performance of codes on this system. By correlating each metric with the flop rate, we can determine to some extent which factors are most important in determining performance.

However, it is clear that it is not straightforward to explain observed performance using these metrics. In particular, the complexities of the memory system, including hardware prefetching, make the interpretation of this data difficult. A useful performance metric available on SPARC processors, for example, is the time the processor spends waiting for memory accesses to be resolved. Unfortunately there is no way to derive this metric using the Power5 hardware counters.

We have investigated the use of the simultaneous multithreading capability of the Power5 processors and its impact on performance. Again, by correlating the speedup due to SMT with other metrics we can observe some of the factors which influence whether SMT is beneficial or not.

A potential use for data such as that collected in this study could be in the design of benchmark suites for a given user community. It would be possible to ensure that a small set of applications were representative of the application mix in the community by ensuring that they gave reasonable coverage of a number of different metrics.

## Acknowledgements

The author gratefully acknowledges the assistance of the following people in collecting data for this study: Martin Plummer, Ian Bush, Andy Sunderland, and Mike Ashworth of CCLRC Daresbury Laboratory; Alan Gray, Joachim Hein, Jon Hill, Kenton D'Mellow, Fiona Reid, Lorna Smith and Kevin Stratford of EPCC, The University of Edinburgh.