Quick-and-Dirty Memory Access Tracing with Instruction-Based Sampling[*]

Lukas Wenzel[1], Sven Köhler[1], Henriette Hofmeier[2], Felix Eberhardt[1]
[1]Hasso Plattner Institute, Potsdam [2]Ruhr University Bochum
{firstname.lastname}@{hpi,rub}.de

ABSTRACT

With the increasing deployment of heterogeneous memory architectures, the efficient execution of a workload becomes more sensitive to fine-grained memory placement decisions. To establish a sound information base for such decisions, we must first understand memory access behavior beyond the level of coarse-grained statistics. However, collecting detailed memory traces is a costly process. Therefore, we propose a low-overhead solution based on instruction-based sampling [3] that provides incomplete yet informative access sequences. We showcase the practical value of such sparse traces by analyzing the overhead and comparing workload runs on two memory technologies with distinct characteristics.

Concept. There is an extensive body of prior work on obtaining detailed memory traces: Running the workload in a full-system simulator [1] provides accurate and detailed information about the progress of memory instructions, at the cost of orders of magnitude longer experiment durations, though no behavioral distortion due to the application running in simulated time. Alternative techniques rely on binary instrumentation [2], where accesses are tracked by augmenting each relevant instruction with a call to a logging function. Because the workload runs natively on hardware, traces are obtained at a lower cost compared to full system simulation, although the added tracing overhead can distort workload behavior.

For our envisioned purposes, the aforementioned techniques' focus on completeness is not required, opening different tradeoffs in terms of overhead. We leverage the extensive performance measurement infrastructure present in most hardware platforms to collect only a subset of all memory accesses issued by the workload. The underlying mechanism is instruction-based sampling, which is generally used to identify sections of program code with a specific performance impact. A hardware performance counter is set up to track events that characterize this impact, and a threshold is set that, when crossed, triggers an interrupt saving the architectural state at the time of the last event. Usually, the instruction pointer is taken from this state to derive a statistical distribution of the instructions responsible for the event, but for memory tracing purposes, the data address related to load or store events is recorded instead. The threshold controls the proportion of memory accesses that can be observed, with the assumption that frequently accessed addresses have a higher probability of appearing in the final sparse trace.

In addition to the access trace, we are interested in the virtual memory layout, to associate accesses with their semantic memory region and thus distinguish unrelated access patterns. Under the goal of memory placement, dynamic heap allocations play an important role and need to be traced as well. This is possible by intercepting the library calls necessarily occurring every state change.

Showcase. To demonstrate the properties of the proposed technique, we collect sparse traces for the five kernels and three pseudo applications of the NAS Parallel Benchmarks suite (NPB) version 3.4.1 as an exemplary workload. Observing relevant differences in the collected traces between heterogeneous memory resources showcases the technique's value. In future work, we will use these traces to optimize memory placement decisions.

We conducted our experiments on an IBM PowerSystem S924 in a dual-socket NUMA configuration running Linux. This system features an IBM / BittWare Hybrid Memory Subsystem (HMS) card, which attaches NVMe storage through a DRAM cache to the physical memory space via an OpenCAPI 3.0 link. HMS memory accesses can have much longer and asymmetric read/write latencies compared to regular system DRAM. Performance characteristics range from 170 GB/s at 65 ns latency for local NUMA accesses to 20 GB/s over OpenCAPI with microsecond NVMe latencies.

We have implemented an interposer library that is applied to the workload binary via the `LD_PRELOAD` mechanism, which tracks heap allocations and implements a common first-fit allocation algorithm between anonymously mapped DRAM and explicitly mapped HMS memory for comparable results. This suffices for the simple behavior of the NPB suite (all allocations up front), but future work can incorporate *libmemkind*, which offers more sophisticated placement and allocation facilities. Access traces are collected using the `perf` tool, which wraps both workload and interposer library and configures the instruction-based sampling infrastructure.

We executed various memory and instrumentation configurations four repetitions each and analyzed with the `time` tool both with and without `perf` instrumentation, to gauge apparent processor and wall-clock time overheads. The geometric mean of instrumentation slowdowns over all measured benchmarks was 6.0 % processor time and introduced 30.5 % wall-clock time overhead at maximum sampling precision and a rate of 250 Hz. While not acceptable in all application scenarios, we believe these slowdowns qualify the technique for online workload observation and placement decision refinement.

The obtained series of traces—as shown in the poster—clearly indicates varying patterns of memory traffic over different allocations, execution phases and memory resources and form an invaluable basis for our future work on memory placement strategies.

ACKNOWLEDGEMENTS

This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 502228341 ("Memento"). Hardware for the experiments was kindly provided by the IBM Research & Development Lab, Böblingen.

[*] This poster abstract was accepted at the 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI'23), Poster Session.

REFERENCES

[1] Eduardo Henrique Molina da Cruz et al. 2011. Using memory access traces to map threads and data on hierarchical multi-core platforms. In *Proc. of the 2011 IEEE Int. Symp. on Parallel and Distributed Processing Workshops and Phd Forum.*

[2] Mathias Payer, Enrico Kravina, and Thomas R Gross. 2013. Lightweight memory tracing. In *Proc. of the 2013 USENIX Annual Technical Conference (USENIX ATC'13).*

[3] Vincent M Weaver. 2016. Advanced hardware profiling and sampling (PEBS, IBS, etc.): creating a new PAPI sampling interface. *Technical Report UMAINE-VMWTR-PEBS-IBS-SAMPLING-2016-08. University of Maine, Tech. Rep.* (2016).

Quick-and-Dirty Memory Access Tracing with Instruction-Based Sampling

Lukas Wenzel[1], Sven Köhler[1], Henriette Hofmeier[2], Felix Eberhardt[1]

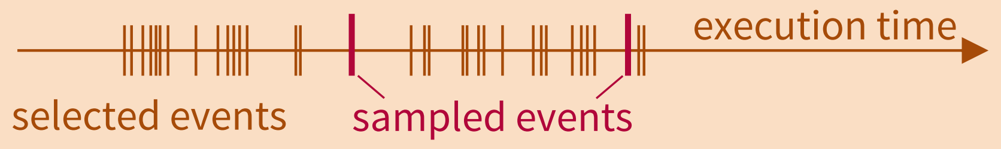
[1] Hasso Plattner Institute, Potsdam [2] Ruhr University Bochum

Motivation

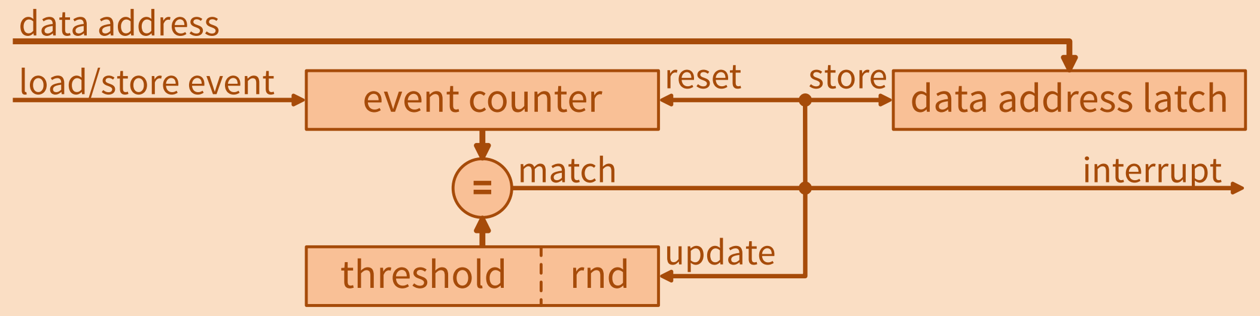
- Despite a homogenous address space, heterogeneous memory resources have substantially varying characteristics
- Applications often fail to leverage different memory resources efficiently
- We propose low-overhead traces to understand the dynamics of memory allocation and access to understand performance and energy impact

Concept and Approach

- Instruction-Based Sampling records architectural state at randomized intervals of selected hardware events [3]



- We record accessed data addresses, sampling over memory read and write instruction events



- Trigger threshold controls proportion of observed memory accesses, with frequently accessed locations more likely to appear

Capturing Allocations

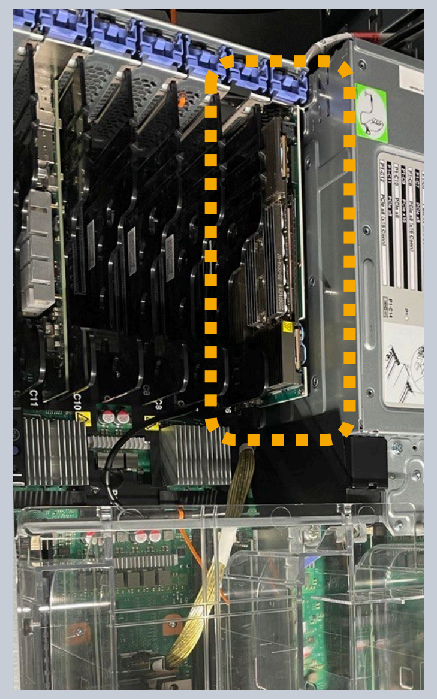
- To understand workload dynamics and semantic relations in the address space, we associate memory accesses to allocated buffers
- We wrote `tracealloc`, a prototypical interposer library that is injected via `LD_PRELOAD`
- It intercepts and logs `malloc()`, `free()`, `calloc()`, `realloc()`, `memalign()`, …

In Relation To Alternative Approaches

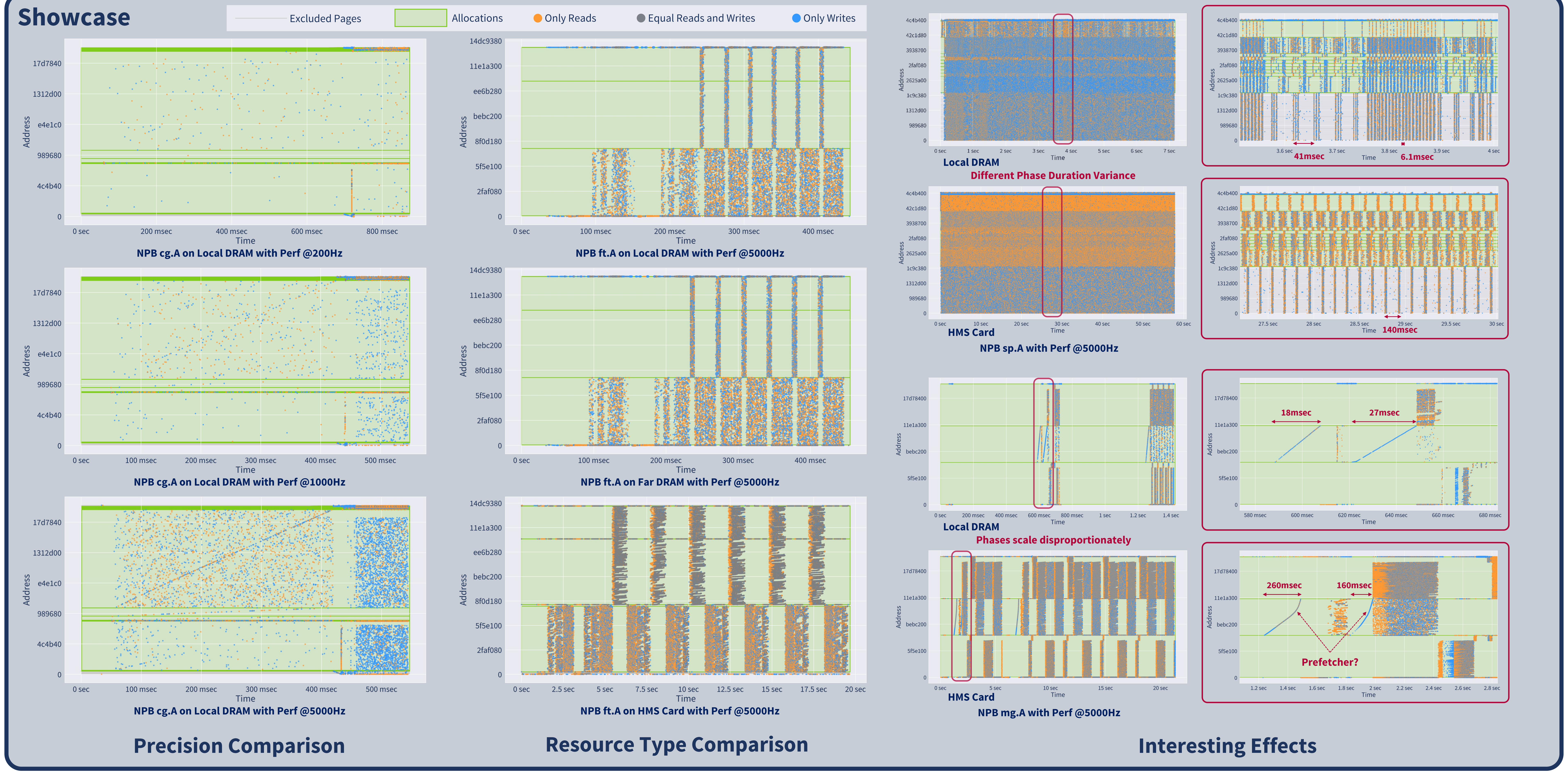
Software Only [2]	PMC Support	Custom Hardware Instrumentation	Hardware Simulation [1]
▪ OS level: *mprotect+ptrace* ▪ Instrumentation *valgrind* ▪ Virtual Machines *QEMU patching*	▪ IBS with Linux' `perf` interface	▪ FPGA-based memory controller tracking accesses	▪ System-level or processor simulator *gem5*
▪ Full coverage ▪ High overhead ▪ Distortion according to overhead	▪ *Sparse* coverage ▪ Low overhead ▪ Distortion according to overhead	▪ Full coverage ▪ Low overhead ▪ Trade higher overhead for no distortion (clock slowdown)	▪ Full coverage ▪ High overhead ▪ No distortion (simulated time)
	This poster		

Experimental Setup & Hardware

- IBM PowerSystem S924, dual socket NUMA (2666 MHz DDR4)
- IBM / BittWare Hybrid Memory Subsystem card (OpenCAPI 3.0 attached NVMe storage)
- NAS Parallel Benchmarks suite version 3.4.1 (OpenMP, C++)



Showcase



Overhead Evaluation

		Processor Time Overhead (Arithmetic Mean in Seconds)		Processor Time Slowdown (Geometric Mean minus 100%)		Wall-Clock Time Overhead (Arithmetic Mean in Seconds)		Wall-Clock Time Slowdown (Geometric Mean minus 100%)	
Local DRAM	**Perf @200Hz**	4.46	(-0.03 - 23.75)	6.0%	(-0.3% - 20.5%)	1.00	(0.53 - 1.37)	128.9%	(44.6% - 706.3%)
Local DRAM	**Perf @1000Hz**	3.96	(0.09 - 16.81)	8.3%	(1.0% - 29.6%)	0.99	(0.53 - 1.37)	128.8%	(44.6% - 706.3%)
Local DRAM	**Perf @5000Hz**	9.92	(0.18 - 44.43)	11.3%	(2.2% - 38.4%)	1.43	(0.78 - 2.38)	161.7%	(64.2% - 706.3%)
Far DRAM	**Perf @200Hz**	3.98	(-0.15 - 14.16)	5.1%	(-1.8% - 11.6%)	0.94	(0.54 - 1.32)	115.1%	(34.6% - 616.7%)
Far DRAM	**Perf @1000Hz**	4.17	(0.01 - 13.64)	6.6%	(0.1% - 22.5%)	1.01	(0.54 - 1.48)	118.5%	(44.4% - 616.7%)
Far DRAM	**Perf @5000Hz**	10.50	(0.15 - 35.20)	9.6%	(1.8% - 28.9%)	1.38	(0.54 - 2.49)	138.1%	(47.7% - 616.7%)
HMS Card	**Perf @200Hz**	1.48	(-0.40 - 6.81)	2.2%	(-3.7% - 6.0%)	0.90	(0.55 - 1.34)	101.5%	(35.1% - 597.3%)
HMS Card	**Perf @1000Hz**	2.35	(0.00 - 8.28)	4.1%	(-0.1% - 8.0%)	0.90	(0.55 - 1.34)	101.5%	(35.1% - 597.3%)
HMS Card	**Perf @5000Hz**	6.89	(-0.10 - 20.13)	7.1%	(-1.1% - 15.2%)	1.46	(0.72 - 2.35)	143.5%	(64.6% - 597.3%)
All Runs	**Perf @200Hz**	**3.30**	**(-0.40 - 23.75)**	**4.4%**	**(-3.7% - 20.5%)**	**0.94**	**(0.53 - 1.37)**	**114.9%**	**(34.6% - 706.3%)**
All Runs	**Perf @1000Hz**	**3.49**	**(0.00 - 16.81)**	**6.3%**	**(-0.1% - 29.6%)**	**0.97**	**(0.53 - 1.48)**	**116.0%**	**(35.1% - 706.3%)**
All Runs	**Perf @5000Hz**	**9.10**	**(-0.10 - 44.43)**	**9.3%**	**(-1.1% - 38.4%)**	**1.43**	**(0.54 - 2.49)**	**147.5%**	**(47.7% - 706.3%)**
All Runs		**5.30**	**(-0.40 - 44.43)**	**6.7%**	**(-3.7% - 38.4%)**	**1.11**	**(0.53 - 2.49)**	**125.6%**	**(34.6% - 706.3%)**

Every run was executed with **each NPB** (cg.A, ep.A, ft.A, is.A, mg.A, bt.A, lu.A, sp.A) and **8 repetitions**.
Values are **relative** to respective configuration **without perf instrumentation**.

References

1. Eduardo Henrique Molina da Cruz et al. 2011. Using memory access traces to map threads and data on hierarchical multi-core platforms. In *Proc. of the 2011 IEEE Int. Symp. on Parallel and Distributed Processing Workshops and Phd Forum.*
2. Mathias Payer, Enrico Kravina, and Thomas R Gross. 2013. Lightweight memory tracing. In *Proc. of the 2013 USENIX Annual Technical Conference (USENIX ATC'13).*
3. Vincent M Weaver. 2016. Advanced hardware profiling and sampling (PEBS, IBS, etc.): creating a new PAPI sampling interface. *Technical Report UMAINE-VMWTR- PEBS-IBS-SAMPLING-2016-08. University of Maine, Tech. Rep.* (2016).

Website
{first.last}@{hpi,rub}.de
https://osm.hpi.de/energy

Experiment hardware kindly provided by IBM Deutschland R&D Lab, Böblingen
SPP 2377
funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 502228341