Triggering Performance Counters for Energy Efficiency Measurements

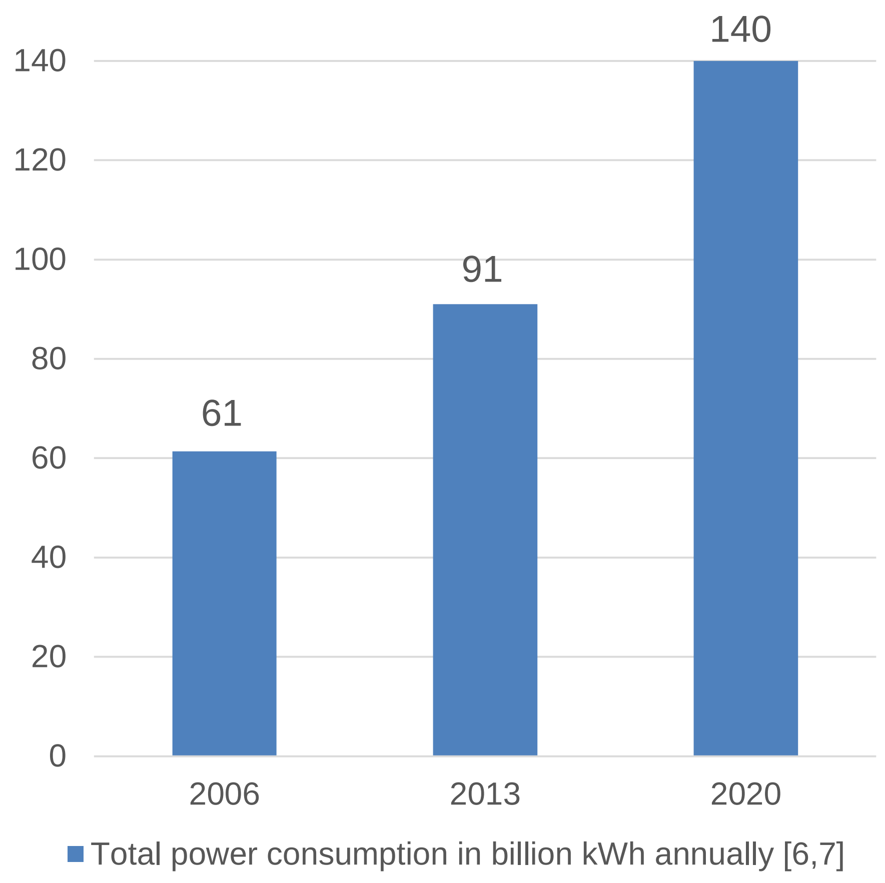
Norbert Schmitt
Jóakim v. Kistowski
Samuel Kounev

Chair of Software Engineering
University of Würzburg
http://se.informatik.uni-wuerzburg.de/

Kiel, 08/11/2016

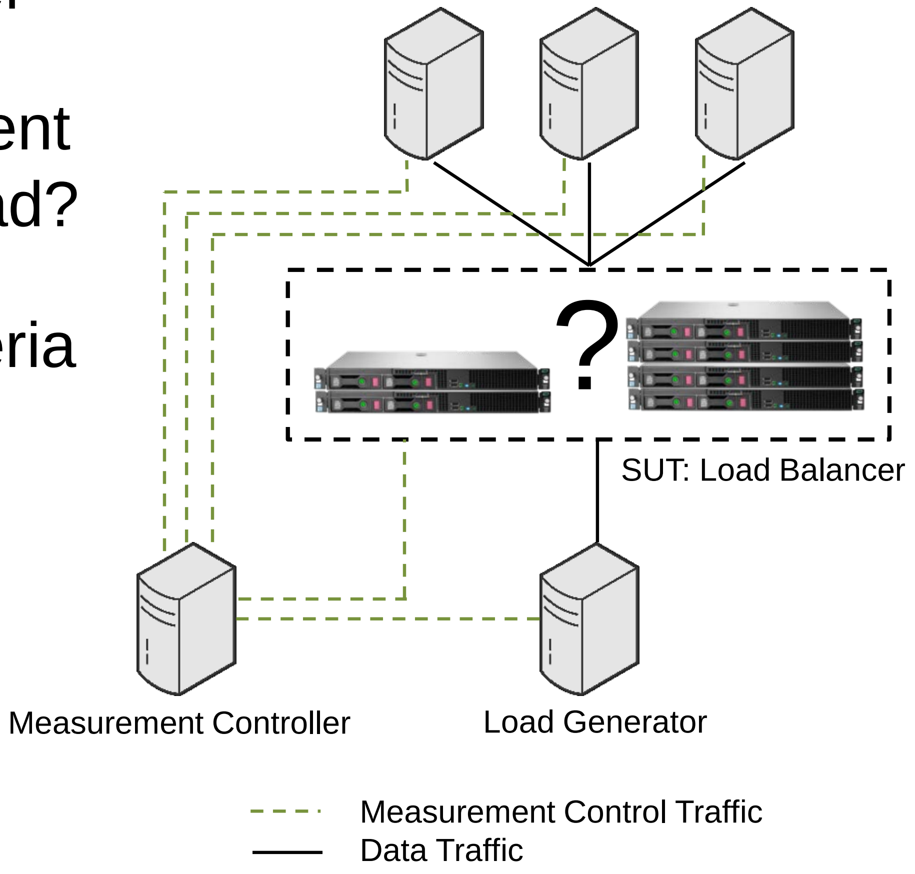
Motivation

- Increasing server energy consumption
- 61 billion kWh in 2006
- An estimated 140 billion kWh in 2020 [6,7]



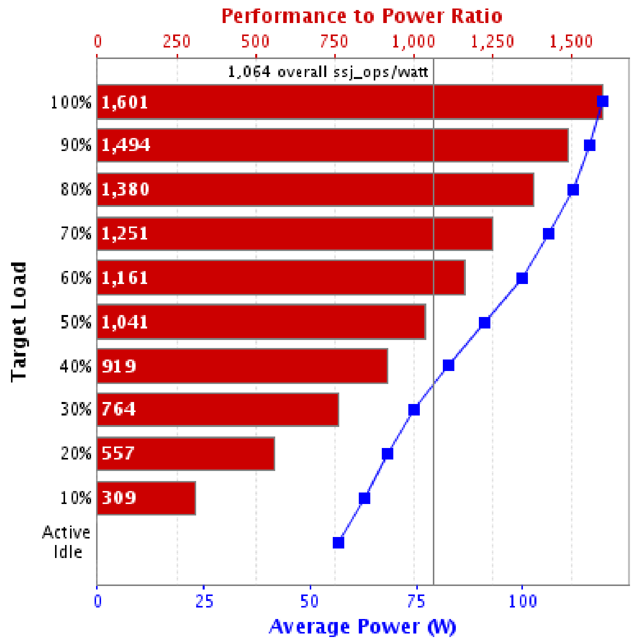
Motivation

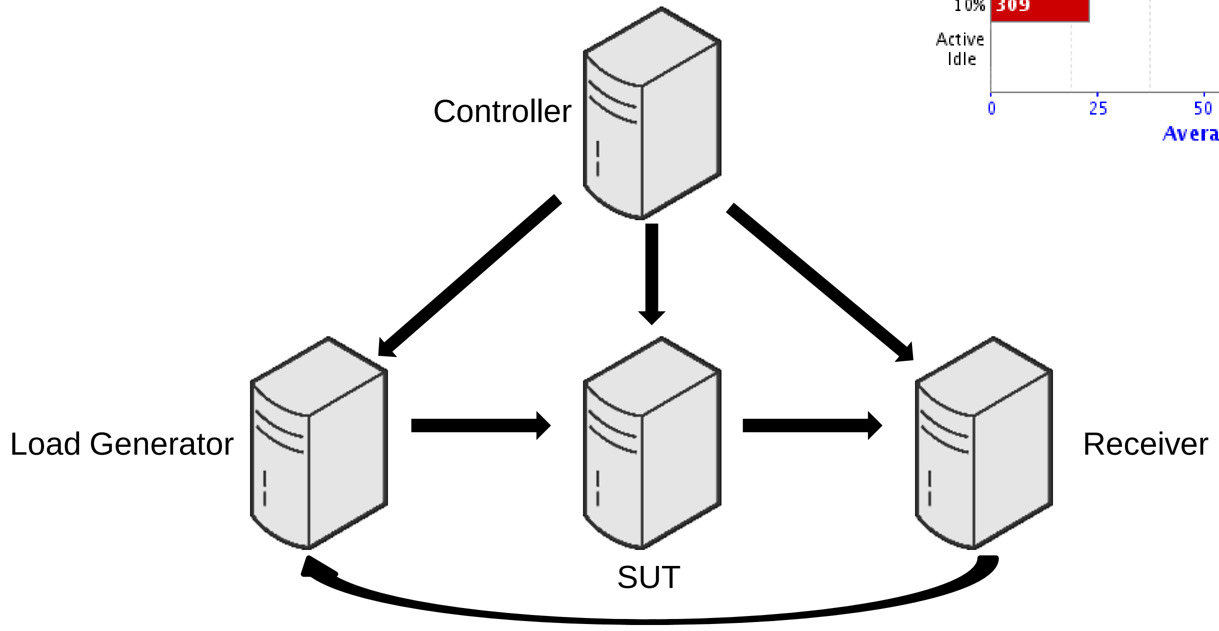
- Simulation of large networks and / or high data traffic to put externally driven workloads under load
  Example: Load Balancer
- Which is the most efficient machine for the workload?
- Huppler [8] defines criteria for a good benchmark:
  - Repeatable
  - Economical
  - …



Motivation

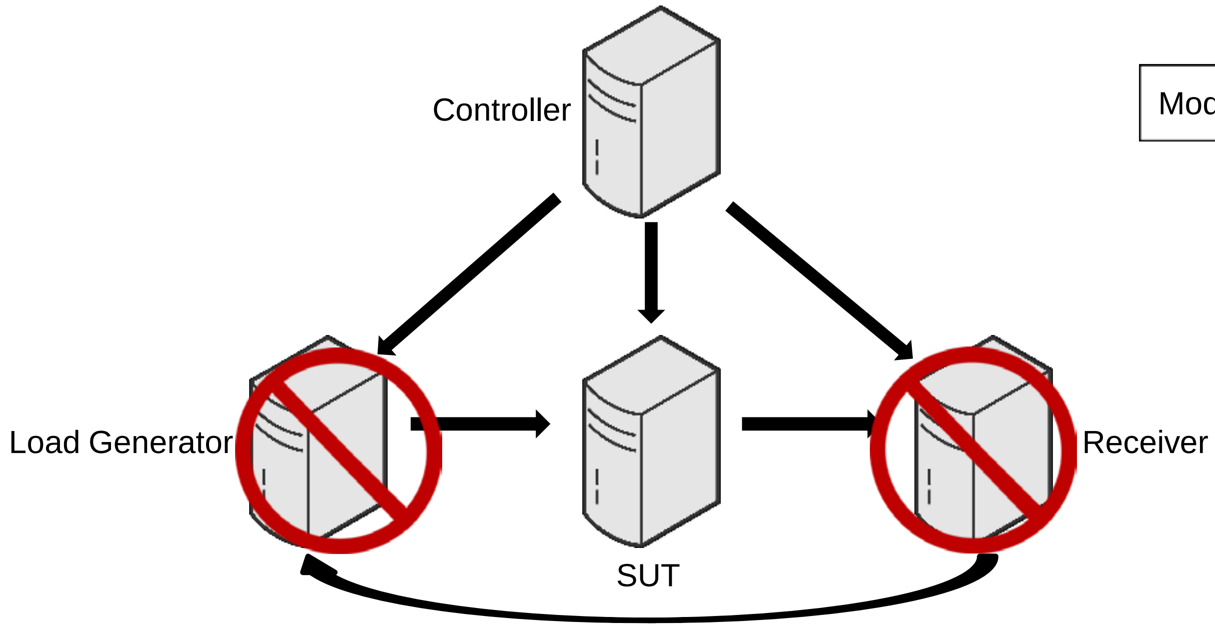
- Efficiency is measured under different load levels
- Complicated to calibrate and maintain load levels with external load generators due to latency

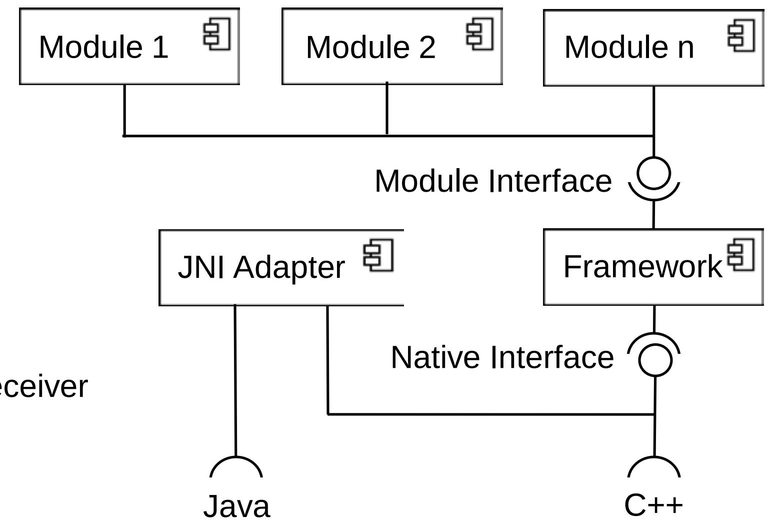




Motivation

- Approximate externally driven workloads on the SUT without the need for extra hardware
- Use Performance Counters for approximation
- Develop a modularized Performance Event Trigger Framework (PET) to approximate workloads

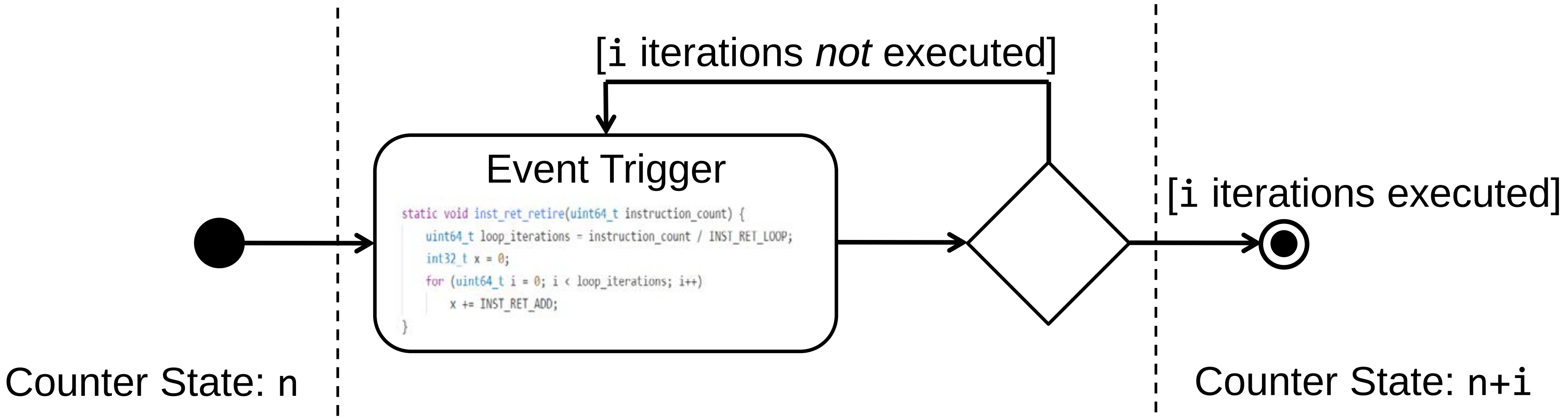




Approach – Event Trigger

- Performance Counter [1,2]
  - **Occurrence Events**
    How often has an event been observed
  - **Duration Events**
    Accumulated clock cycles for which an event has been observed

- Event Trigger
  - Stand-Alone implementation to cause `i` counted events



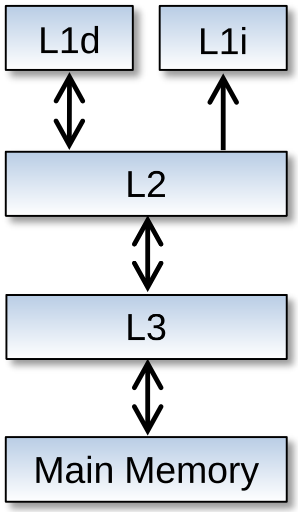
```
static void inst_ret_retire(uint64_t instruction_count) {
    uint64_t loop_iterations = instruction_count / INST_RET_LOOP;
    int32_t x = 0;
    for (uint64_t i = 0; i < loop_iterations; i++)
        x += INST_RET_ADD;
}
```

Approach – Side Effects

- Some Performance Counters cannot be modified without affecting other Counter Values
- They can be imposed either by hardware constraints or the implementation of an event trigger

Events counted:

✗	L1d miss event
✗	L2 miss event
✗	L3 miss event
✓	Read byte event



Example: Trigger event „Read byte from memory controller“ (Accessing main memory)

Approach - Composition

- Different implementations to incorporate side effects
  1. Naive: Neglect side effects
  2. Accumulation: Sum over all side effects $\boldsymbol{s}_i$ caused by triggering a number of events v_i
     $$\boldsymbol{s}_{total} = \sum_{i=1}^{n} \boldsymbol{s}_i * v_i$$
  3. Simulated Annealing: A numerical solution between the imposed side effects and event triggers

Approach – Evaluate Event Triggers

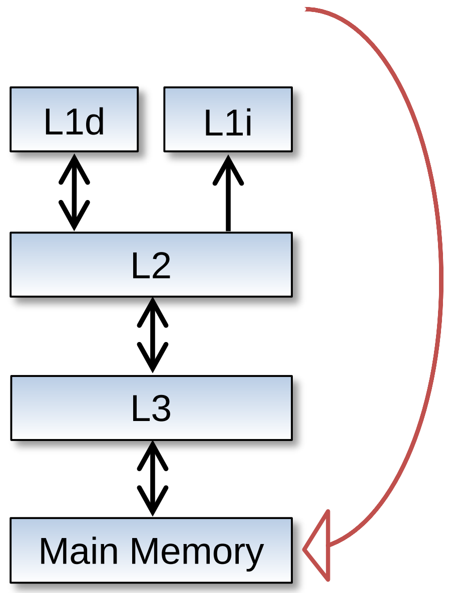
- Run the event trigger as a single process and in parallel as workloads can and do use multithreading
  - Single process
    → If it does not work in single process, the implementation might be erroneous
  - 4/8 processes
    → Number of physical/logical CPUs to determine if the implementation does scale in a multithreaded environment
- Each event trigger is set a reference value, a target, it has to reach
  - The lower the deviation from the target value the better is the implementation of the event trigger

Event Triggers – L3 miss

- Use the different caching modes supported by the CPU to prohibit caching → Automatically miss L3

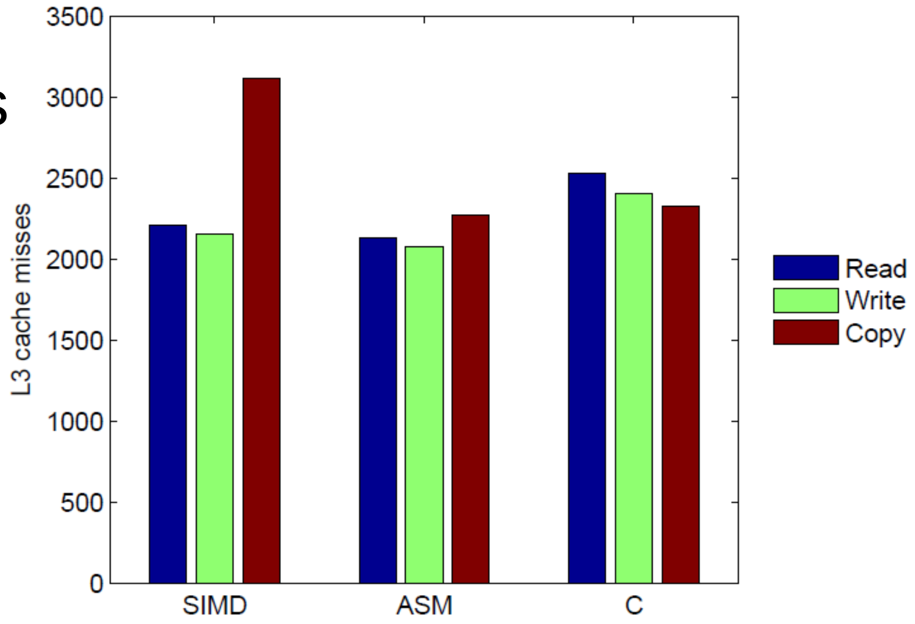
- Strong Uncachable (UC)
  - Set by Memory Type Range Register (MTRR)
  - Linux Kernel documentation discourages the use of MTRR [4]

- Uncachable Minus (UC-)
  - Set by Page Attribute Table (PAT)
  - Function in Linux Kernel available



Event Triggers – L3 miss

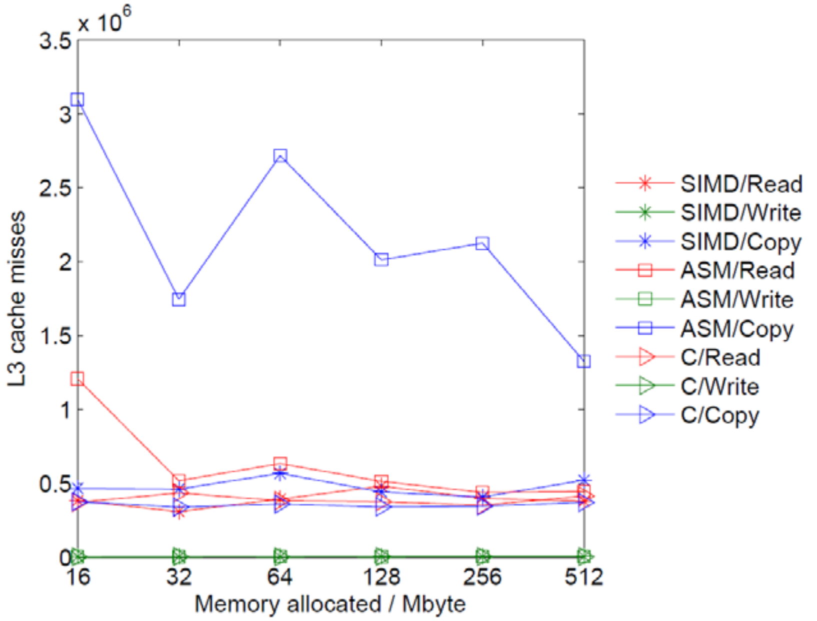
- Uncachable memory can be `mmap`-ed to user space
- Target value of $8 * 10^6$ L3 misses for 8 processes not reached
- Even worse for 1 and 4 processes
- No L3 misses are actually triggered



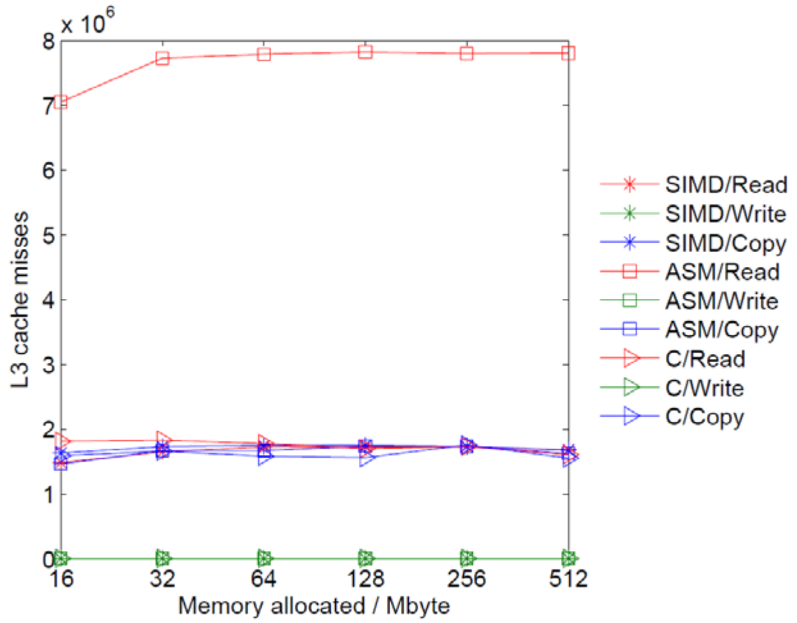
L3 cache misses for 8 processes on uncachable memory

Event Triggers – L3 miss

- Traverse a large array of at least twice the cache size
- **Problem**: Hardware prefetching loading data we do not want in cache [3]
- **Solution**: Increase stride to 6 times the cache line size with a deviation of -2.4% when 8 processes are running



L3 misses with stride 2



L3 misses with stride 6

Event Triggers – L2 miss / L3 hit

- Constraints for triggering L2 misses hitting L3:
  - Instead of traversing a large array, it must be small enough to fit inside L3 not to produce L3 misses on accident
  - The array must also be large enough for long strides to not access data already prefetched and generating a L2 hit

- Try to “confuse” the prefetcher by adding a random factor r after a stride s

$$p_n = p_{n-1} + s + r$$

Event Triggers – L2 miss / L3 hit

- L2 miss / L3 hits scale well to four processes

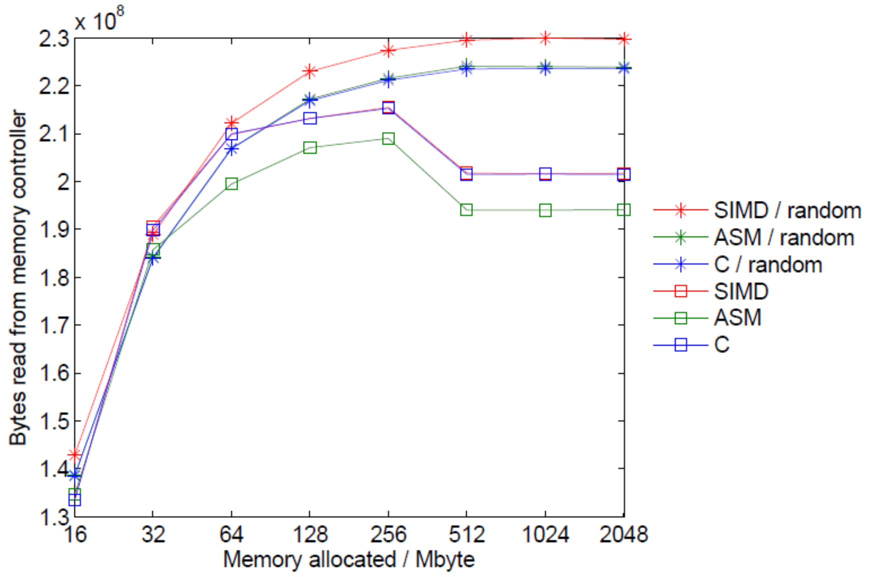
Processes	Instruction Set	Result (w/o random)	Result (w/ random)	Deviation
1	SIMD	165,031	923,975	−7.60%
	Assembler	204,282	966,181	−3.38%
	C	213,862	971,336	−2.87%
4	SIMD	443,037	3,391,786	−15.21%
	Assembler	1,225,584	3,408,932	−14.78%
	C	743,568	3,537,446	−11.56%
8	SIMD	2,284,193	5,253,885	−34.33%
	Assembler	1,747,484	4,047,238	−50.59%
	C	1,516,549	4,199,528	−47.51%

L2 misses / L3 hits generated; Targets: $1 * 10^6$, $4 * 10^6$ and $8 * 10^6$

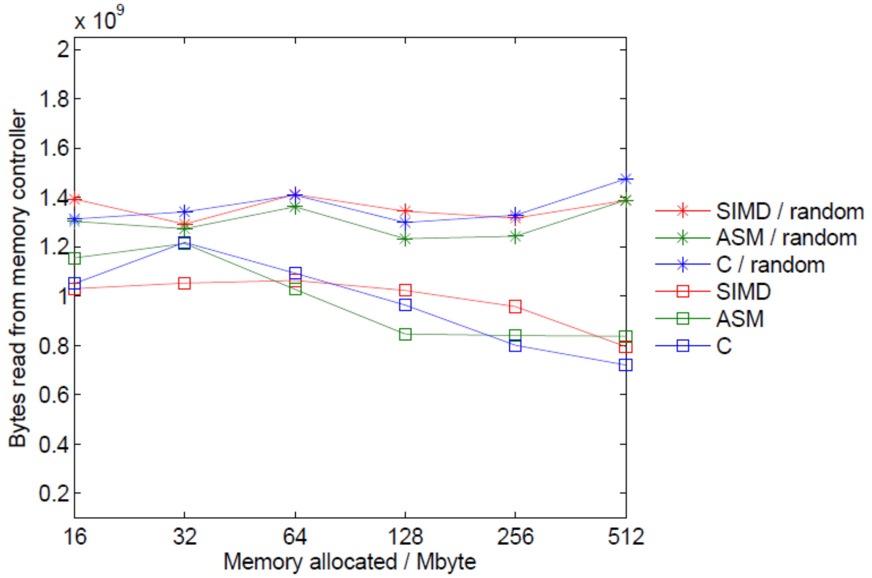
- Generating L2 misses with virtual CPUs provides no benefits. Triggering L2 misses only scales to the number of physical CPUs

Event Triggers – Read byte from MC

- Read bytes implemented in the same way as L3 misses
- Overcounting by a large margin
  Target for 1 process: $0.64 * 10^8$
  Target for 8 processes: $0.512 * 10^9$



Bytes read using 1 process and stride 6



Bytes read using 8 processes and stride 6

Event Triggers – Read byte from MC

- Use the kernel module to circumvent the caches

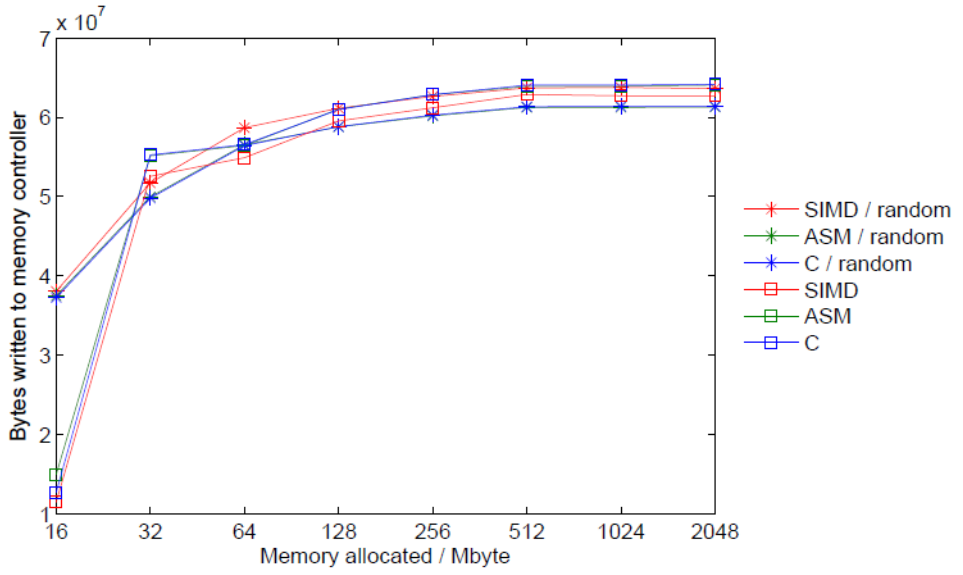
Processes	Instruction Set	Result	Deviation
1	SIMD	$64.003 * 10^6$	$0{,}004\%$
	Assembler	$64.038 * 10^6$	$0{,}060\%$
	C	$64.025 * 10^6$	$0{,}039\%$
4	SIMD	$242.41 * 10^6$	-5.31%
	Assembler	$255.98 * 10^6$	-0.01%
	C	$191.16 * 10^6$	-25.33%
8	SIMD	$326.43 * 10^6$	-36.24%
	Assembler	$343.67 * 10^6$	-32.88%
	C	$347.82 * 10^6$	-32.07%

Bytes read with uncachable memory
Targets: $64 * 10^6$, $256 * 10^6$ and $512 * 10^6$

- Read bytes from memory controller scales well to the number of physical CPUs for the SIMD and Assembler instruction sets

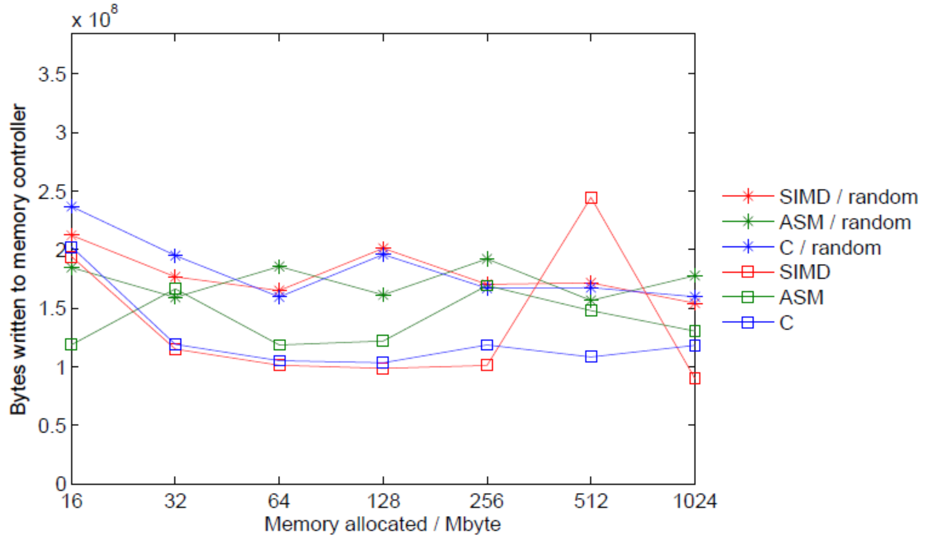
Event Triggers – Write byte to MC

- Uncachable memory worked well for bytes read so intuitively it should work when writing bytes
- Underestimating bytes written for all instruction sets ranging from -47.58% to -73.51%
- Do not use uncachable memory when triggering bytes written
- SIMD and ASM reach deviations of -0.03% and -0.01%

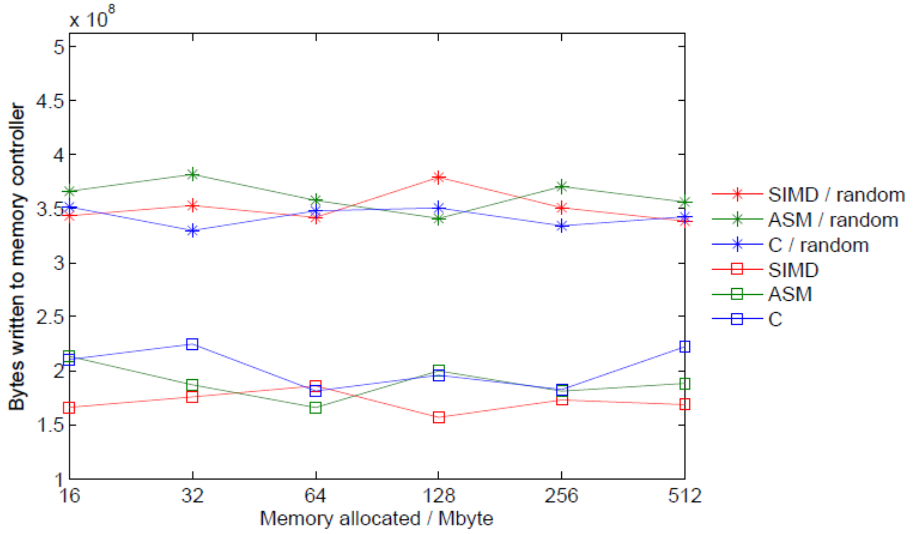


Bytes written for a single process and a stride of 6
Target: $6.4 * 10^7$

Event Triggers – Write byte to MC



Bytes written using 4 processes and stride 6
Target: $2.56 * 10^8$



Bytes written using 8 processes and stride 6
Target: $5.12 * 10^8$

- The event trigger is struggling to reach its target value if multiple processes are used

Event Triggers – Context switch

- Create C++11 threads that can be joined instantly
  - Intuitively, each thread should cause two switches
    → Half the amount of event triggers
- Introducing a linear factor of 0.5
  - Unexpected major deviations
  - Removing the factor results in large overcounting
- But a linear factor can still improve the accuracy

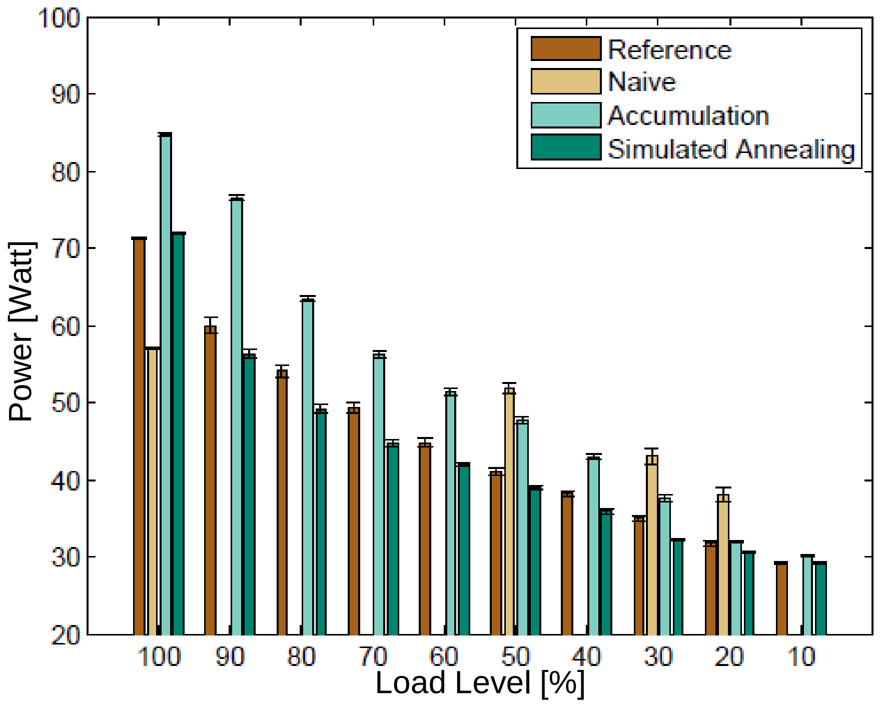
	Factor 0.5		Factor 0.8		Factor 1.0	
Processes	Result	Deviation	Result	Deviation	Result	Deviation
1	70,350	−29.7%	100,588	0.6%	120,641	20.6%
4	271,683	−32.1%	400,689	0.2%	481,324	20.3%
8	470,265	−41.2%	757,056	−5.4%	940,653	17.6%

Context switches triggered
Targets: $1 * 10^5$, $4 * 10^5$ and $8 * 10^5$

Approximating Workloads

Workload	Measurement	Mean	Max	CV
SSJ	Naive	12.35%	26.44%	19.37%
	Accumulation	13.28%	27.61%	7.03%
	Simulated Annealing	**−5.25%**	**−9.35%**	**3.66%**

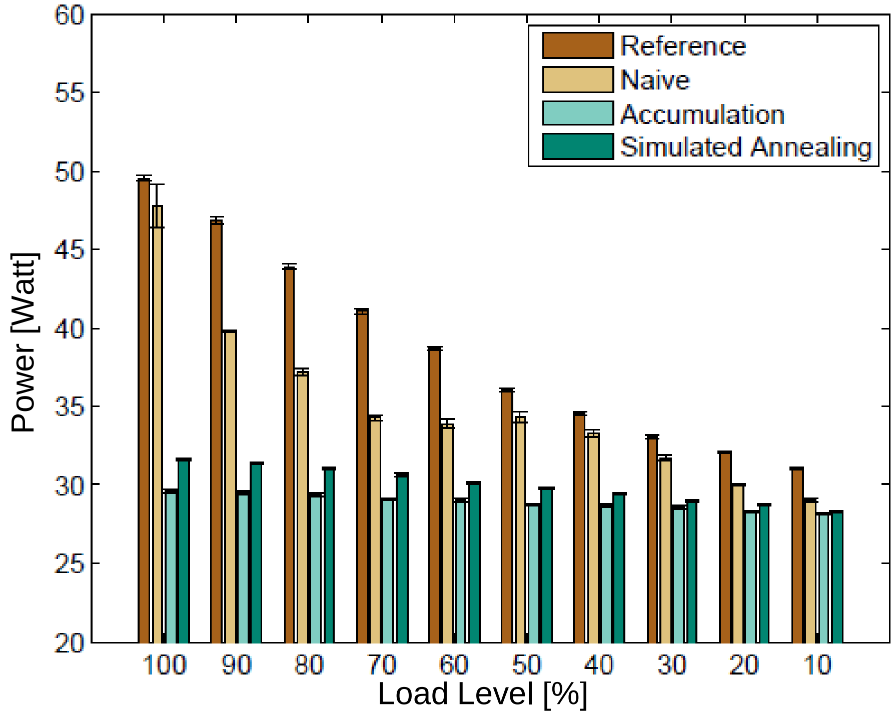
- Naive measurement with side effects has a low throughput due to long runtimes
- Accumulation still overestimates power consumption despite removing side effects



Approximating Workloads

Workload	Measurement	Mean	Max	CV
DPI Firewall	Naive	**−8.84%**	**−16.47%**	**5.86%**
	Accumulation	−23.68%	−40.23%	14.33%
	Simulated Annealing	−21.00%	−36.19%	12.32%

- Externally driven workloads can be approximated
- Underestimation expected due to the NIC not stressed in the approximation



Conclusion

- Know your hardware to avoid unwanted effects on the events to trigger
- Simultaneous Multithreading (SMT) is in most cases not beneficial when triggering performance events on purpose
- Intuition can be misleading and counterproductive
- Externally driven workloads can be approximated with reasonable accuracy
- Complex testbed setups can be simplified for faster and easier deployment → The PET framework reaches an average accuracy from below 10% down to 1%

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Thank You!

norbert.schmitt@uni-wuerzburg.de

References

1. AMD, *AMD64 Architecture Programmer‘s Manual Volumen 2: System Programming*, Advanced Micro Devices Inc., April 2016. http://support.amd.com/TechDocs/24593.pdf
2. Intel, *Intel® 64 and IA-32 Architectures Software Developer‘s Manual*, Intel Corporation, June 2016. http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html
3. Intel, *Intel® 64 and IA-32 Architectures Optimization Reference Manual*, Intel Corporation, June 2016. http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf
4. R. Gooch and L. R. Rodriguez, „*MTRR (memory type range register) control*“, accessed: 2016-08-15. http://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/Documentation/x86/mtrr.txt
5. GNU, „*GCC Common Function Attributes*“, accessed: 2016-11-04 https://gcc.gnu.org/onlinedocs/gcc/Common-Function-Attributes.html#Common-Function-Attributes
6. R. Brown et. al., „*Report to congress on server and data center energy efficiency: Public law 109-431*“, Lawrence Berkeley National Laboratory, Jun. 2008. http://eetd.lbl.gov/sites/all/les/pdf 4.pdf
7. J. Whitney and P. Delforge, “*Data center efficiency assessment*“, http://www.nrdc.org/energy/les/data-center-eciency-assessment-IP.pdf, Aug. 2014.
8. K. Huppler, „The Art of Building a Good Benchmark“, IBM Corporation, 3605 Highway 52 North, Rochester, MN 55901, USA 2009.