Investigation of a set of real applications using Power5 hardware counters

Mark Bull

EPCC, University of Edinburgh

markb@epcc.ed.ac.uk

Overview

- Applications
- Power5 hardware counters
- Results
- Conclusions

Acknowledgements

- Many thanks to the following HPCx staff for collecting data for this study:

  Mike Ashworth, Ian Bush, Alan Gray, Joachim Hein, Jon Hill, Kenton D'Mellow, Martin Plummer, Fiona Reid, Lorna Smith, Kevin Stratford and Andy Sunderland

Disclaimer

- This is NOT an application beauty contest!
- Just because one application scores higher on some metric than another does not make it better.
- Flop rates are not a useful measure of achieved science.
- Some types of application are intrinsically harder for modern microprocessors than others.

Applications

- We studied a set of 19 applications, all of which consume cycles on HPCx, the UK national supercomputing resource.
  - limited set of application domains, broadly reflecting the usage of the service.

- **Quantum Chemistry**: AIMPRO, CASTEP, CRYSTAL, GAMESS-UK, SIESTA, VASP
- **Molecular Dynamics**: DL_POLY, LAMMPS, MDCASK, NAMD, AMBER-PMEMD, AMBER-SANDER
- **Computational Fluid Dynamics**: NEWT, PCHAN, LUDWIG
- **Atomic Physics**: H2MOL, PRMAT
- **Plasma Physics**: CENTORI
- **Ocean Modeling**: POLCOMS

Hardware

- The current HPCx service consists of a cluster of 160 IBM eServer 575 nodes.
- Each node has 16 1.5GHz Power5 processors and 32GB main memory.
  - CPU supports 2-way simultaneous multi-threading (SMT)
- Cache hierarchy:
  - 32KB L1 per processor
  - 1.9MB L2 per pair of processors
  - 36MB L3 per pair of processors
- Interconnect: IBM HPS

Power5 hardware counters

- The Power5 CPU has six hardware counter registers
- Two of these are reserved: they always count processor cycles and completed instructions
- Other four counters can count many other events
- We used the AIX version of **`hpmcount`** to access the hardware counters
  - only gives access to a small, but useful, subset of events
  - ACTC version of **`hpmcount`** gives access to many more events, but was not available on our system at the time. Most of these are not particularly interesting....

Experiments

- Each application was run on a single node using 16 processes
  - except for a few which were run on 15, 32 and 64 for reasons of problem size/geometry
  - hardware counter data for these was scaled accordingly
- For each application five runs were executed, using counter groups 1,2,3,4, and 8
- For each application, we repeated the above runs, using double the number of processes, and with SMT enabled
  - same number of CPUs
- Following raw event counts recorded:

Raw data

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

Derived metrics

- From this raw data we can derive an interesting and useful set of metrics, including these:

Flop rate

% of peak flop rate

Floating point load/stores per cycle

Flops per load/store (computational intensity)

Level 1 cache references per nanosecond

Percentage of load/stores which are FP

Level 1 cache hit ratio

Level 1 cache hits per nanosecond

Level 2 cache hit ratio

Level 2 cache hits per nanosecond

Level 3 cache hit ratio

Level 3 cache hits per nanosecond

Level 3 cache misses per nanosecond

Memory accesses per nanosecond

TLB misses per nanosecond

Instructions per second

% of instructions which are floating point

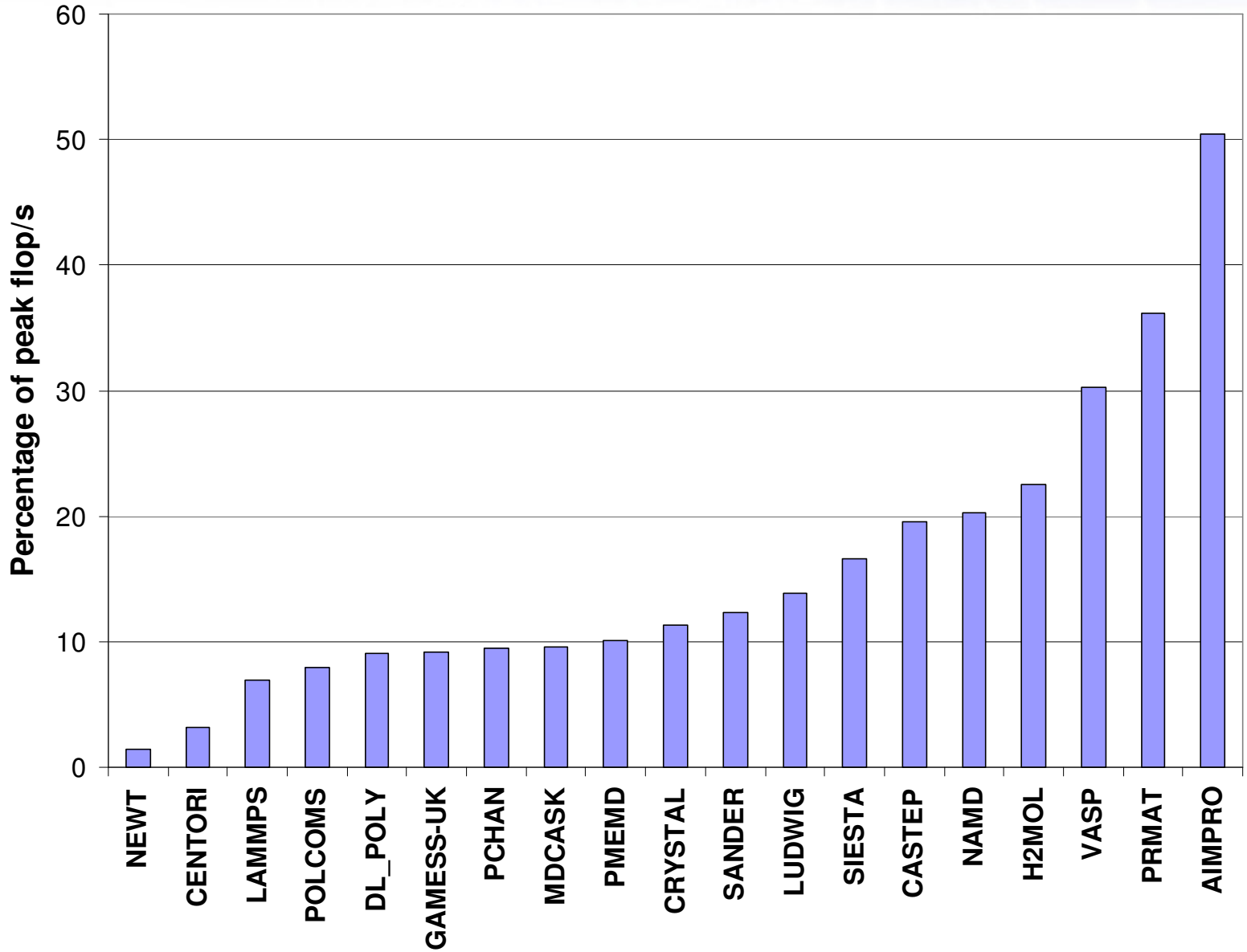
% flops in FMAs

Percentage of peak flop rate

- Minimum: 1.5%
- Maximum: 50.4%
- Mean: 15.8%

- Many applications in the 8% to 20% range
- Codes with very high flop rates are typically doing a lot of dense linear algebra or FFTs (no big surprise!)

Percentage of peak flop rate

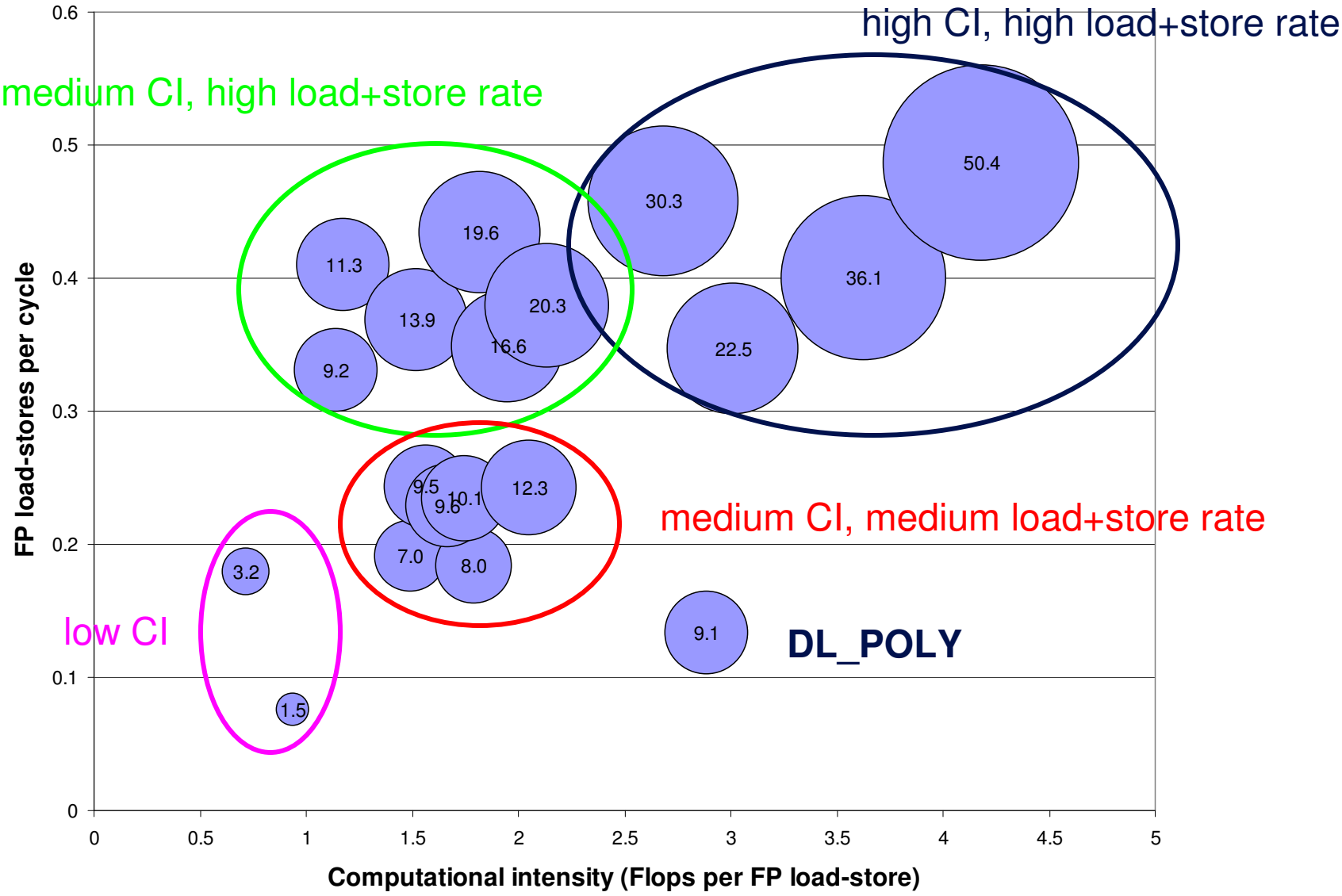


Computational intensity

- Computational intensity is the ratio of floating point operations to floating point loads+stores
- Minimum: 0.71
- Maximum: 4.18
- Mean: 2.00

- Flop rate can be expressed as the product of computational intensity and the floating point load+store rate

Computational intensity

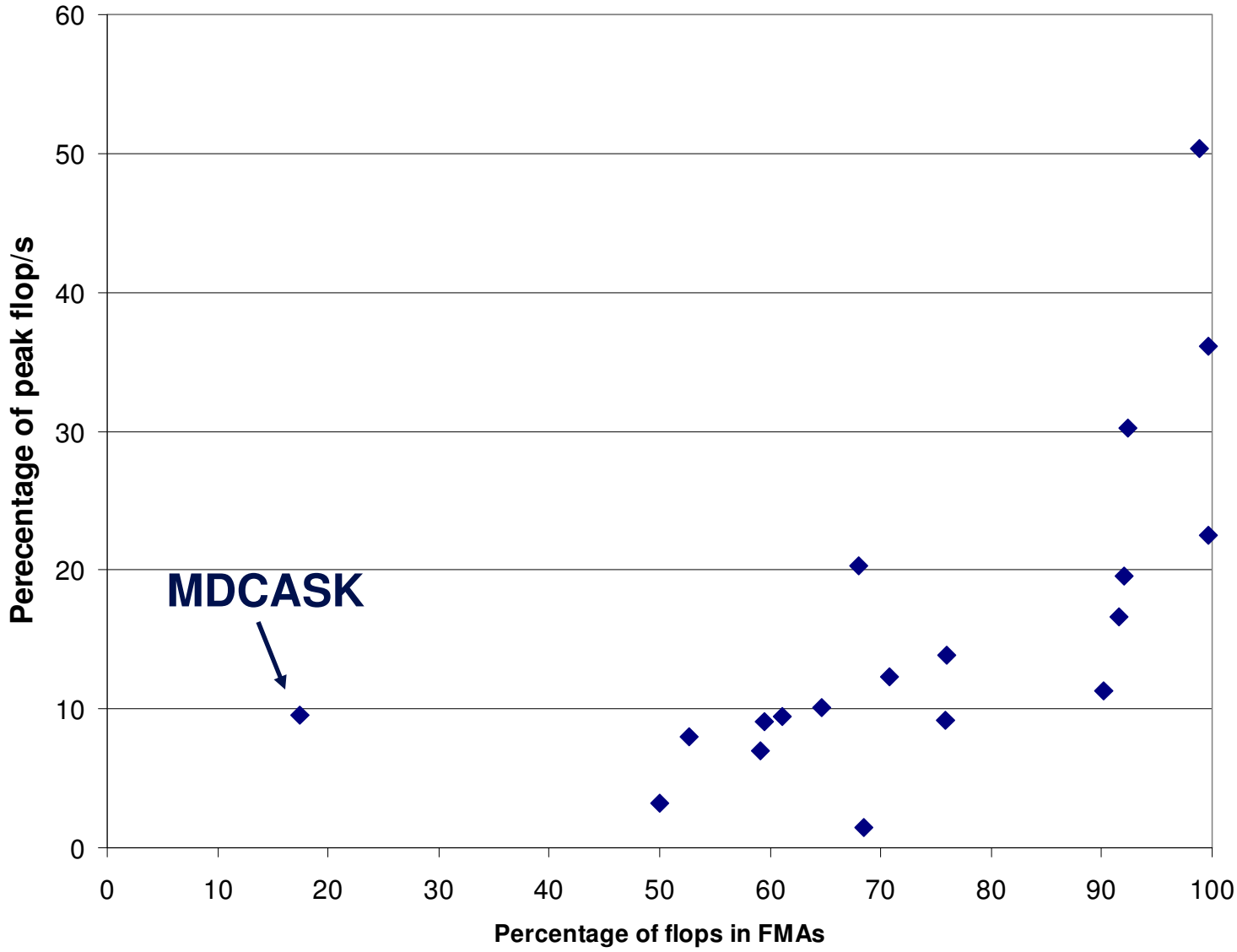


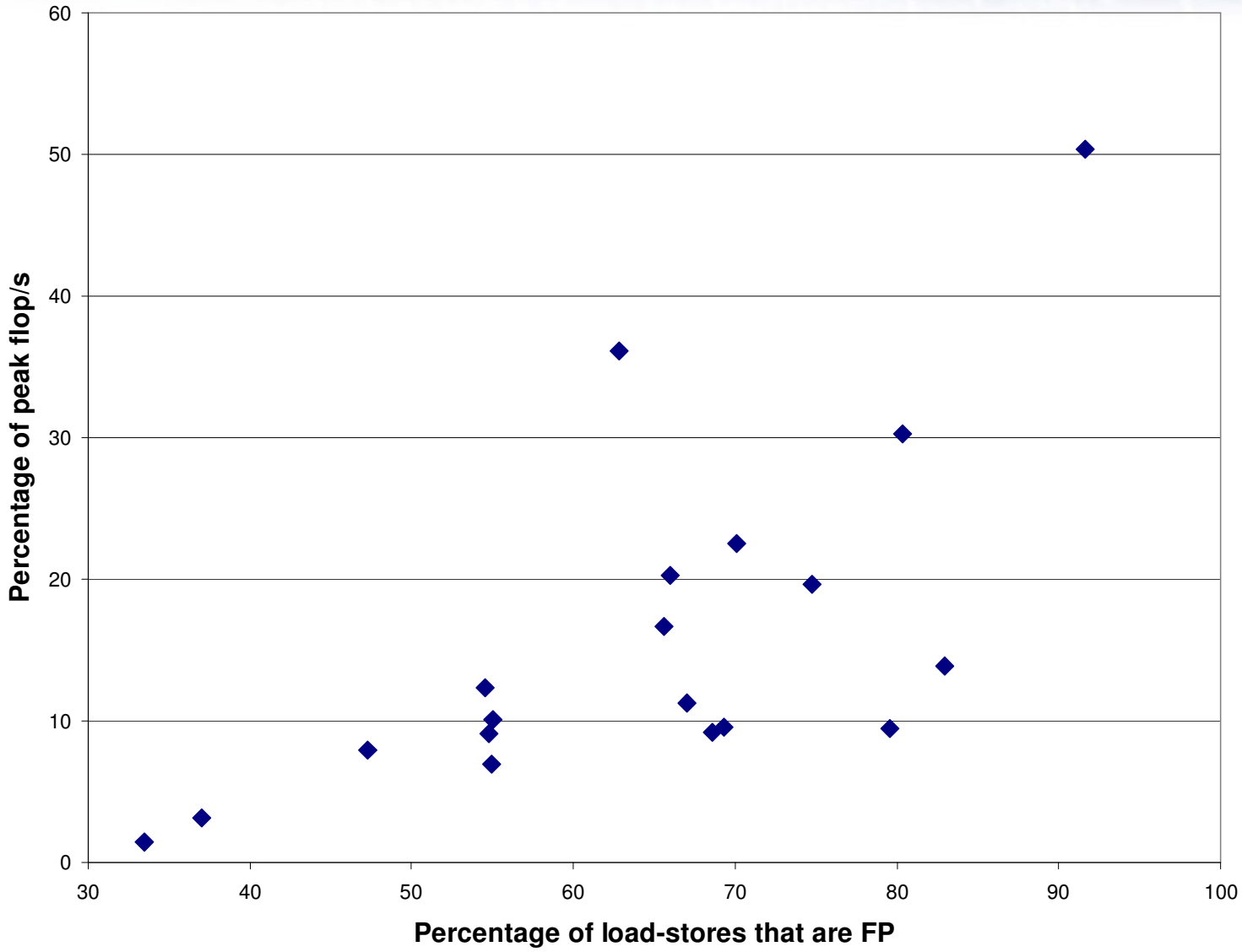
Other metrics

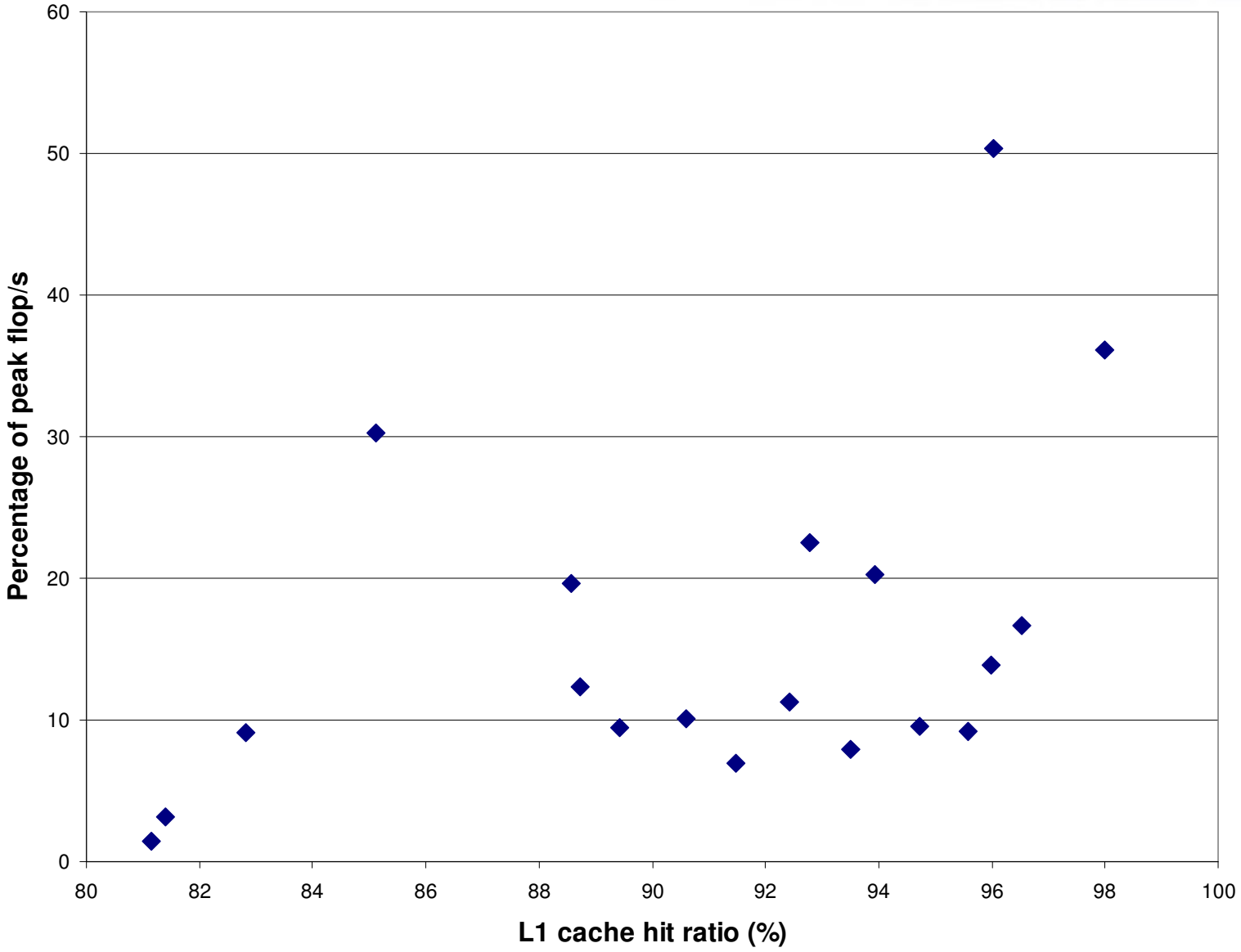
- Clearly both computational intensity and floating point load-store rate are important in determining the flop rate.
- What about other metrics?
- Can see if they are important by computing the correlation between each metric and the flop rate.
- For a sample size of 19, a correlation of more than 0.58 (or less than –0.58) is significant at the 1% level.

Other metrics

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


60
50
40
30
20
10
0
Perecentage of peak flop/s
MDCASK
0
10
20
30
40
50
60
70
80
90
100
Percentage of flops in FMAs


60
50
40
30
20
10
0
Percentage of peak flop/s
30
40
50
60
70
80
90
100
Percentage of load-stores that are FP


Percentage of peak flop/s
60
50
40
30
20
10
0
80
82
84
86
88
90
92
94
96
98
100
L1 cache hit ratio (%)

What's missing?

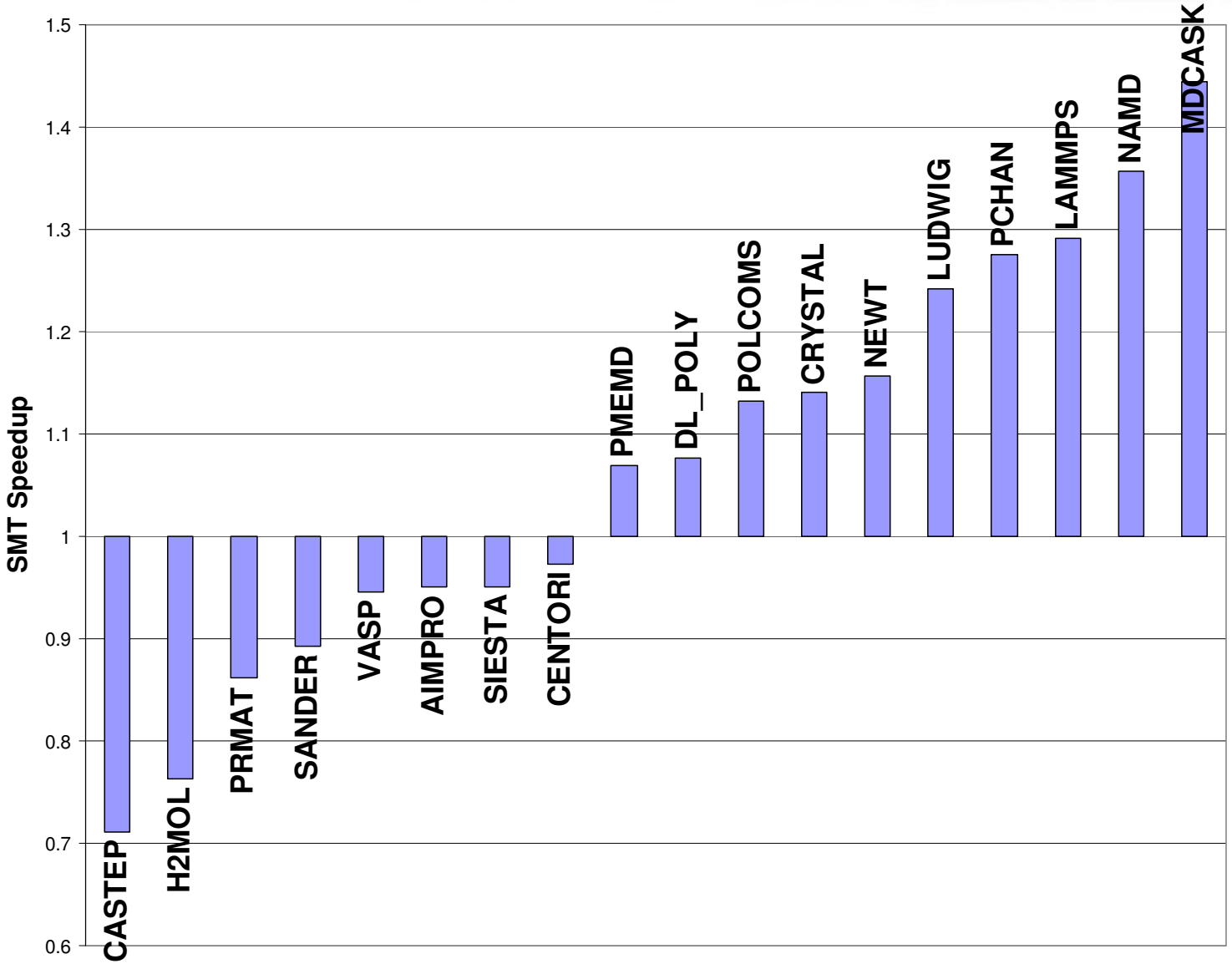
- Note there is no mention so far of bandwidth!
- **`hpmcount`** reports bandwidth from the various levels of memory hierarchy
- This is potentially very misleading, because of hardware prefetching.
  - for example, an L2 hit can occur either because the data was really in L2, or because it was just prefetched from L3 or main memory.
  - can't assume that an L2 hit has not consumed bandwidth below L2
- Power5 hardware counters can't tell us what is really going on in the memory hierarchy!
  - N.B. can count how many prefetch streams were started, but this says nothing about whether the prefetched data was used or not...

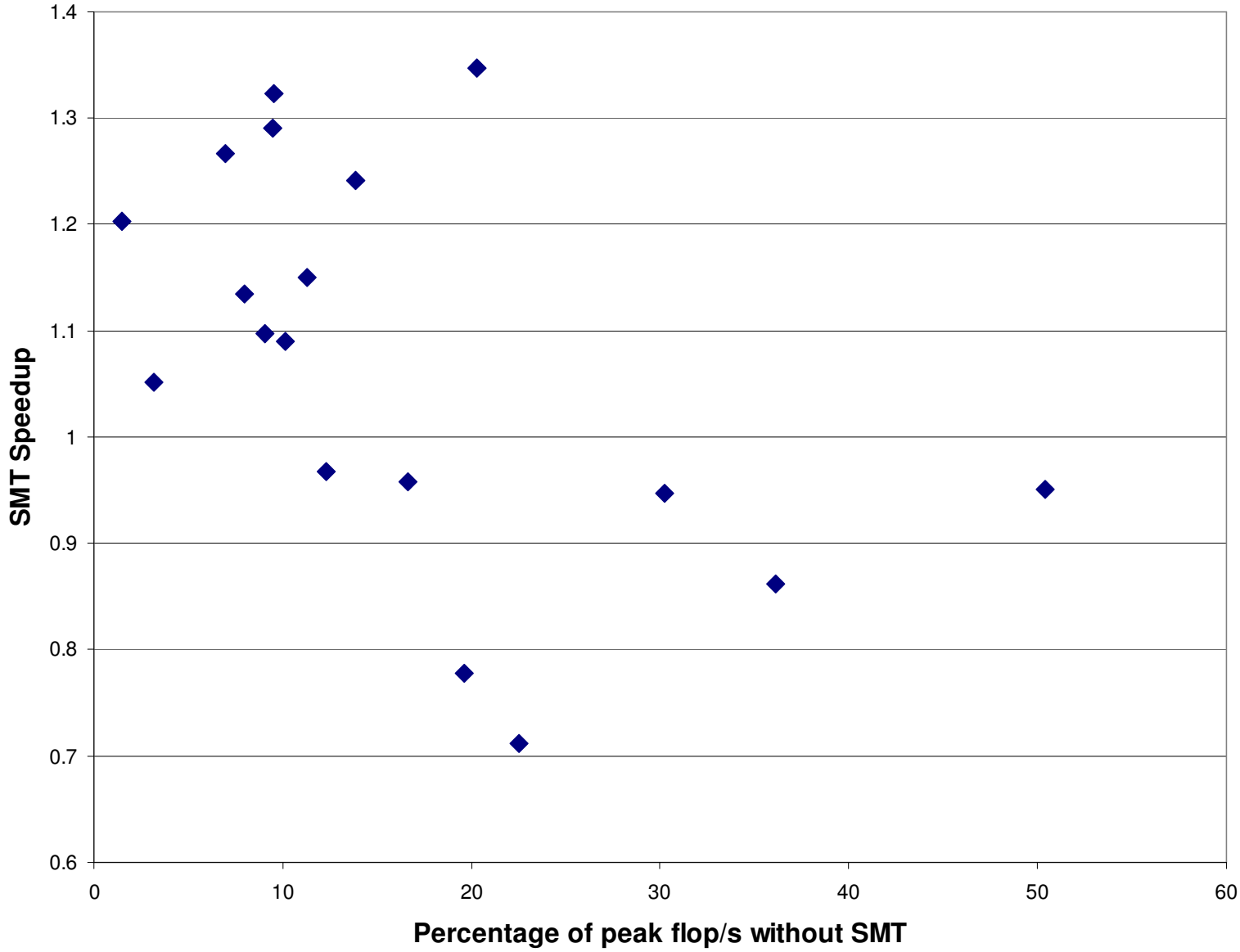
Simultaneous multithreading

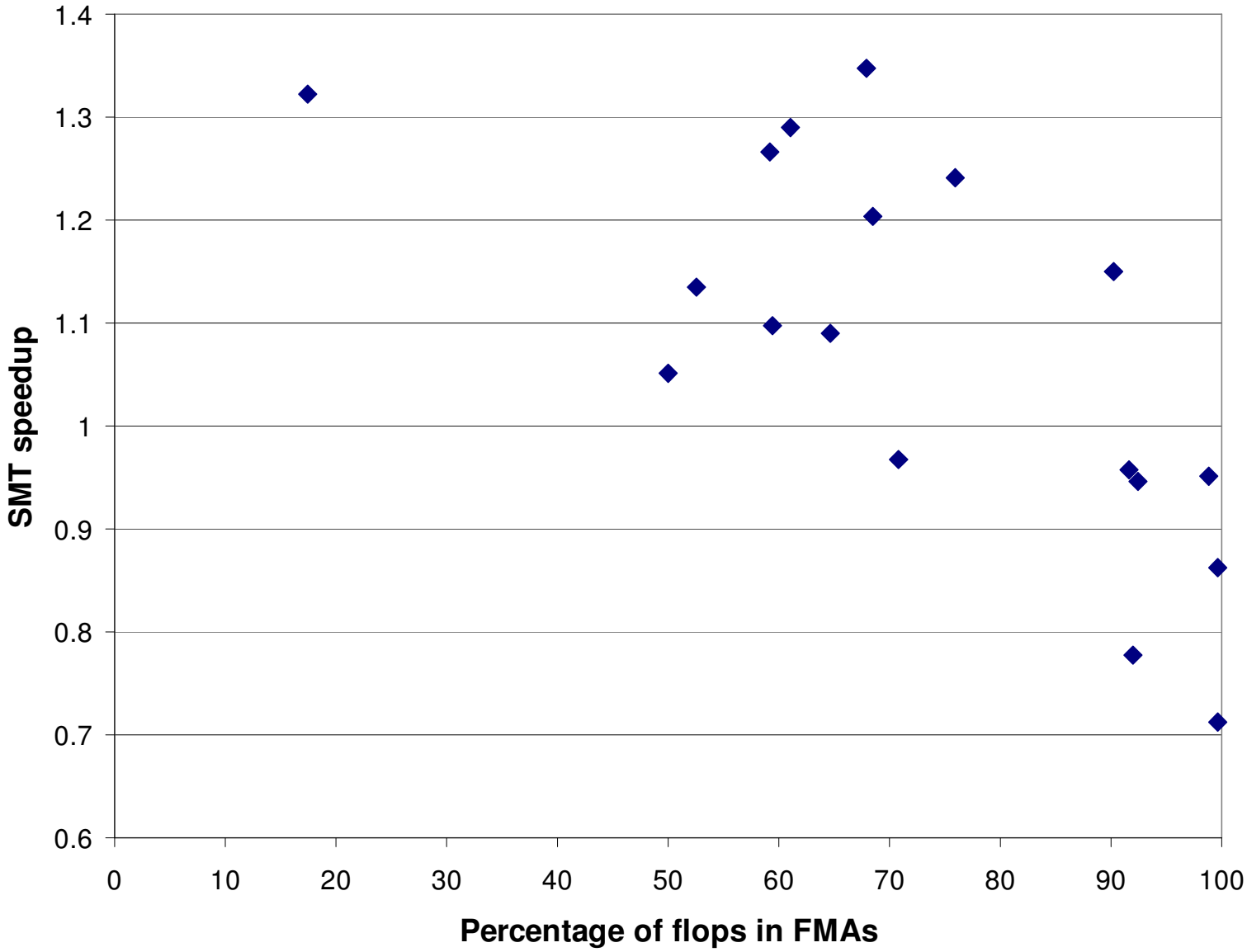
- For each application (except one) compute the speedup obtained by doubling the number of processes and enabling SMT.
- Worst: 27% slowdown
- Best: 44% speedup
- Geometric mean: 6% speedup
- Geometric mean of apps that benefit: 21% speedup

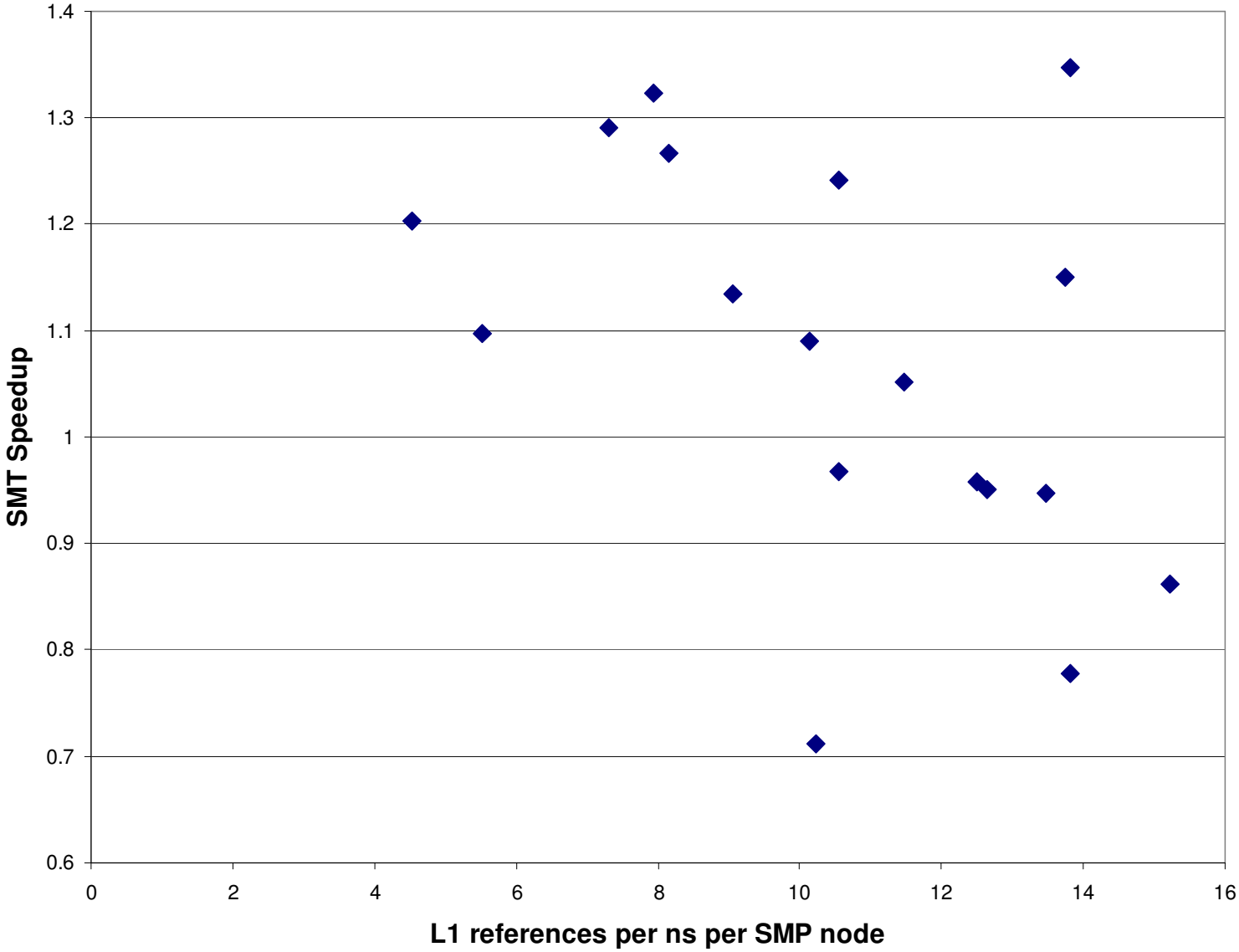
- Can look at correlation with other metrics: only a few are significant.

SMT speedup


SMT Speedup
1.5
1.4
1.3
1.2
1.1
1
0.9
0.8
0.7
0.6
CASTEP
H2MOL
PRMAT
SANDER
VASP
AIMPRO
SIESTA
CENTORI
PMEMD
DL_POLY
POLCOMS
CRYSTAL
NEWT
LUDWIG
PCHAN
LAMMPS
NAMD
MDCASK


SMT Speedup
1.4
1.3
1.2
1.1
1
0.9
0.8
0.7
0.6
0
10
20
30
40
50
60
Percentage of peak flop/s without SMT


SMT speedup
1.4
1.3
1.2
1.1
1
0.9
0.8
0.7
0.6
0
10
20
30
40
50
60
70
80
90
100
Percentage of flops in FMAs


SMT Speedup
1.4
1.3
1.2
1.1
1
0.9
0.8
0.7
0.6
0
2
4
6
8
10
12
14
16
L1 references per ns per SMP node

Conclusions

- Power5 hardware counters allow us to derive some interesting metrics
  - but important data about memory bandwidth utilisation is missing
- Applications typically can't achieve more than about 20% of peak performance without using tuned libraries
- Have identified some of the factors which contribute to high floating point performance
- Applications with high flop rates and high FMA percentages tend not to benefit from SMT
  - probably because contention for the floating point units causes pipeline stalls
  - also some evidence that applications using lots of bandwidth (at least to L1 cache) also tend not to benefit.
- This type of data could be useful in characterising an application mix and selecting a representative set of benchmarks