A brief correlation study of x86 compiler flags and performance events

2nd Annual Concurrency Forum Meeting
February 5th 2013

Andrzej Nowak, CERN openlab

Based on the work of Mirela-Madalina Botezatu and Andrzej Nowak

CERN openlab

Overview

1. Motivation and open questions
2. Superficial comparisons of GCC and ICC
3. Compiler flag mixes
4. PMU event correlations
5. Bottleneck identification
6. Compiler flag prediction

A full technical report by Mirela Botezatu is available on the openlab website

Motivation

- Out of all performance dimensions, ILP and pipelining are those over which we have very little control
- The quality of a compiler determines the quality of the binary code run on a system
- The programmer controls the compiler through its many flags
- Performance events are a powerful tool, but at the same time difficult to use

Our questions

- **Can we combine knowledge about compiler flags and the response they produce in hardware?**
  - Can we automatically characterize benchmarks?
  - Which compiler flags are beneficial on which code?
  - Can we predict which ones to use depending on the workload?
  - Is compile time a concern?

Study setup

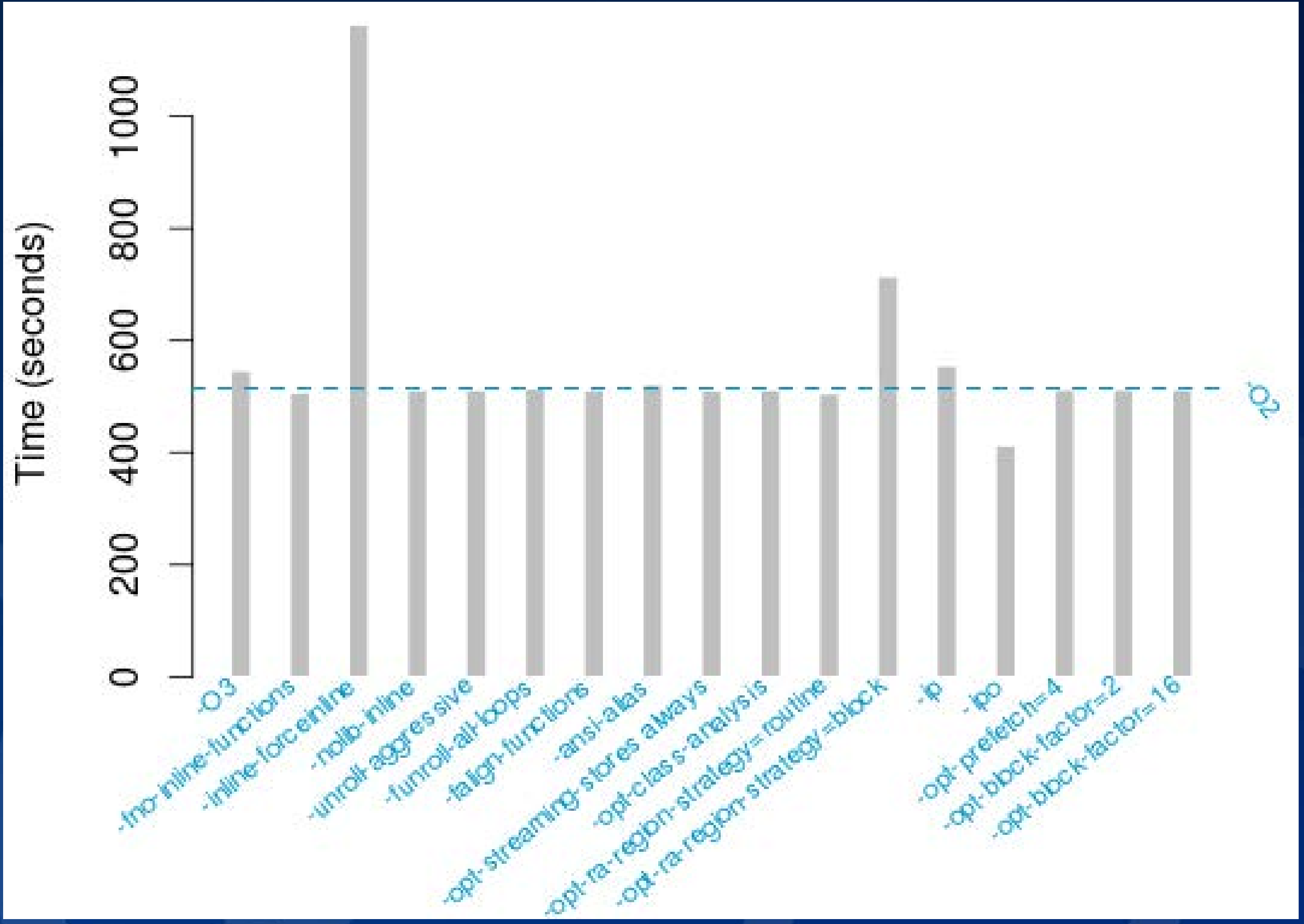
- **Master-slave setup with 25 machines running measurements in parallel, 29’000 test runs**
- **Hardware:**
  - 25 dual socket Westmere-EP servers
  - 24 threads each @ 2.7GHz
  - HT on
  - 3.6 kernel
- **Benchmarks**
  - HEP snippets
  - ROOT benchmarks
  - I/O intensive benchmarks from GOODA
  - Adobe C++ benchmarks
  - FFT

Artificial benchmarks (Adobe)
GCC 4.6.3 vs. ICC 13.0.1

Benchmark	Exec. time GCC –O2	Exec. time ICC –O2	ICC Gain	Exec. time GCC –O3	Exec. time ICC –O3	ICC Gain
functionobjects.cpp	245.05	238.60	**2%**	240.97	240.58	**0%**
loop_unroll.cpp	383.04	198.63	**48%**	388.93	167.63	**56%**
Simple_types_constant_folding.cpp	104.33	155.6	**-49%**	97.05	155.79	**-59%**
Simple_types_loop_invariant.cpp	354.92	245.38	**30%**	333.19	245.13	**26%**
Stepanov_abstraction.cpp	248.99	213.49	**14%**	245.77	234.73	**4%**
Stepanov_vector.cpp	301.38	214.303	**28%**	303.06	228.004	**24%**

Time measured in seconds

ICC compile time
HEPSPEC06, various flags tested



GCC-ICC potential sources of differences

- Inlining: at O2 in ICC, at O3 in GCC
- IPO: at O2 in ICC
- Vectorization: at O2 in ICC
- Strict aliasing: At O2 in GCC, in ICC you have to ask for it explicitly
- Loop unrolling: O2 in ICC, but only some loop optimizations available in GCC with “frerun-loop-opt”
- ICC uses optimized math library functions by default

Correlations of ICC flag usage and performance

- **Not included were flags that:**
  - disregard strict standards compliance
  - are enabled by default
  - “tune for this architecture”
- **Split between CPU intensive and I/O intensive benchmarks (27 and 10 benchmarks respectively)**
- **If we use flag A, <u>is there</u> speed increase?**
  - 1% threshold
- **What if we combine multiple flags?**
- **What if we use the PMU to monitor performance response?**

Gains (top) and regressions (bottom)

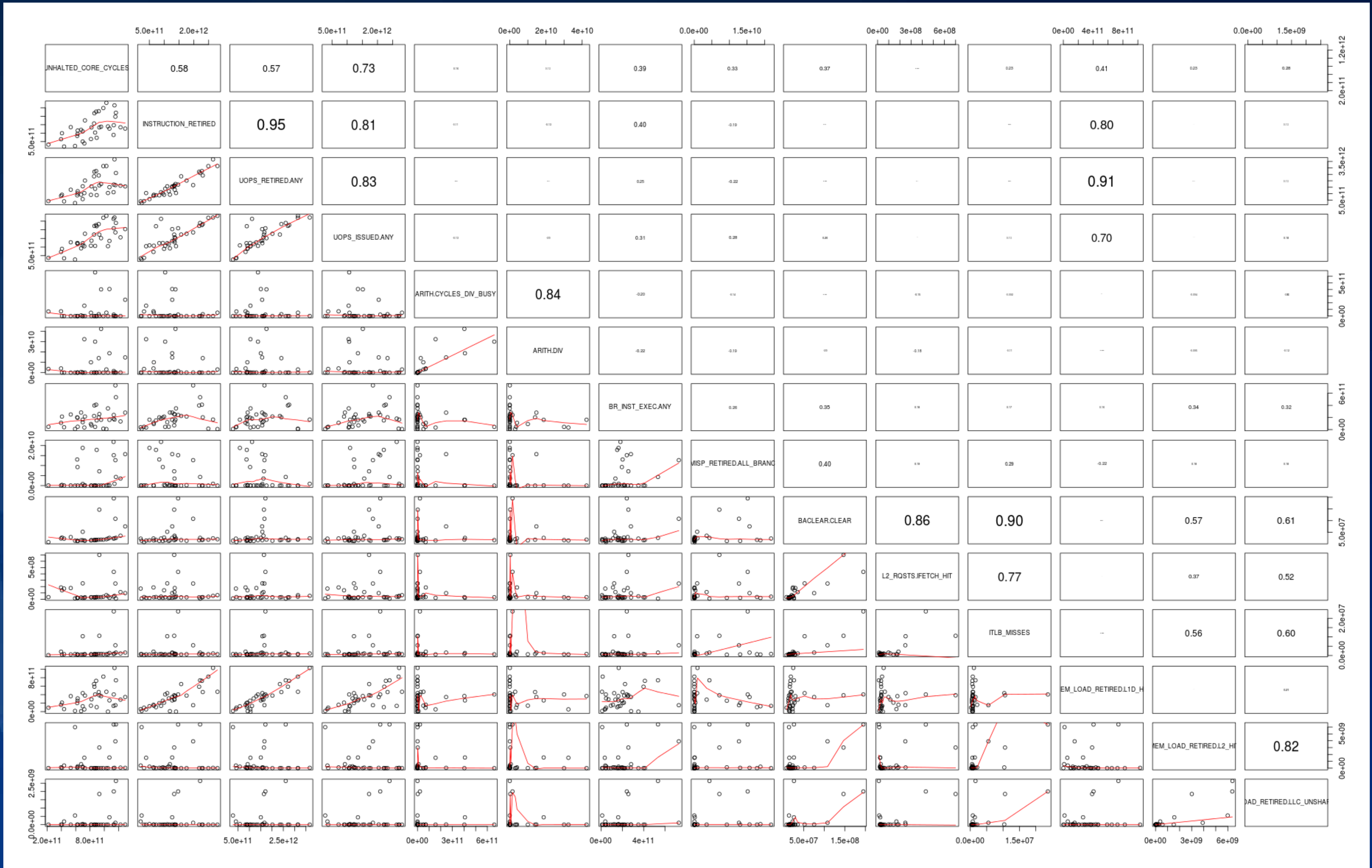
Compiler flag	Counts	Compiler flag	Counts
O3	963	Opt-streaming-stores-always	694
Ipo	951	Ansi-alias	686
Opt-ra-region-strategy=routine	821	Opt-prefetch=4	674
Ip	761	Faling-functions	657
Opt-ra-region-strategy=block	760	Unroll-aggressive	652
Funroll-all-loops	753	fno-inline-functions	628
Nolib-inline	740	ipo	694
Inline-forceinline	738	Opt-block-factor=16	616
Opt-class-analysis	700	Opt-block-factor=2	608

Compiler flag	Counts	Compiler flag	Counts
Opt-streaming-stores-always	1071	Ansi-alias	686
Nolib-inline	1004	Opt-prefetch=4	675
O3	838	Funroll-all-loops	673
Ipo	822	Inline-forceinline	665
Opt-ra-region-strategy=block	818	Unroll-aggressive	656
fno-inline-functions	773	Opt-class-analysis	647
Opt-ra-region-strategy=routine	757	Opt-block-factor=16	590
Ip	710	Opt-block-factor=2	586

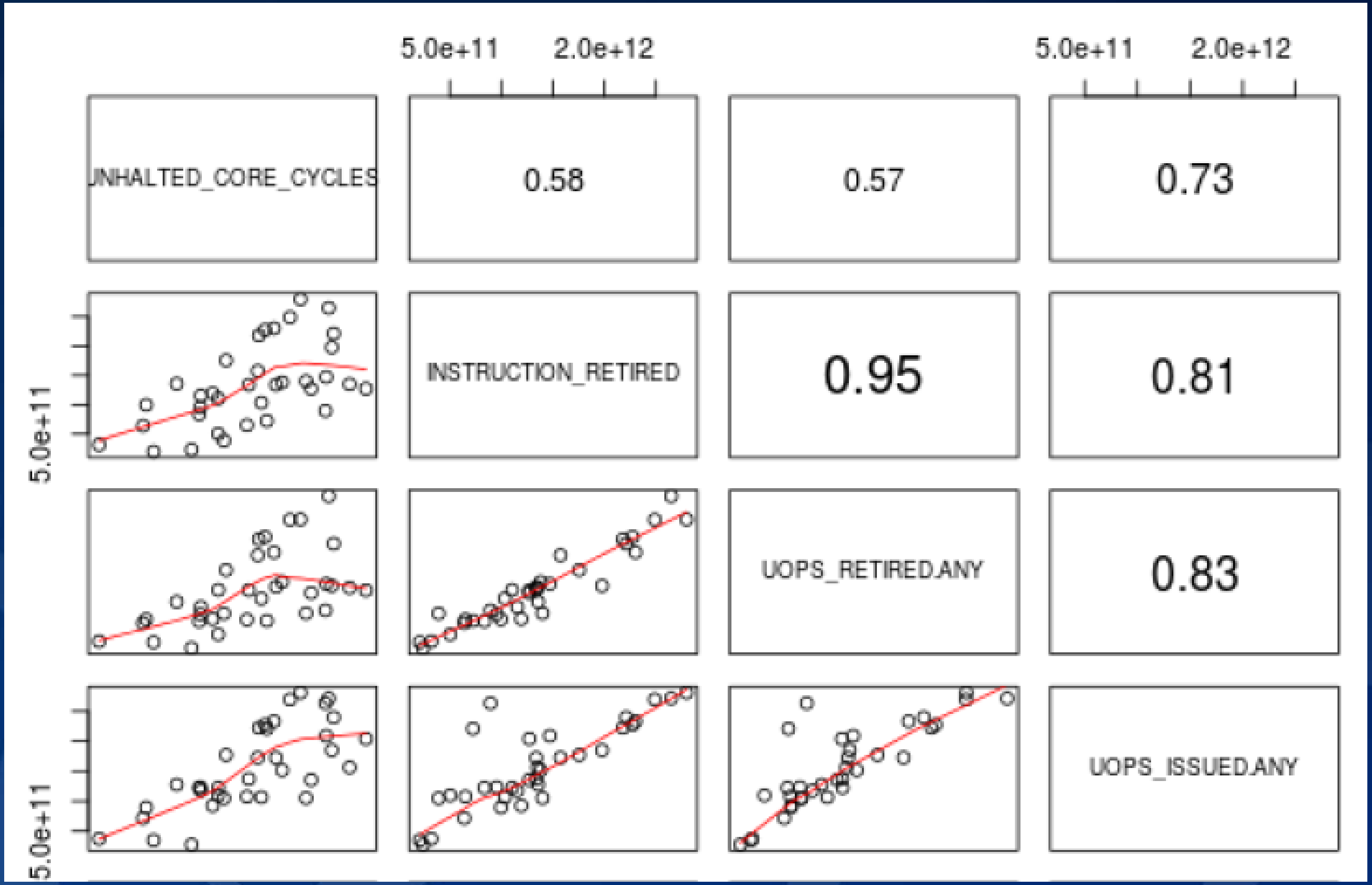
PMU event counting

- **Most quite stable with low run to run variations**
- **Some (predictably) unstable:**
  - MEM UNCORE RETIRED.REMOTE DRAM
  - MEM UNCORE RETIRED.REMOTE HITM
- **Tip: control process and memory pinning**

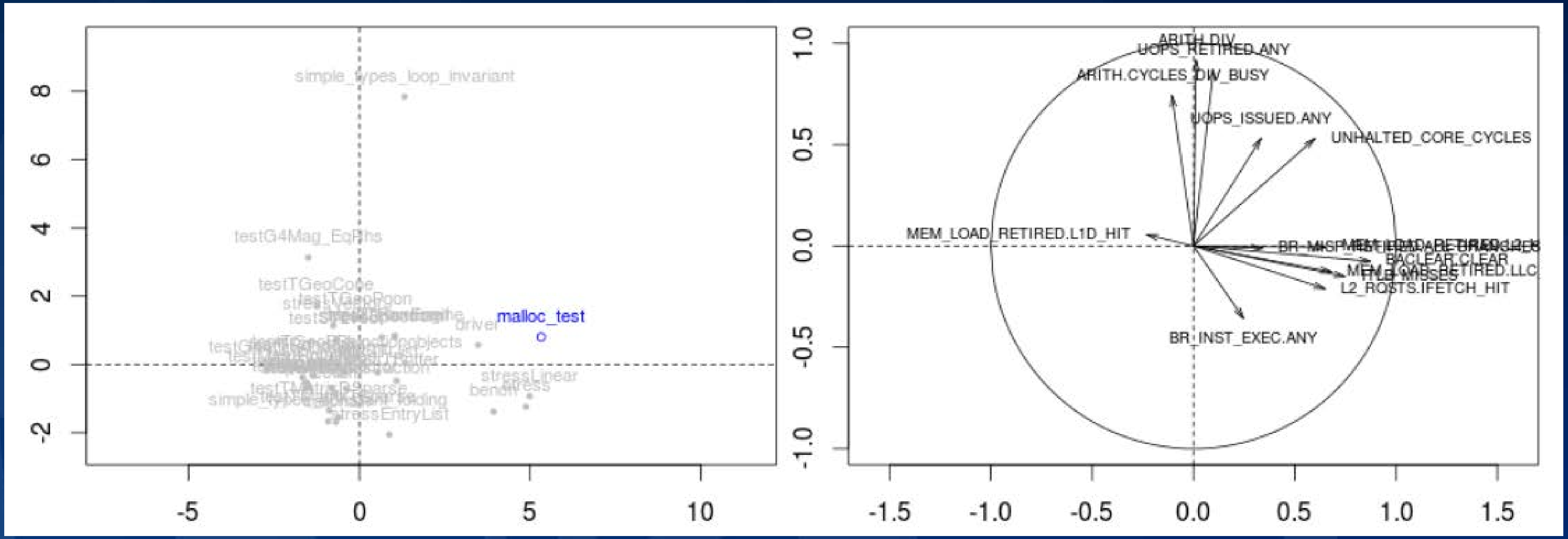
PMU event correlations (1)



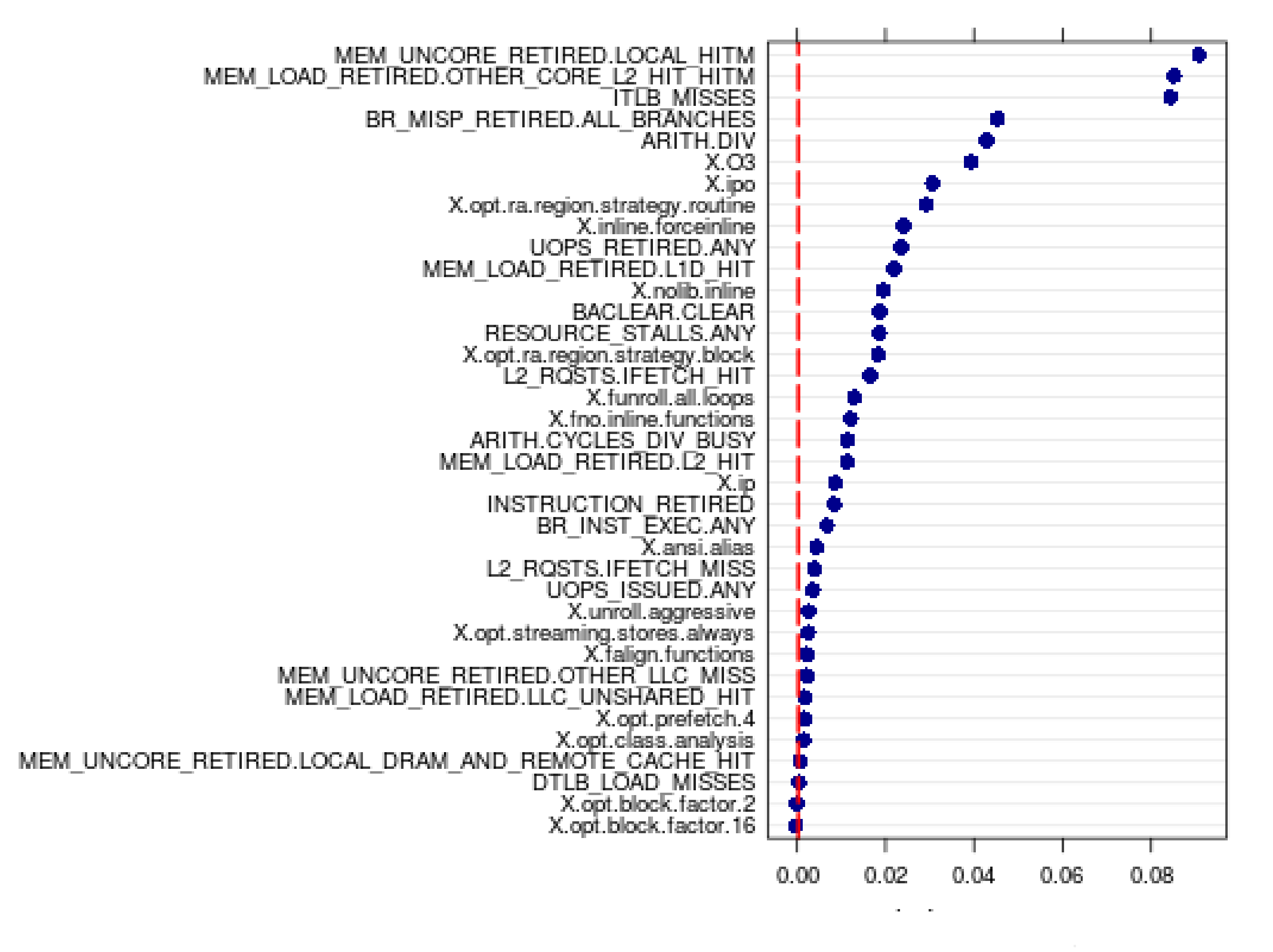
PMU event correlations (2)



Bottleneck identification



Compiler flag prediction - difficulties



Summary and conclusions

- **Results of similar experiments were difficult to reproduce**
- **It is possible to semi-automatically characterize benchmarks**
- **It is possible to establish which compiler flags would be likely to reduce a particular bottleneck**
- **It is difficult to predict with good accuracy which compiler flags will improve a particular workload**
- **Remarks:**
  - There is potential in this approach, but more detailed information about the program needs to be considered in a (possibly) multi-stage approach
  - Similar work (FDO with PMU events) is ongoing with relation to the GOODA profiler (Baptiste Wicht) and elsewhere

THANK YOU

Q & A

CERN openlab

Questions? Andrzej.Nowak@cern.ch