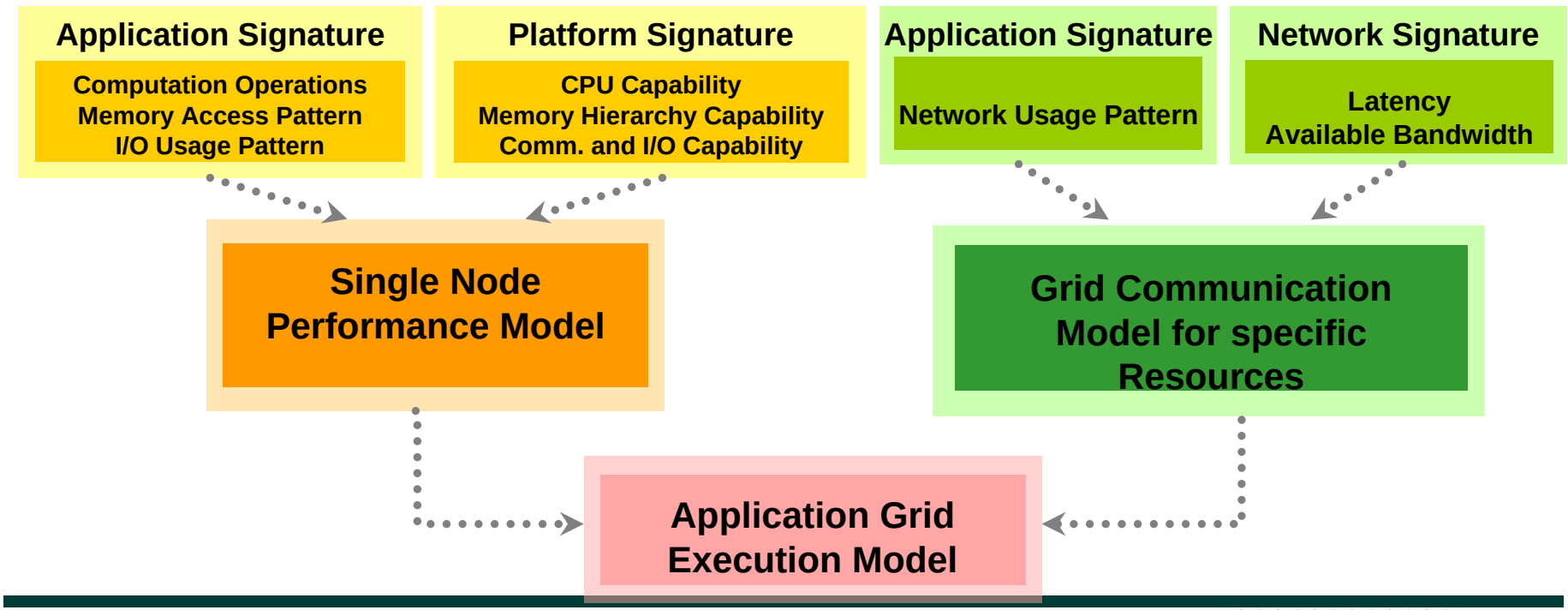
EMAN Performance Modeling How accurate are our models?

Lennart Johnsson, Bo Liu

Performance Modeling - the Big Picture

- Construct Grid Application Performance Model from
  - Single node models and Network models
  - Experience from different input data (control variables and data set sizes)



Single Node Performance Modeling

- Use hardware counters to measure the application's workload
- Use Memory Reuse Distance to capture the application's memory access pattern
- Combined with system micro-benchmark to measure unavailable hardware features, e.g. MissPenalty
- The developed models are for Intel IA64, IA32 and x86_64

$$EstimatedExeTime(psize) = k_0 \left(A + B + C + D\right) / CpuClock(arch) \quad \cdots\cdots (1)$$

$$A = k_1 \times \left(totalFp(psize) / FpPipeline\,Num(arch) \right) \times FpRepeatRate(arch) \quad \cdots\cdots (2)$$

$$B = k_2 \times L1MissCount(psize) \times L1MissPenalty(arch) \quad \cdots\cdots (3)$$

$$C = k_3 \times L2MissCount(psize) \times L2MissPenalty(arch) \quad \cdots\cdots (4)$$

$$D = k_4 \times L3MissCount(psize) \times L3MissPenalty(arch) \quad \cdots\cdots (5)$$

$$L(j)MissPenalty(arch) = L(j+1)Lantency(arch) - L(j)Latency(arch) \quad \cdots\cdots (6)$$

- Questions:
  - How accurate is the floating-point count using counters?
  - How valid is the assumption that application modeling can be made independent of architecture using our approach?
  - For a given platform how well can we predict execution time using our model? (unloaded system)

Compiler influence

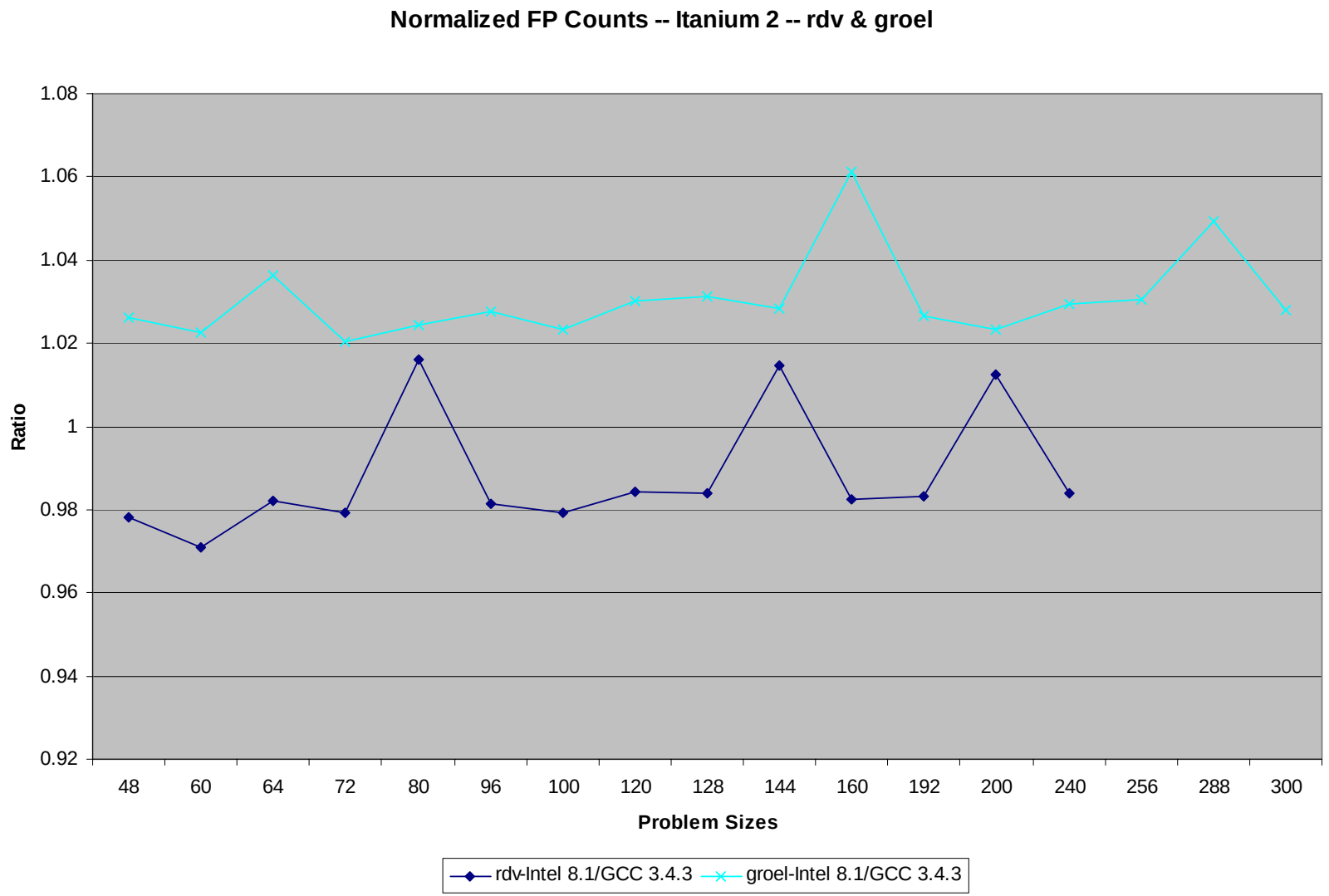
- Two EMAN data sets:

  Groel (4916 micrographs) and rdv (5613 micrographs)
- Itanium2 (900 MHz, L2D 256kB, L3D 3 MB,4GB memory)
  - GCC 3.4.3
  - Intel 8.1
- Opteron (2.0 GHz,L1D 64kB, L2D 1MB, 4GB memory)
  - GCC 3.2.3
  - Pathscale 2.0

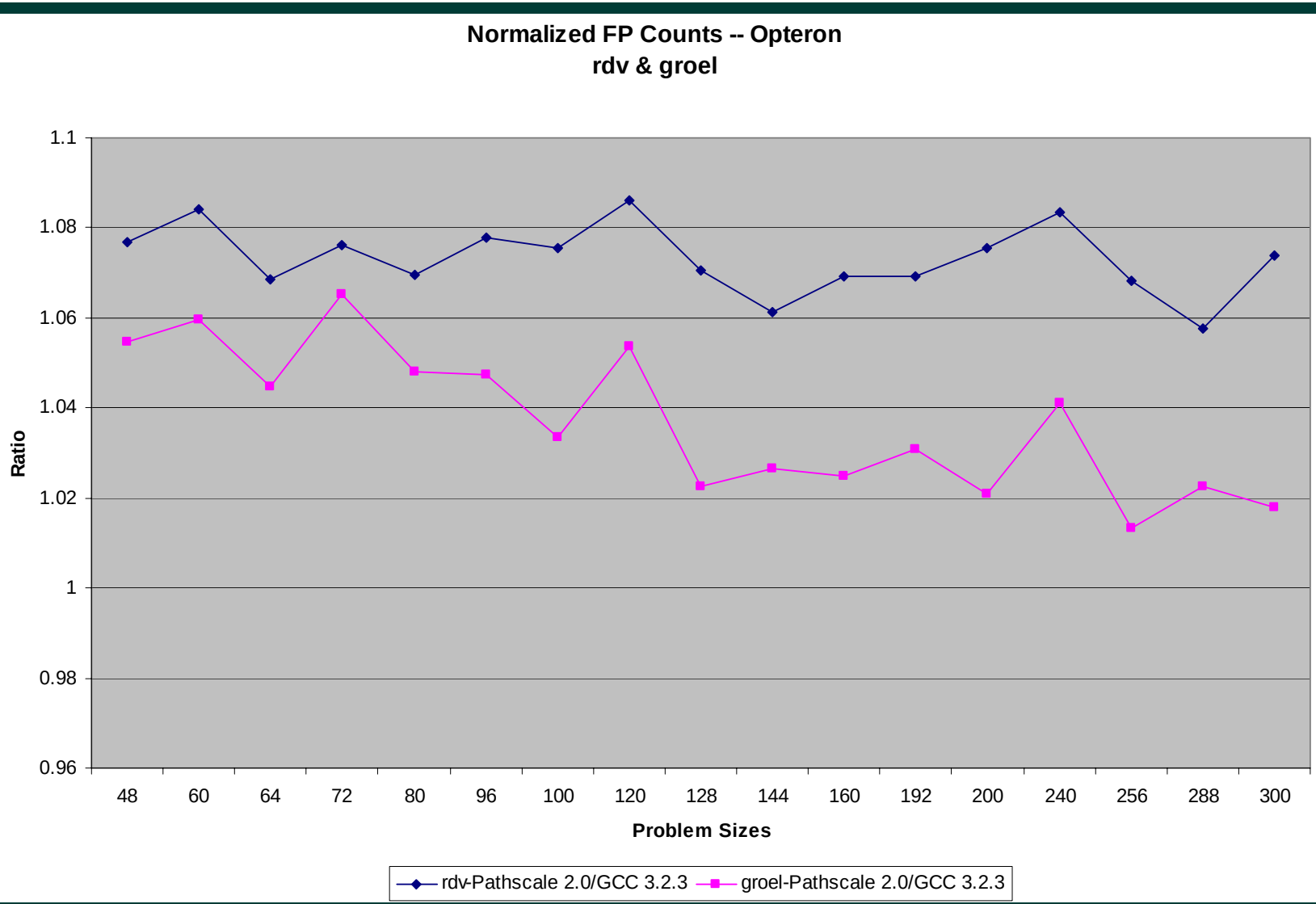
Notice:

1) *each data point is the average of 10 runs on each platform*
2) *The hardware counters' stability on Itanium shows no more than 10^{-5} relative variation*
3) *The hardware counter's stability on Opteron shows 10^{-4} relative variation for several cases with occasional variations higher than 10^{-3}*

Itanium2 floating-point operations



Opteron floating-point operations



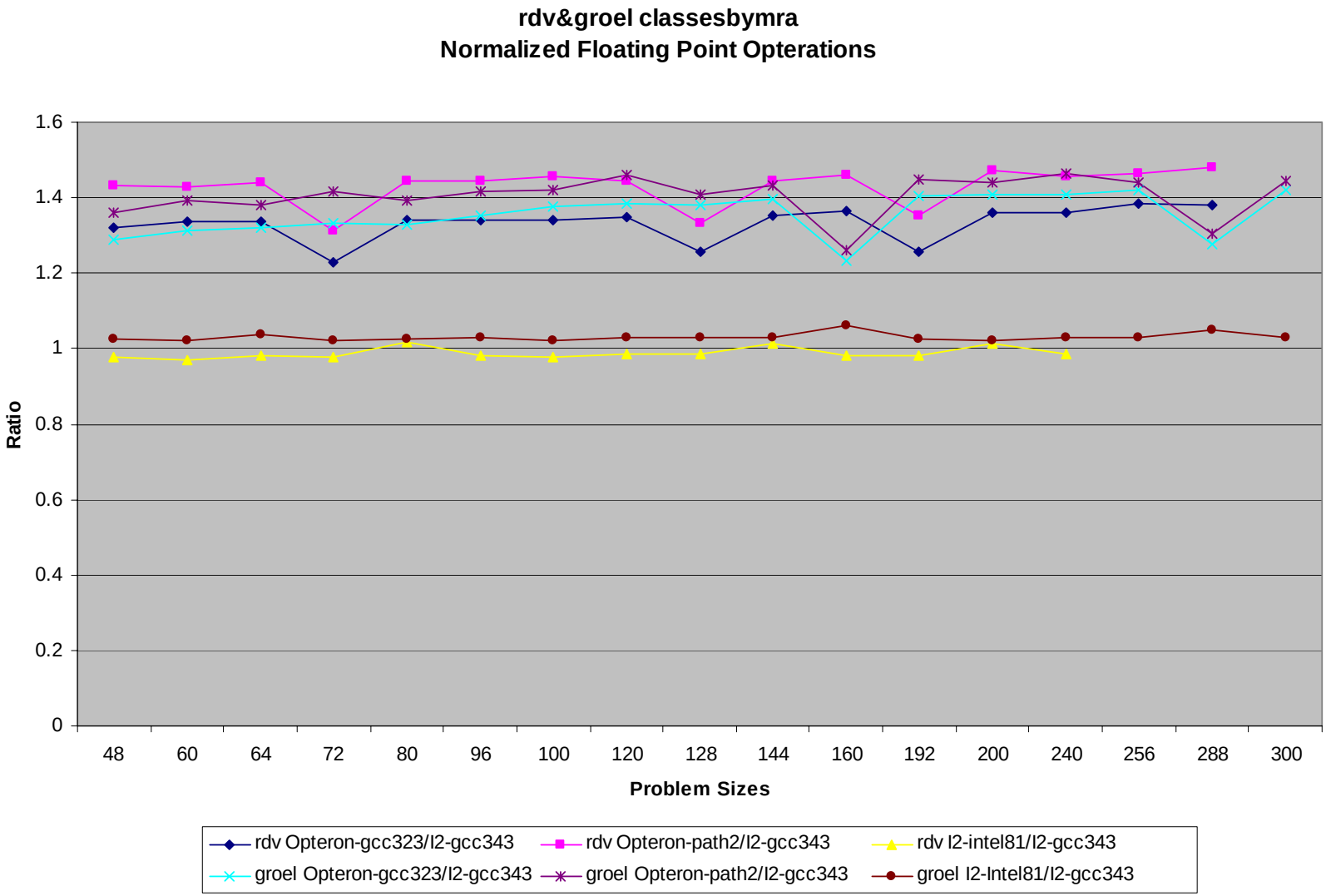
Compiler influence on FP ops count

- Itanium2:
  - Groel: Intel 8.1 on average generates 3.1% more floating-point instructions (max 6.1% more, min 2.1% more) than GCC 3.4.3
  - Rdv: Intel 8.1 on average generates 1.2% fewer floating-point instructions (max 1.6% more, min 2.9% fewer) than GCC 3.4.3

- Opteron:
  - Groel: Pathscale 2.0 on average generates 3.7% more floating-point instructions (max 6.5% more, min 1.3% more) than GCC 3.2.3
  - Rdv: Pathscale 2.0 on average generates 7.3% more floating-point instructions (max 8.6% more, min 6.1% more) than GCC 3.2.3

Notice: classesbymra for the Groel and Rdv data sets uses the same binaries on both platforms, but they have different control parameters

Cross-platform FP Ops comparison



FP Ops Counts — IA64

- Command: pfmon -eFP_OPS_RETIRED --irange=main *program*
- FP_OPS_RETIRED: Retired FP Operations
  - Provides information on number of retired floating-point operations, excluding all predicated off instructions
  - A weighted sum of basic floating-point operations
  - To count how often specific opcodes are retired, use IA64_TAGGED_INST_RETIRED
    - Counted as 4 ops: fpma, fpms, and fpnm
    - Counted as 2 ops: fpma, fpnma (f2=f0), fma, fms, fnma, fprcpa, fprsqrta,fpmpy, fpmax, fpamin, fpamax, fpcmp, fpcvt
    - Counted as 1 op: fms, fma, fnma (f2=f0 or f4=f1), fmpy, fadd, fsub, frcpa,frsqrta, fmin, fmax, famin, famax, fpmin, fcvt.fx, fcmp

FP Ops Counts — Opteron

- Command: perfex --event=0x00430300 *program*
- K8_DISPATCHED_FPU_OPS:
  - 0x00:0xF:0x3F(EVENT_MASK:COUNTER:UNIT_MASK)
  - UNIT_MASK bit
    - 0 Add pipe ops excluding junk ops
    - 1 Multiply pipe ops excluding junk ops
    - 2 Store pipe ops excluding junk ops
    - 3 Add pipe junk ops
    - 4 Multiply pipe junk ops
    - 5 Store pipe junk ops
    - 6-7 Reserved
  - DISPATCHED_FP_OPS_MUL_ADD 0x00430300
  - DISPATCHED_FP_OPS_ALL 0x00433F00

Cross-platform “stability”

- GCC 3.2.3 on the Opteron on average generates 35.6% more (max 42.1% more, min 23.1% more) floating-point instructions than GCC 3.4.3 on Itanium2 for groel
- GCC 3.2.3 on the Opteron on average generates 33.2% more (max 38.4% more, min 22.7% more) floating-point instructions than GCC 3.4.3 on Itanium2 for rdv

Cross-platform “stability”

- GCC 3.2.3 on the Opteron on average generates 31.6% more (max 38.2% more, min 16.0% more) floating-point instructions than Intel 8.1on Itanium2 for groel
- GCC 3.2.3 on the Opteron on average generates 33.9% more (max 38.8% more, min 20.8% more) floating-point instructions than Intel 8.1on Itanium2 for rdv

Workload based on counters - Conclusion

- Architectures and their hardware counters may be too different to assert that workload determined through counters are independent of the platform on which the data is collected.

Single Node Performance Modeling

- Use hardware counters to measure the application's workload
- Use Memory Reuse Distance to capture the application's memory access pattern
- Combined with system micro-benchmark to measure unavailable hardware features, e.g. MissPenalty
- The developed models are for Intel IA64, IA32 and x86_64

$$EstimatedExeTime(psize) = k_0 \left(A + B + C + D\right) / CpuClock(arch) \quad \cdots\cdots (1)$$

$$A = k_1 \times \left(totalFp(psize) / FpPipeline\,Num(arch) \right) \times FpRepeatRate(arch) \quad \cdots\cdots (2)$$

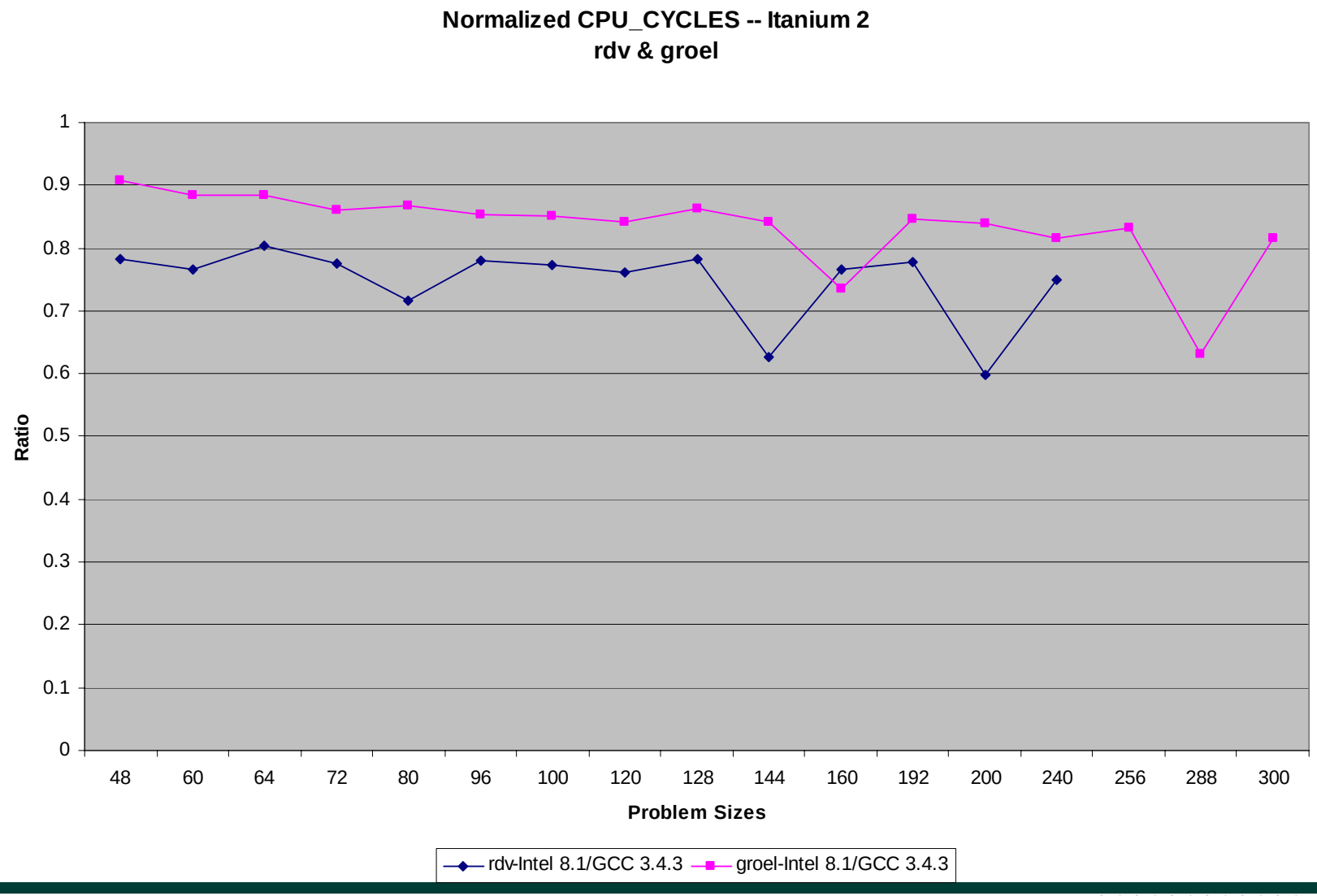
$$B = k_2 \times L1MissCount(psize) \times L1MissPenalty(arch) \quad \cdots\cdots (3)$$

$$C = k_3 \times L2MissCount(psize) \times L2MissPenalty(arch) \quad \cdots\cdots (4)$$

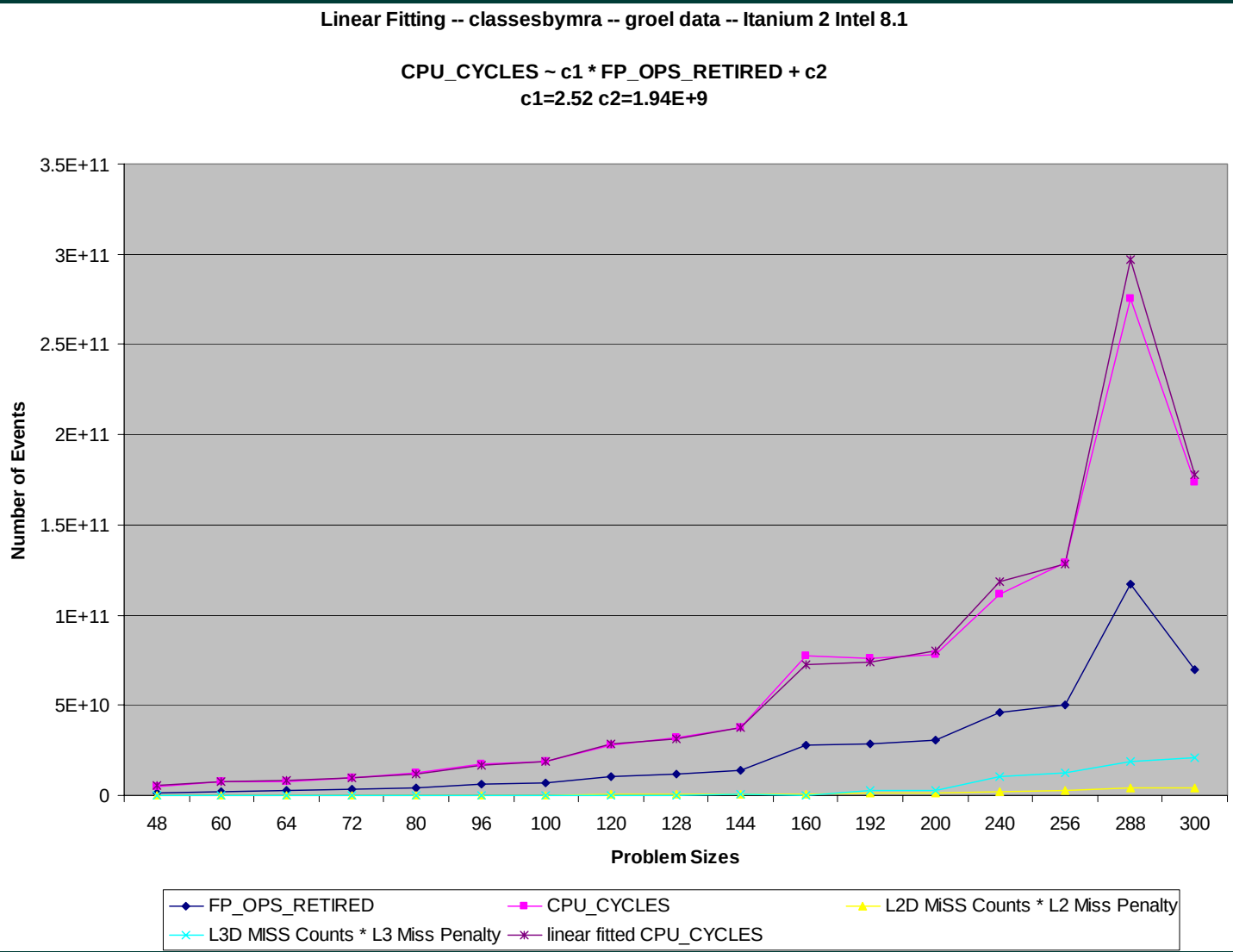
$$D = k_4 \times L3MissCount(psize) \times L3MissPenalty(arch) \quad \cdots\cdots (5)$$

$$L(j)MissPenalty(arch) = L(j+1)Lantency(arch) - L(j)Latency(arch) \quad \cdots\cdots (6)$$

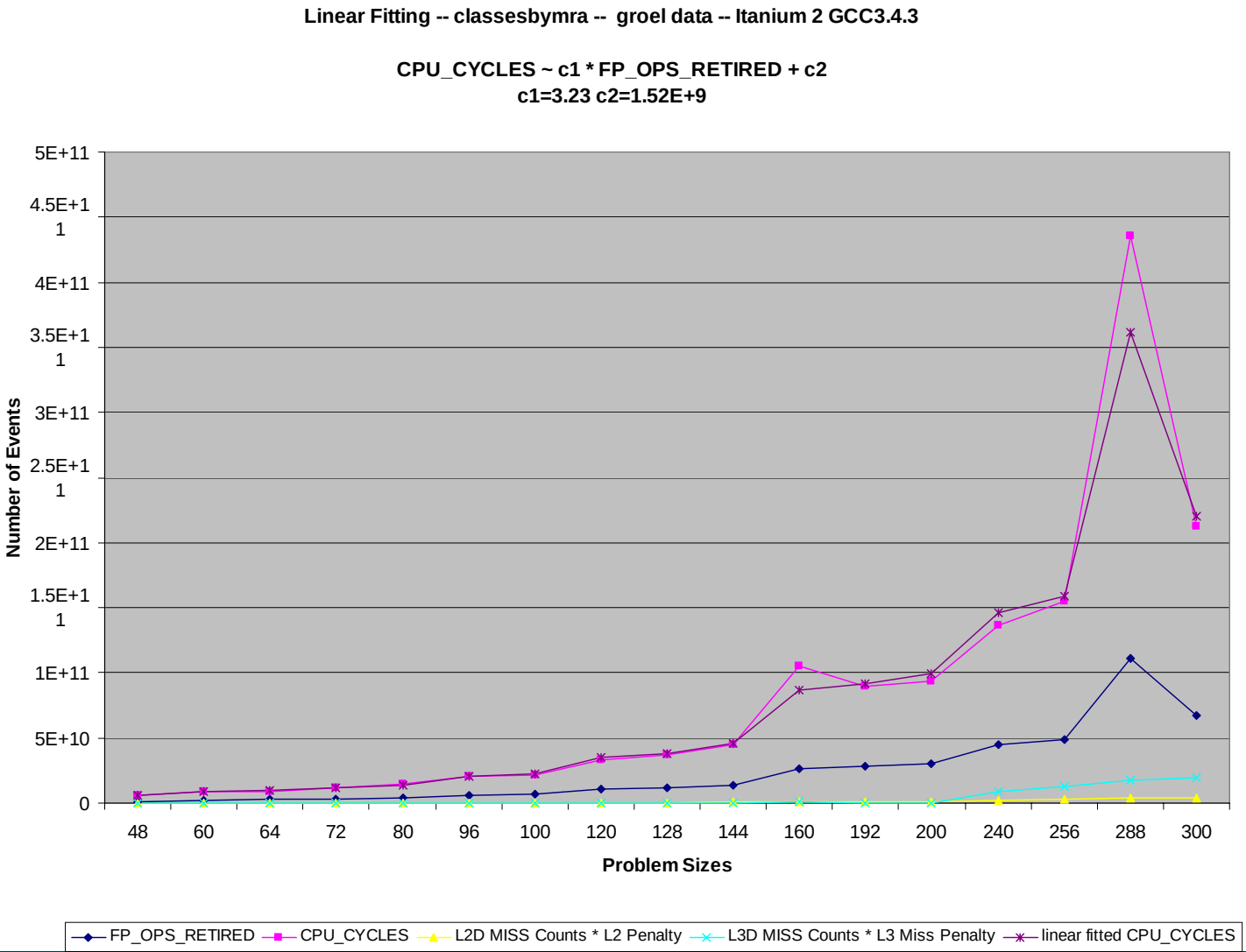
Itanium2: compiler influence on cycle count



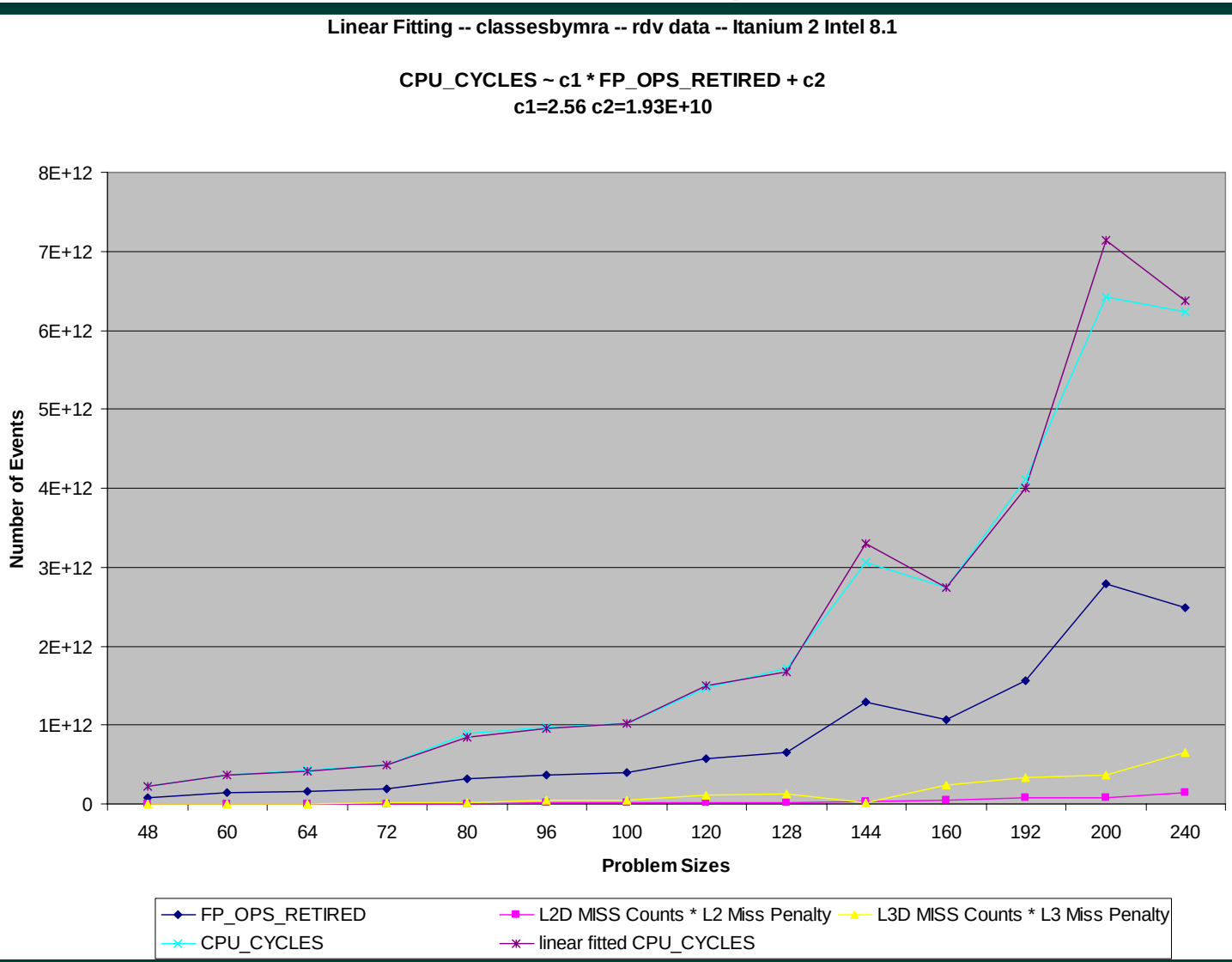
Itanium2: groel, Intel 8.1



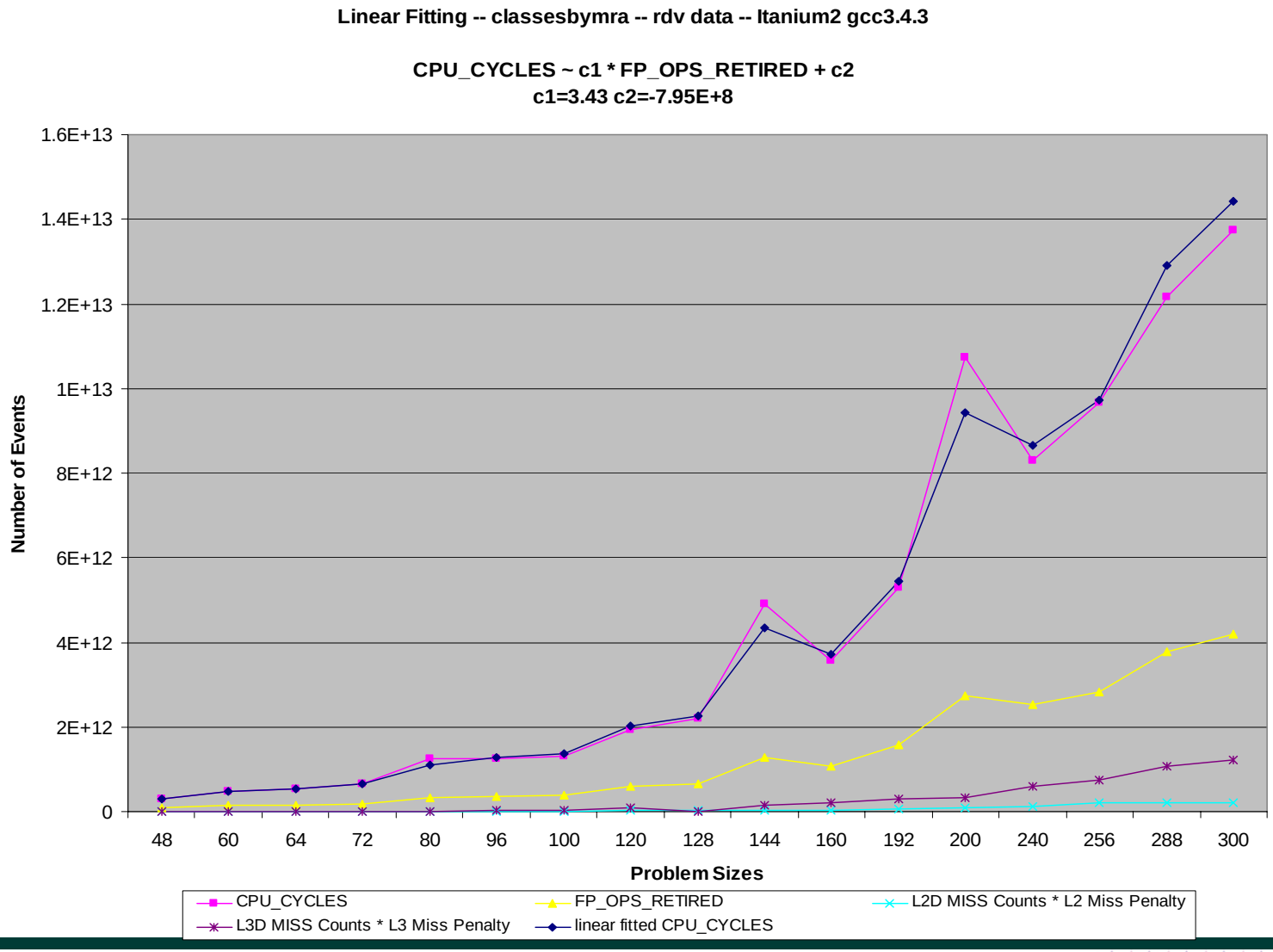
Itanium2: groel, GCC 3.4.3



Itanium2: rdv, Intel 8.1



Itanium2: rdv, GCC 3.4.3

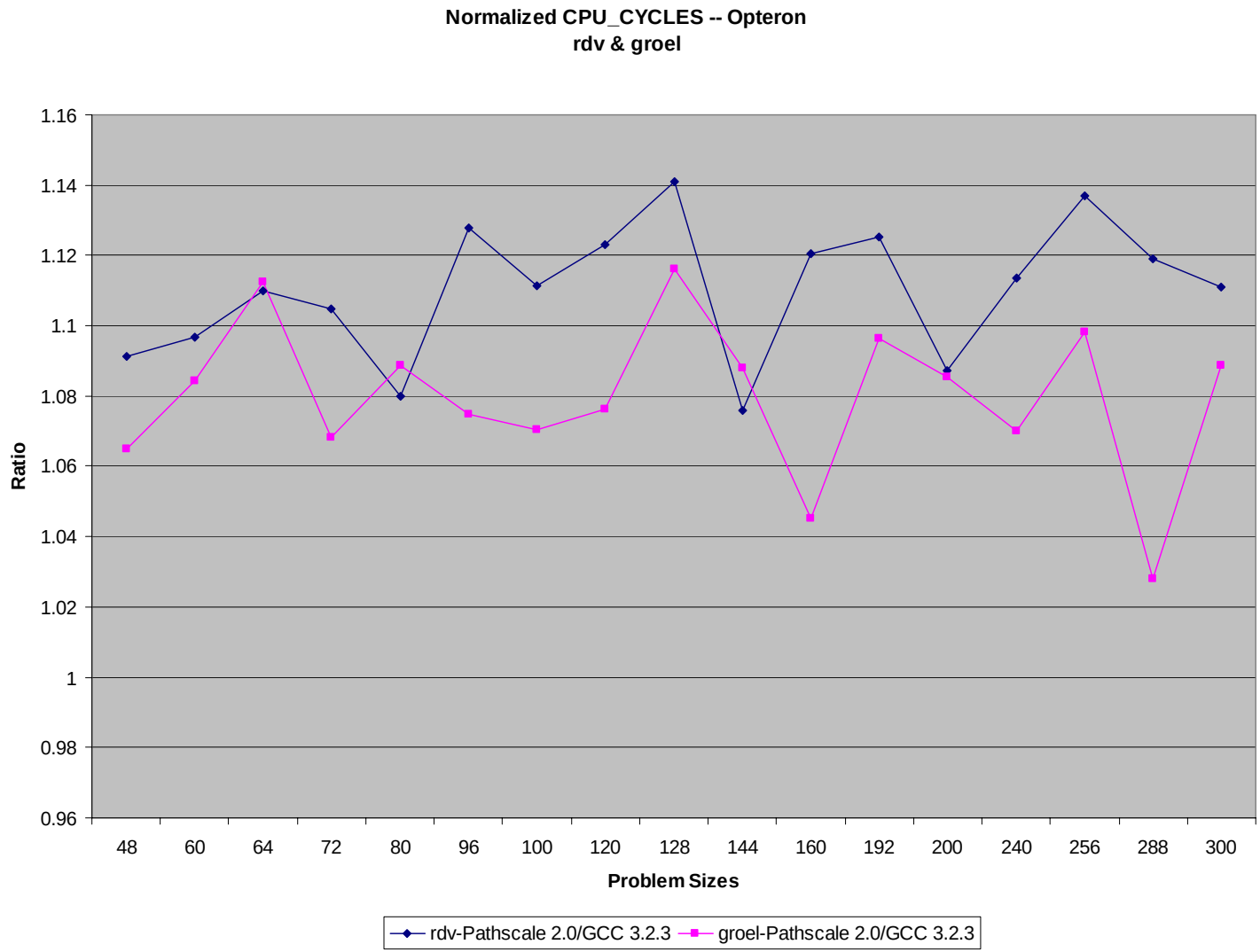


Memory system conclusions

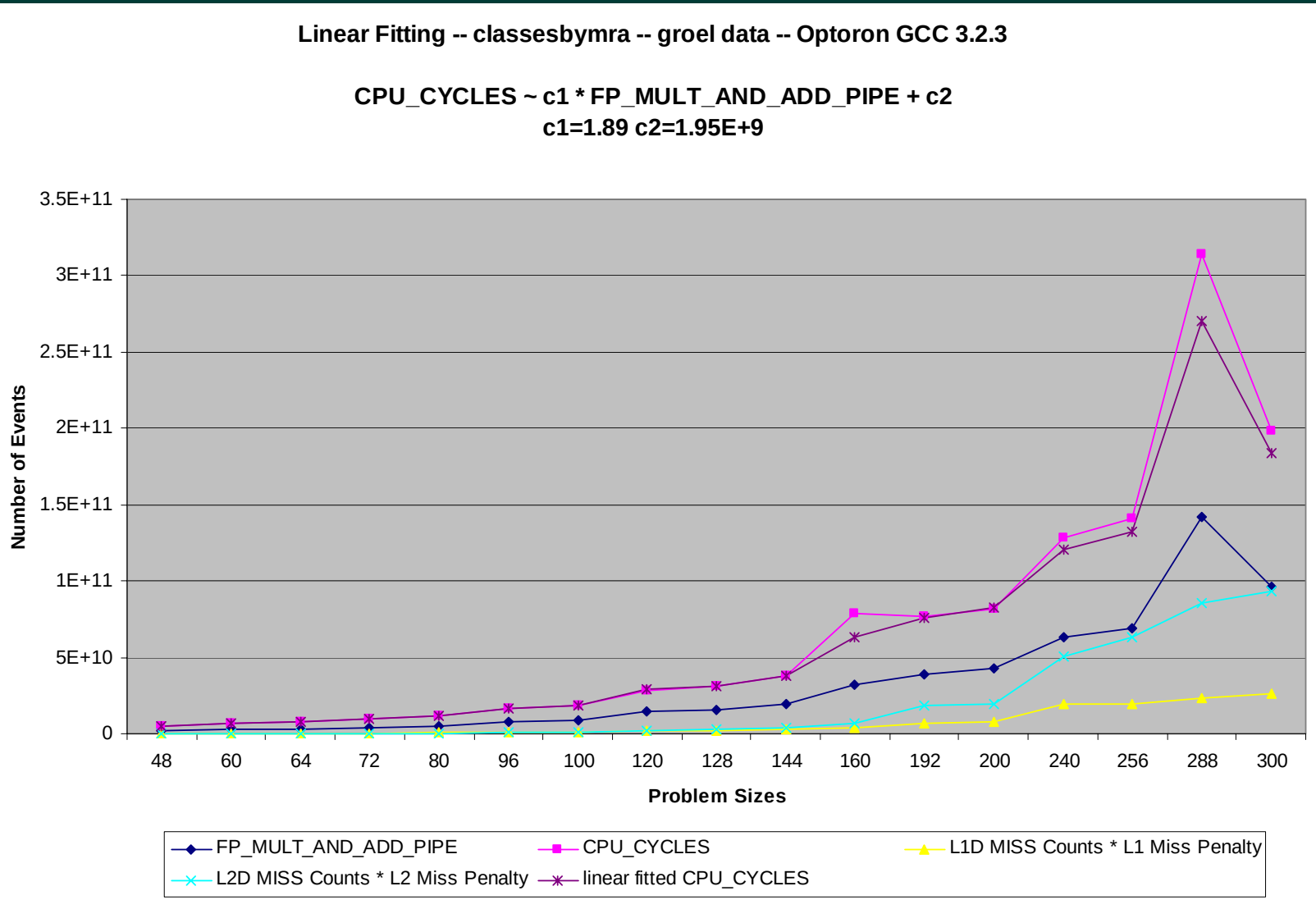
- Itanium2
  - L2 data misses account for
    - rdv
      - 0.4% ~ 2.1% for GCC 3.4.3 and
      - 0.7% ~ 2.1% for Intel 8.1
    - groel
      - 0.7% ~ 1.9% for GCC 3.4.3 and
      - 1% ~ 2.4% for Intel 8.1
  - L3 data misses account for
    - rdv
      - 0.2% ~ 8.9% for GCC 3.4.3 and
      - 0.3% ~ 10.6% for Intel 8.1
    - Groel
      - 0.05% ~ 9% for GCC 3.4.3 and
      - 0.08% ~ 11.9% for Intel 8.1

- The difference in cycle count is not captured by the difference in workload or the difference in cache behaviors for GCC 3.4.3 and Intel 8.1 on the Itanium.

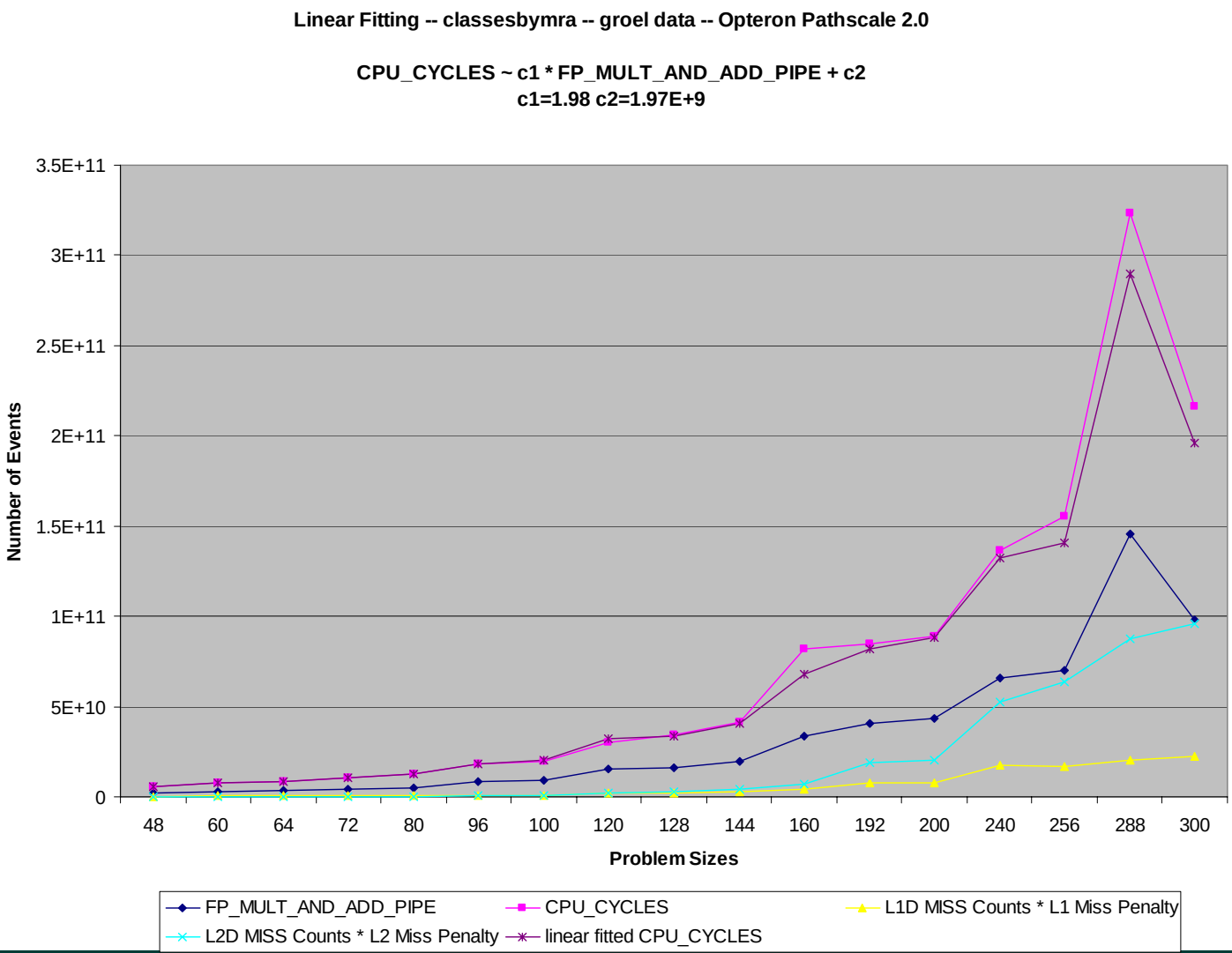
Opteron: Compiler influence on cycle count



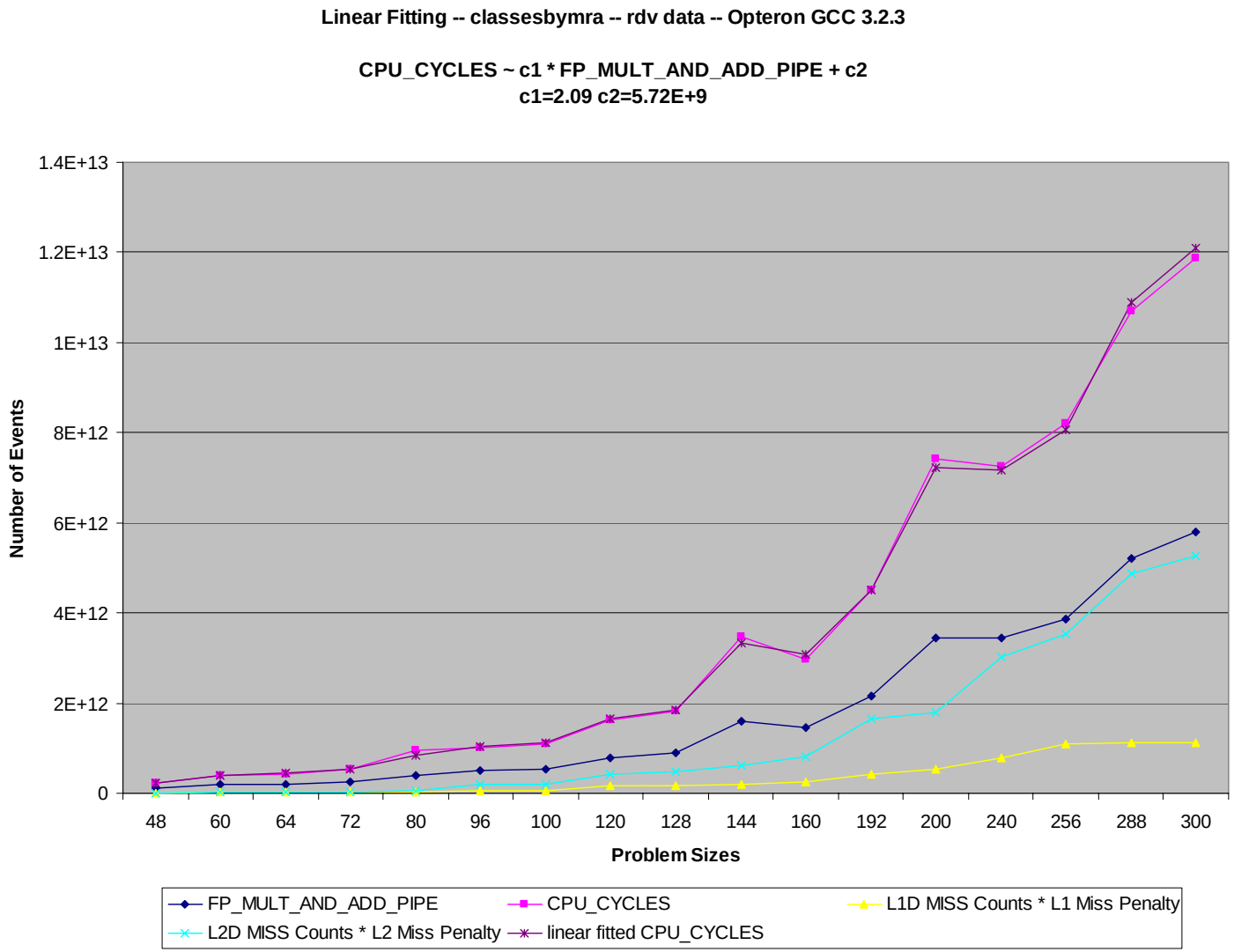
Opteron: groel, GCC 3.2.3



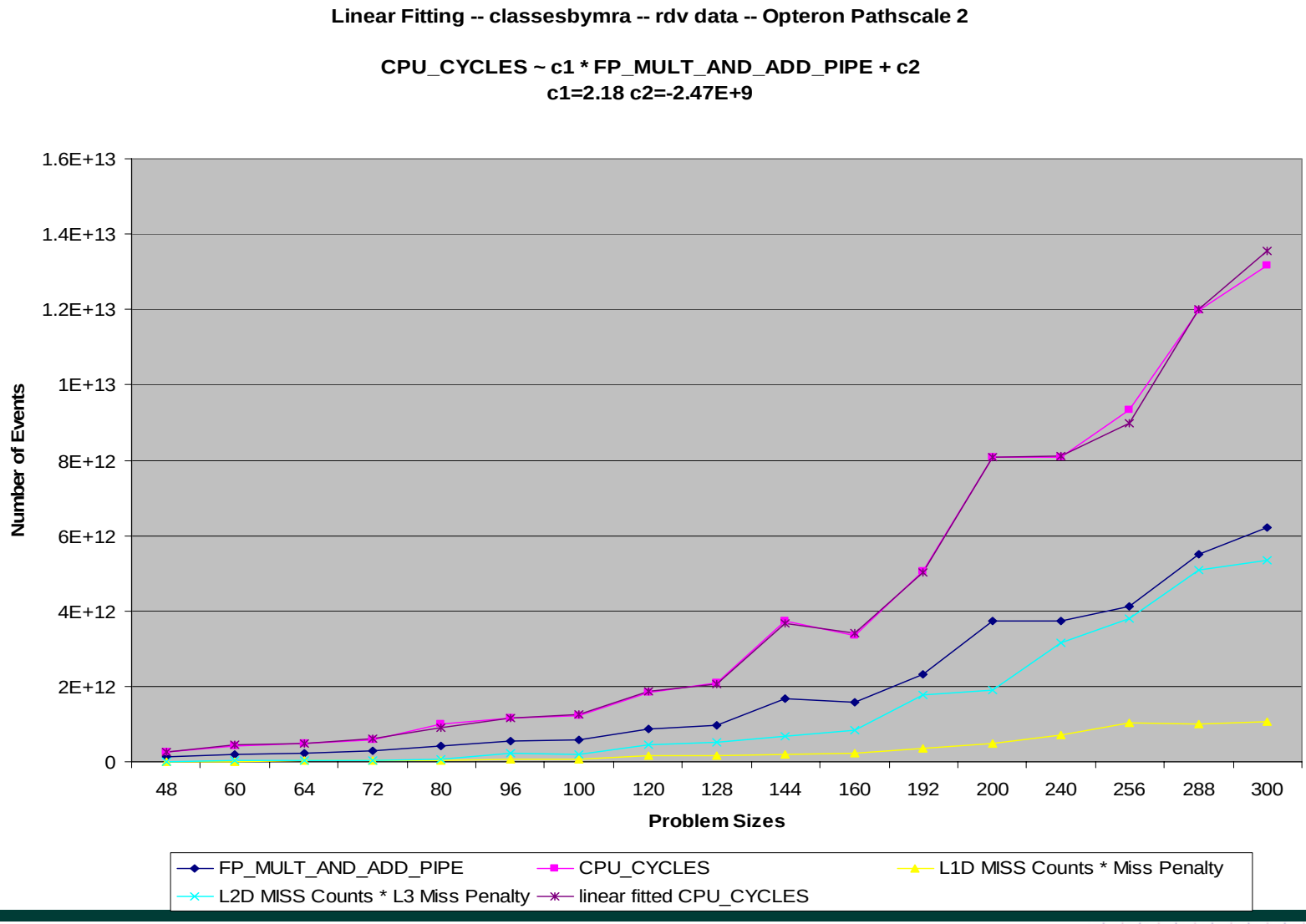
Opteron: groel, Pathscale 2.0



Opteron: rdv, GCC 3.2.3



Opteron: rdv, Pathscale 2.0m



Memory system conclusions

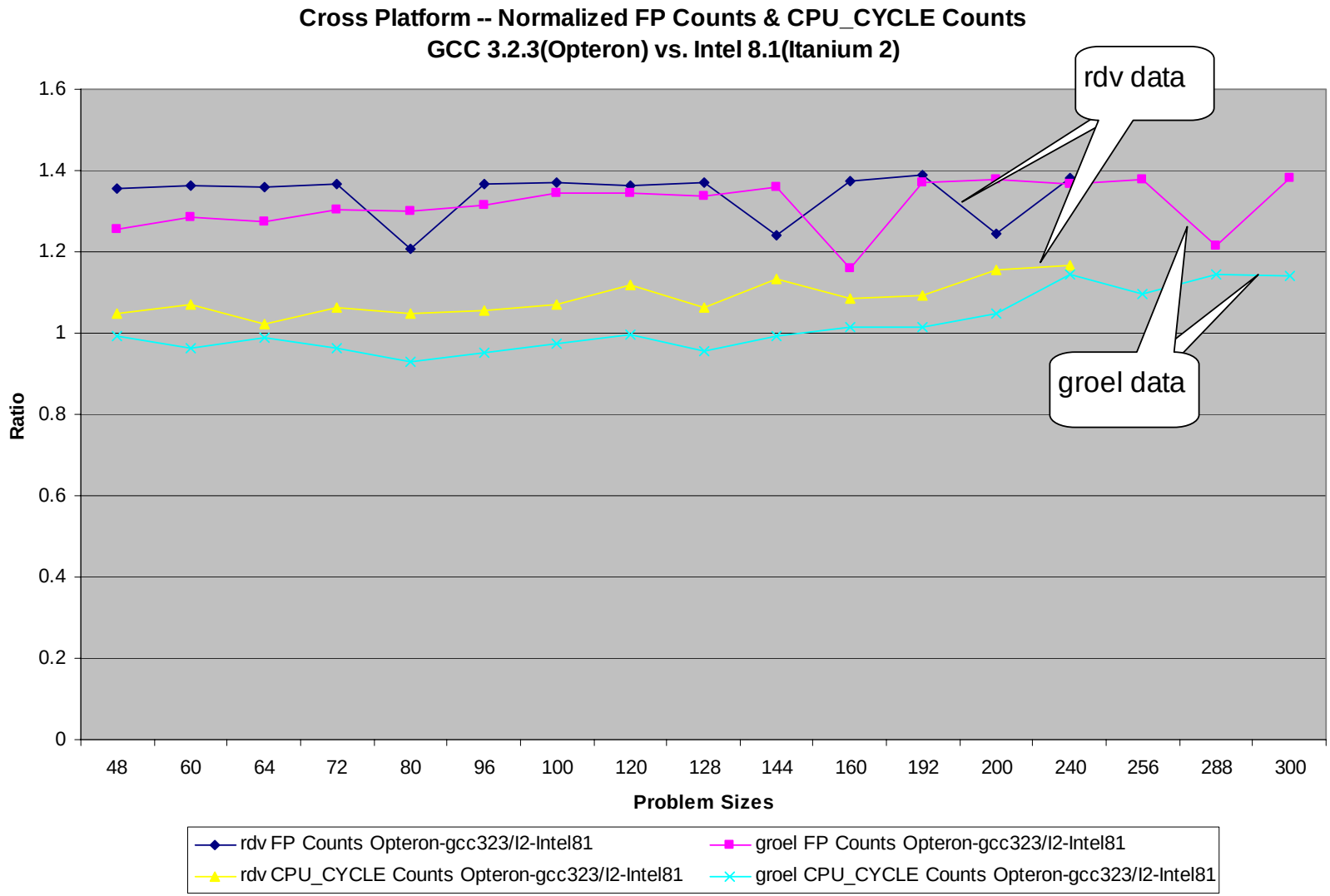
- Opteron
  - L1 data misses account for
    - rdv
      - 2.8% ~ 13.4% for GCC 3.2.3 and
      - 2.6% ~ 10.9% for Pathscale 2.0
    - groel
      - 4.5% ~ 15% for GCC 3.2.3 and
      - 4.5% ~ 12.6% for Pathscale 2.0
  - L2 data misses account for
    - rdv
      - 0.7% ~ 45.6% for GCC 3.2.3 and
      - 1.3% ~ 42.6% for Pathscale 2.0
    - groel
      - 2.1% ~ 47.1% for GCC 3.2.3 and
      - 2.0% ~ 44.3% for Pathscale 2.0

- Small difference in cache behavior for Pathscale and GCC 3.2.3 with a slight advantage for Pathscale.

Compiler influence on Cycle count

- Itanium2:
  - Groel: code generated by Intel 8.1 on average requires 16.6% fewer cycles (max 36.8% fewer cycles, min 9.3% fewer cycles) than code generated by GCC 3.4.3
  - Rdv: code generated by Intel 8.1 on average requires 25.3% fewer cycles (max 40.2% fewer cycles, min 19.7% fewer cycles) than code generated by GCC 3.4.3
- Opteron:
  - Groel: code generated by Pathscale 2.0 on average requires 8.0% more cycles (max 11.6% more cycles, min 2.8% more cycles) than code generated by GCC 3.2.3
  - Rdv: code generated by Pathscale 2.0 on average requires 11.0% more cycles (max 14.1% more cycles, min 7.6% more cycles) than code generated by GCC 3.2.3

Cross-platform cycle count stability



Cycle count conclusion

- Groel: On average the Opteron requires 1.8% more cycles (max 14.6% more, min 6.9% fewer) than the Itanium2 with GCC 3.2.3 on the Opteron (best) and Intel 8.1 on the Itanium2 (best). There is a trend though towards more cycles for the Opteron for larger micrographs.
- Rdv:On average the Opteron requires 8.5% more cycles (max 16.6%, min 2.0%) than the Itanium2 with GCC 3.2.3 on the Opteron (best) and Intel 8.1 on the Itanium2 (best). There is a trend though towards more cycles for the Opteron for larger micrographs.

CPU_CYCLES=C1*Floats + C2

	Itanium 2					Opteron				
	c1	c2	min cycle counts	max cycle counts	Max abs relative error	c1	c2	min cycle counts	max cycle counts	Max abs relative error
rdv GCC	3.43	-7.95E+8	2.89E+11	1.37E+13	12.3%	2.09	5.72E+9	2.36E+11	1.19E+13	11.9%
rdv Intel/Pathscale	2.56	1.93E+10	2.26E+11	6.42E+12	11.1%	2.18	-2.47E+9	2.58E+11	1.32E+13	9.9%
groel GCC	3.23	1.52E+9	5.73E+9	4.35E+11	17.8%	1.89	1.95E+9	5.16E+9	3.14E+11	19.6%
groel Intel/Pathscale	2.52	1.94E+9	5.20E+9	2.75E+11	7.9%	1.98	1.97E+9	5.50E+9	3.23E+11	17.3%

Plans

- Do we need to time codes for “all” interesting compilers on “all” platforms for desired accuracy?
- How well does the application input and parameter space need to be covered for desirable accuracy?