`numap`: A Portable Library For Low Level Memory Profiling

Manuel Selva [1,2] Lionel Morel [1] Kevin Marquet [1]

[1]Universite de Lyon, Insa Lyon, Inria, CITI
[2]LIRMM, CNRS, Universite de Montpellier

July 20, 2016

Multi-core And The Memory Wall

DDR3-2133 SDRAM

Memory latency: **10.3 ns**

Memory bandwidth: **17.6 GB/s**

Processors

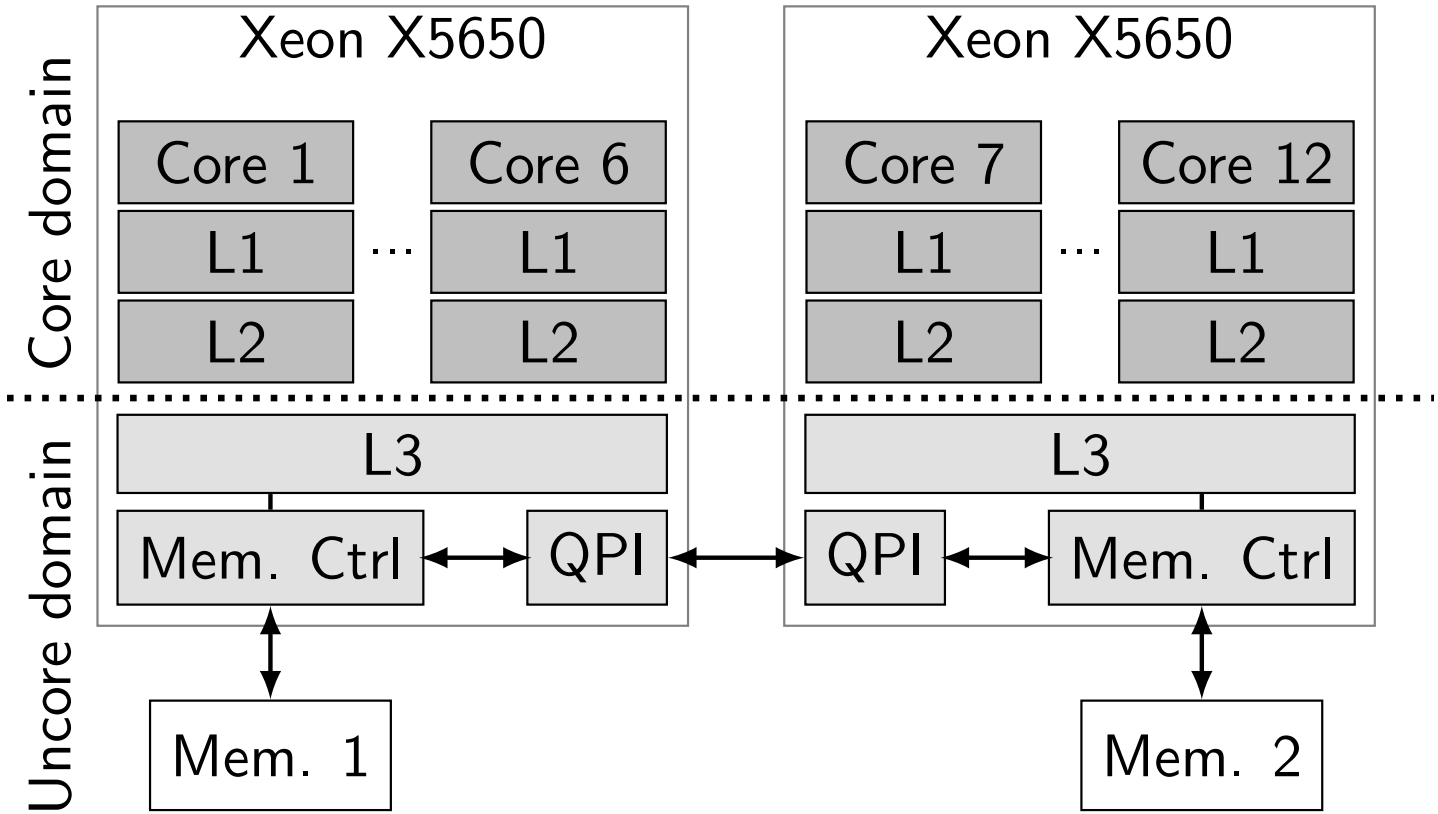
1 x 4-cores 2GHz ARM Cortex A15

Compute bandwidth: **256 GB/s**

2 x 6-cores 3GHz Intel Xeon X5650

Compute bandwidth: **2304 GB/s**

Complex Memory Architectures



Many cache levels

Distributed shared memory architectures

A.K.A. Non Uniform Memory Architectures - **NUMA**

Impact Of Memory Architecture On Software?

Transparent for correctness

- Single shared physical address space
- Memory consistency models

Not transparent for performances

- Cache hit ratio **variation**
- Memory controller **overload**
- Interconnect **overload**
- **Remote access latency** $>$ local access latency

Impact Of Memory Architecture On Software?

Transparent for correctness

Single shared physical address space

Memory consistency models

Not transparent for performances

Cache hit ratio **variation**

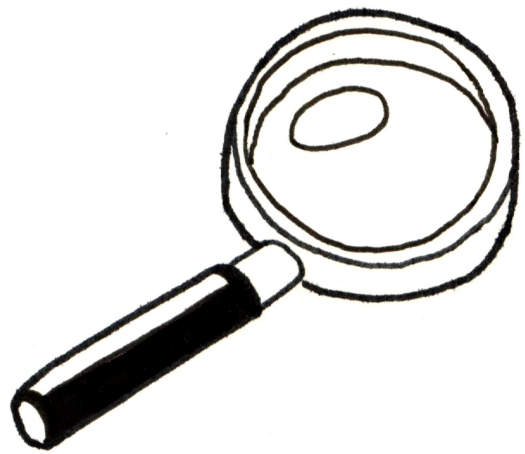
Memory controller **overload**

Interconnect **overload**

Remote access latency $>$ local access latency

Software (Programmer, Compiler, OS) **must** take memory architecture into account

Memory Profiling Is Needed



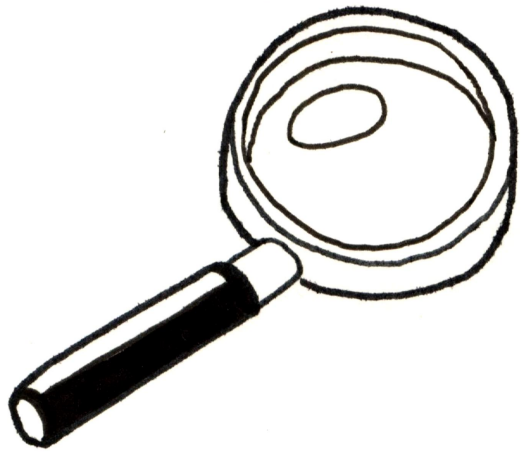
What is cache hit ratio?

Is the memory bandwidth **overloaded**?

Is the interconnect **overloaded**?

Where software makes remote memory accesses?

Memory Profiling Is Needed



Memphys - 2010
Memprof - 2012
Carrefour - 2013
HPCToolkit - 2014
ScaAnalyzer - 2015

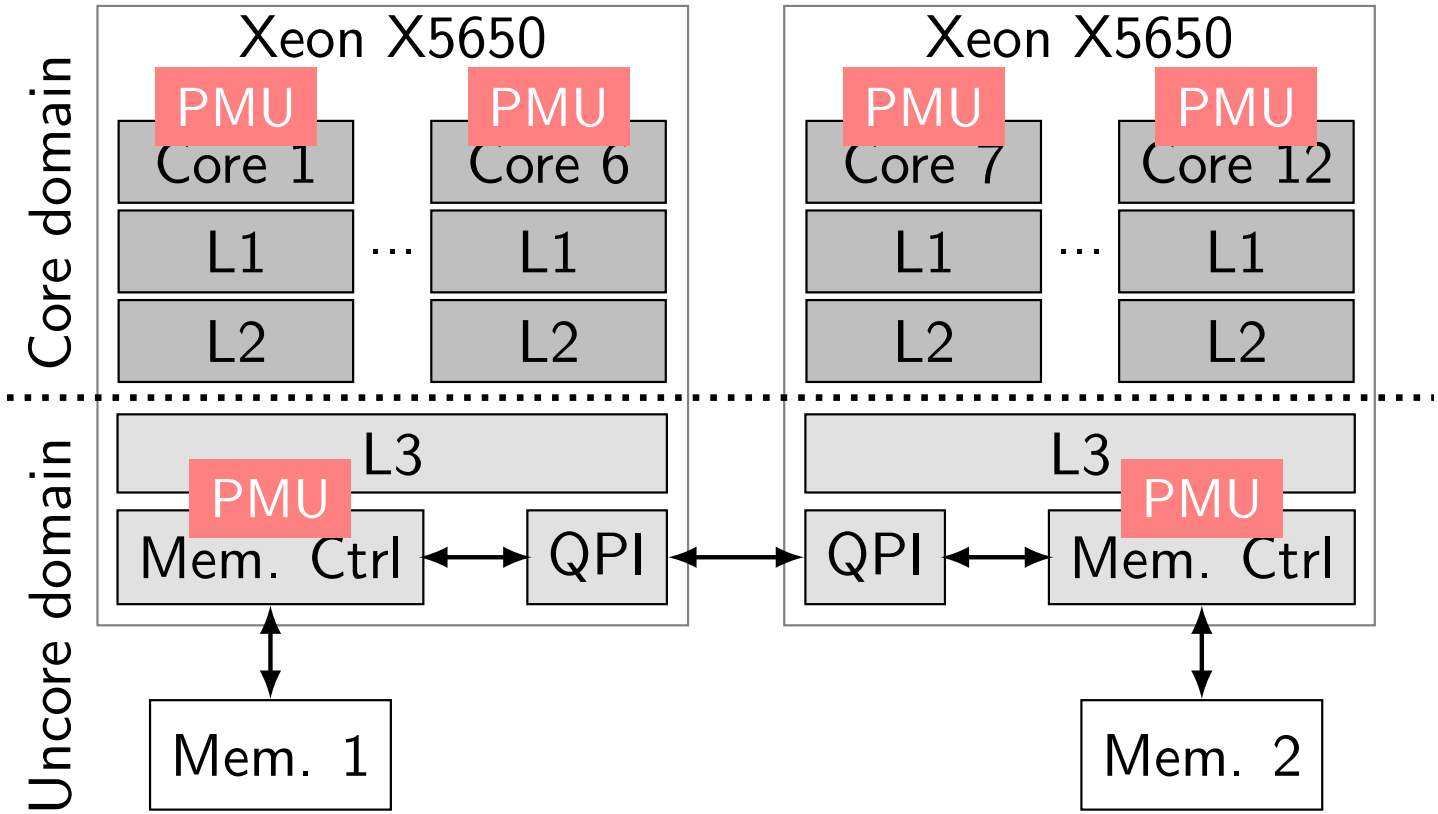
What is cache hit ratio?

Is the memory bandwidth **overloaded**?

Is the interconnect **overloaded**?

Where software makes remote memory accesses?

Performance Monitoring Unit (PMU)



Counting

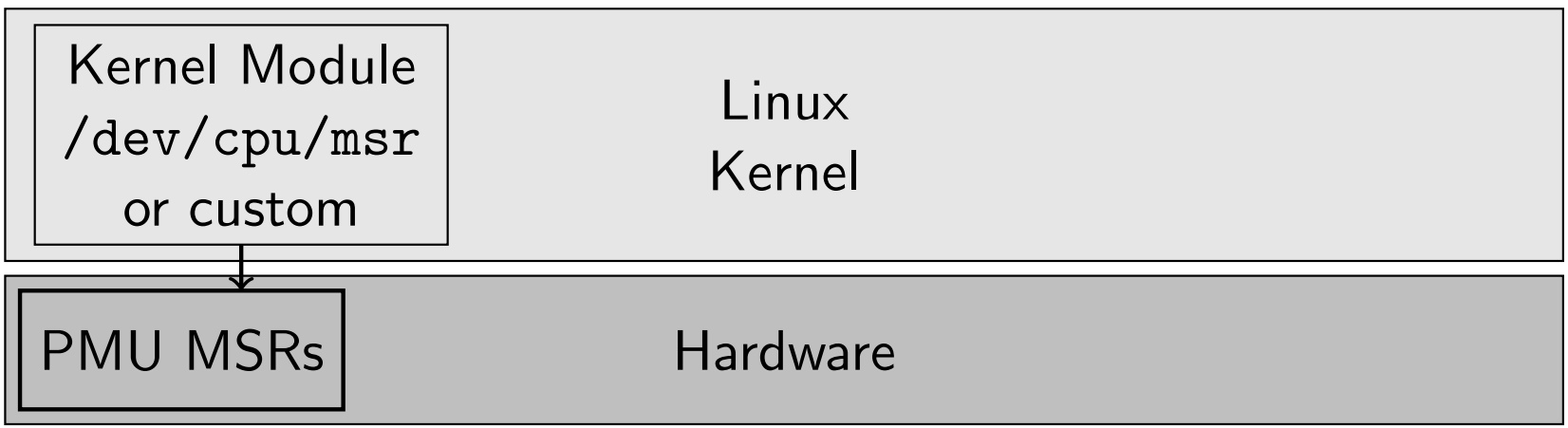
- L1/L2/L3 hits/misses
- Retired instructions
- Branch mispredictions

Sampling

- Retired instructions, memory accesses
- PMU collects extra information when event occurs

How To Use The PMU

How To Use The PMU



How To Use The PMU

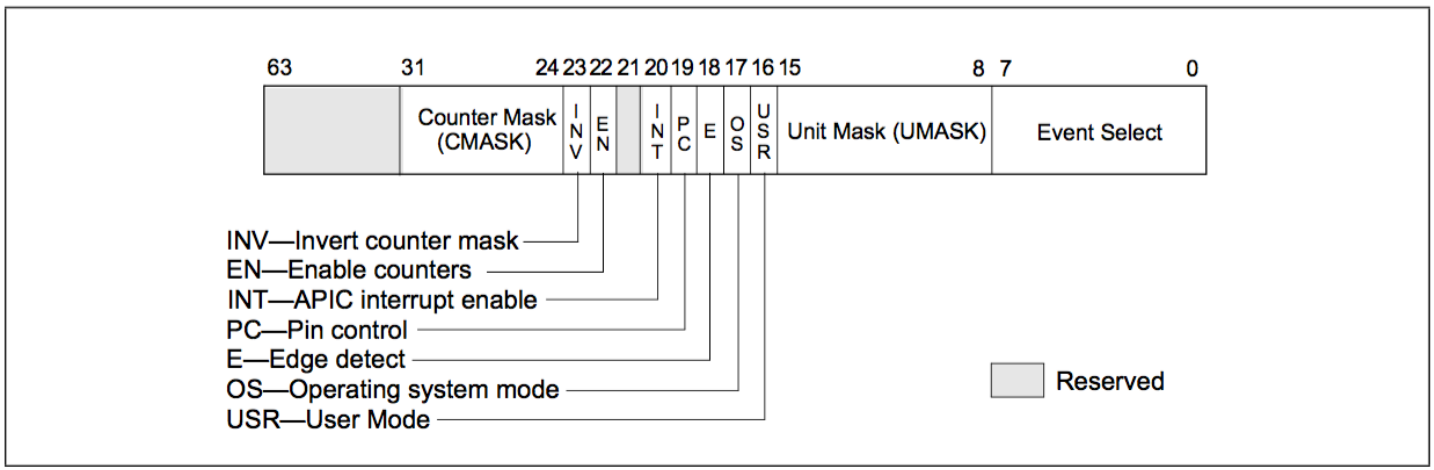
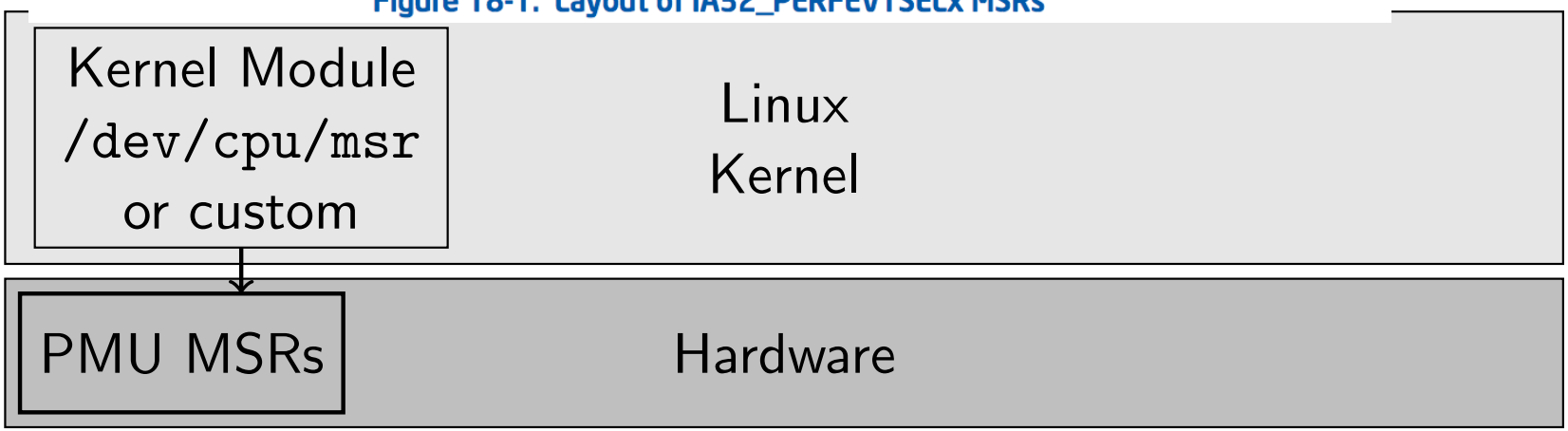
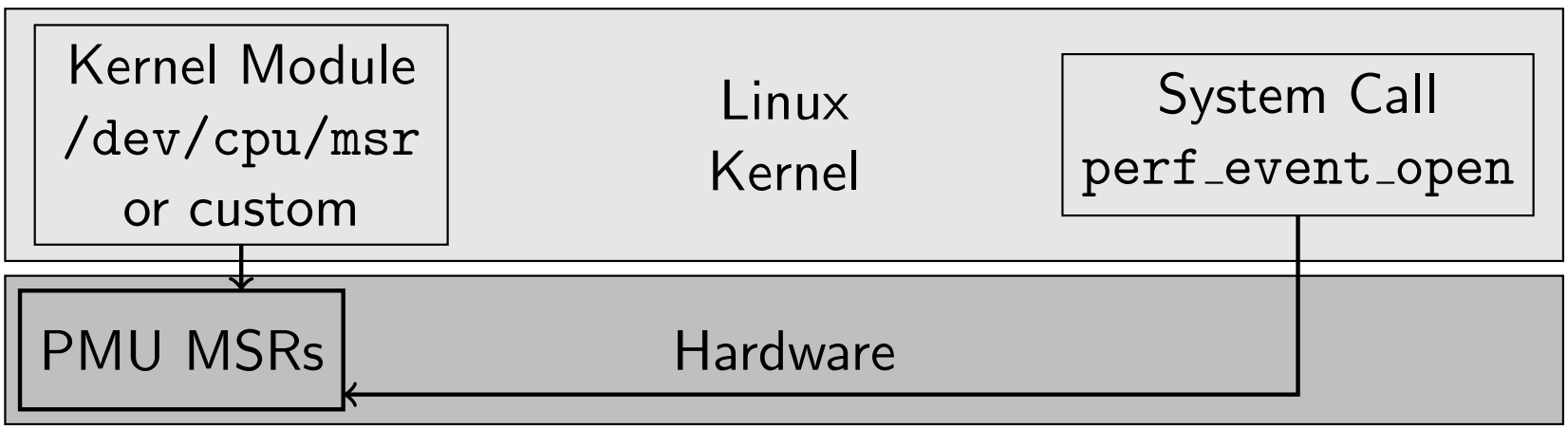


Figure 18-1. Layout of IA32_PERFEVTSELx MSRs

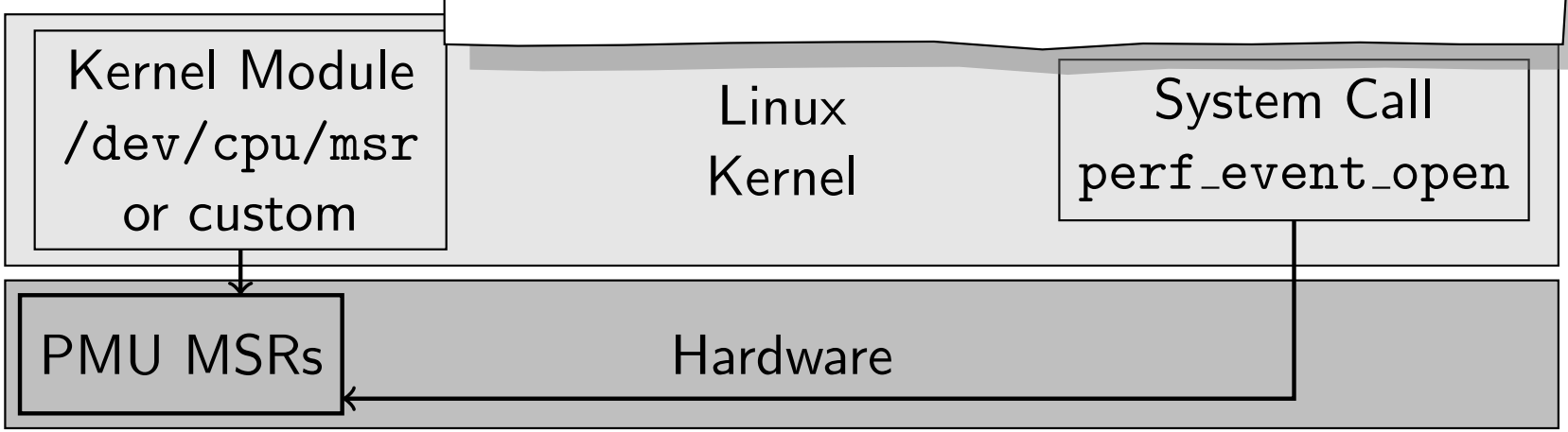


How To Use The PMU

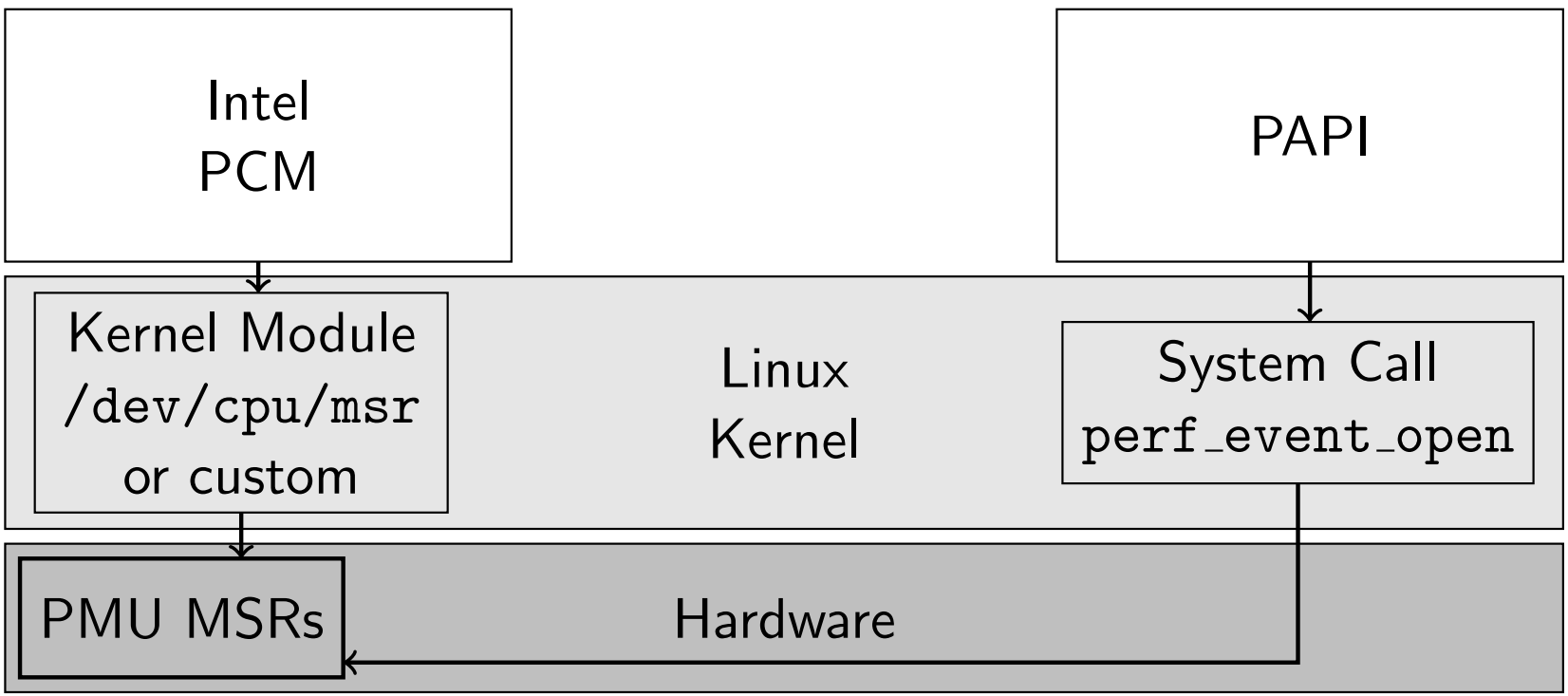


How To Use The PMU

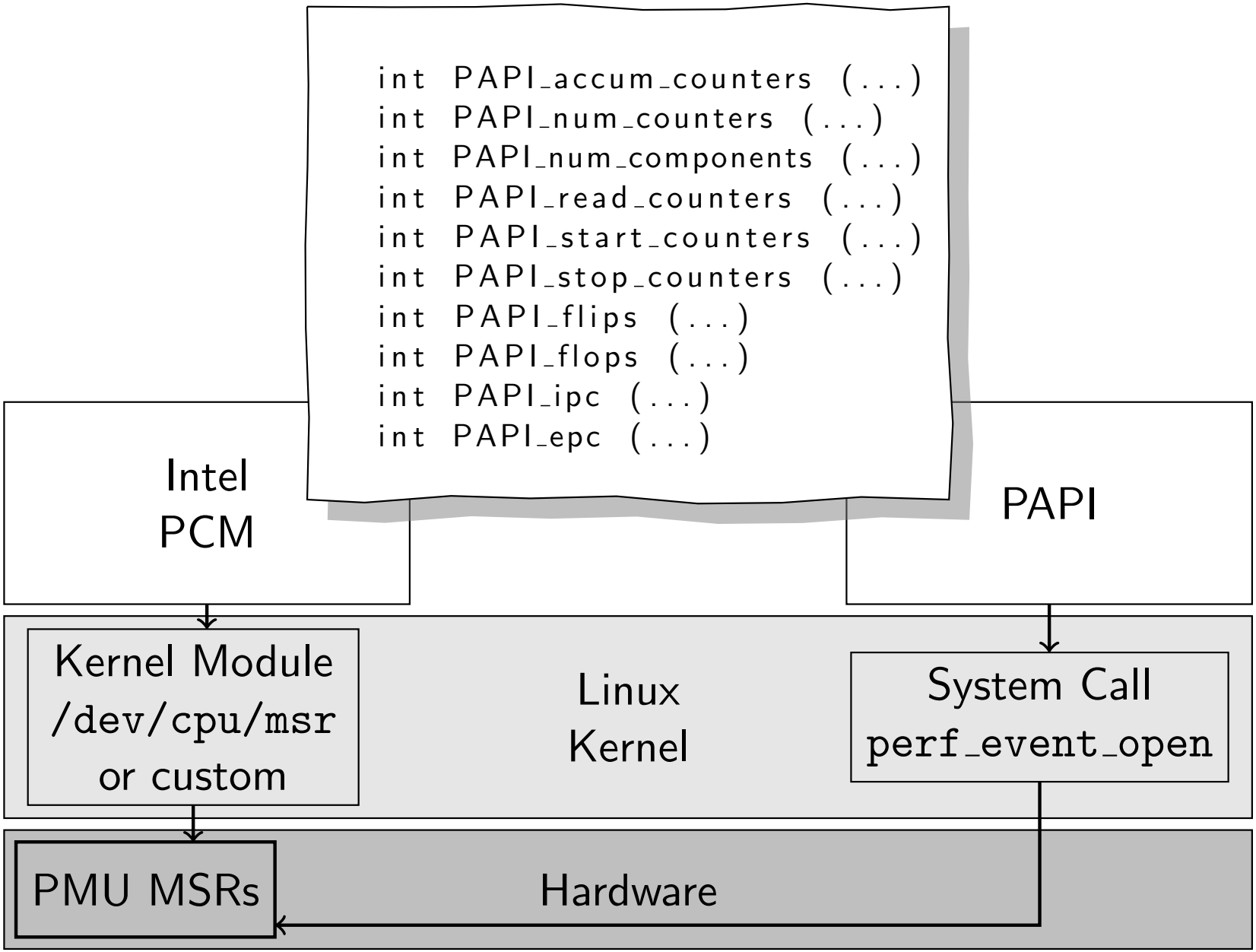
```
// Set system call parameters
struct perf_event_attr pe_attr;
attr.config = 0x100b;
attr.config1 = 3;
attr.sample_period = 20000;
attr.sample_type = PERF_SAMPLE_IP
                 | PERF_SAMPLE_ADDR
                 | PERF_SAMPLE_WEIGHT
                 | PERF_SAMPLE_DATA_SRC;
attr.precise_ip = 2;
attr.mmap = 1;
attr.task = 1;
attr.exclude_kernel = 1;
attr.exclude_hv = 1;
attr.disabled = 1;

// Make the call
int fd = perf_event_open(&attr, TID, -1, -1, 0);
```



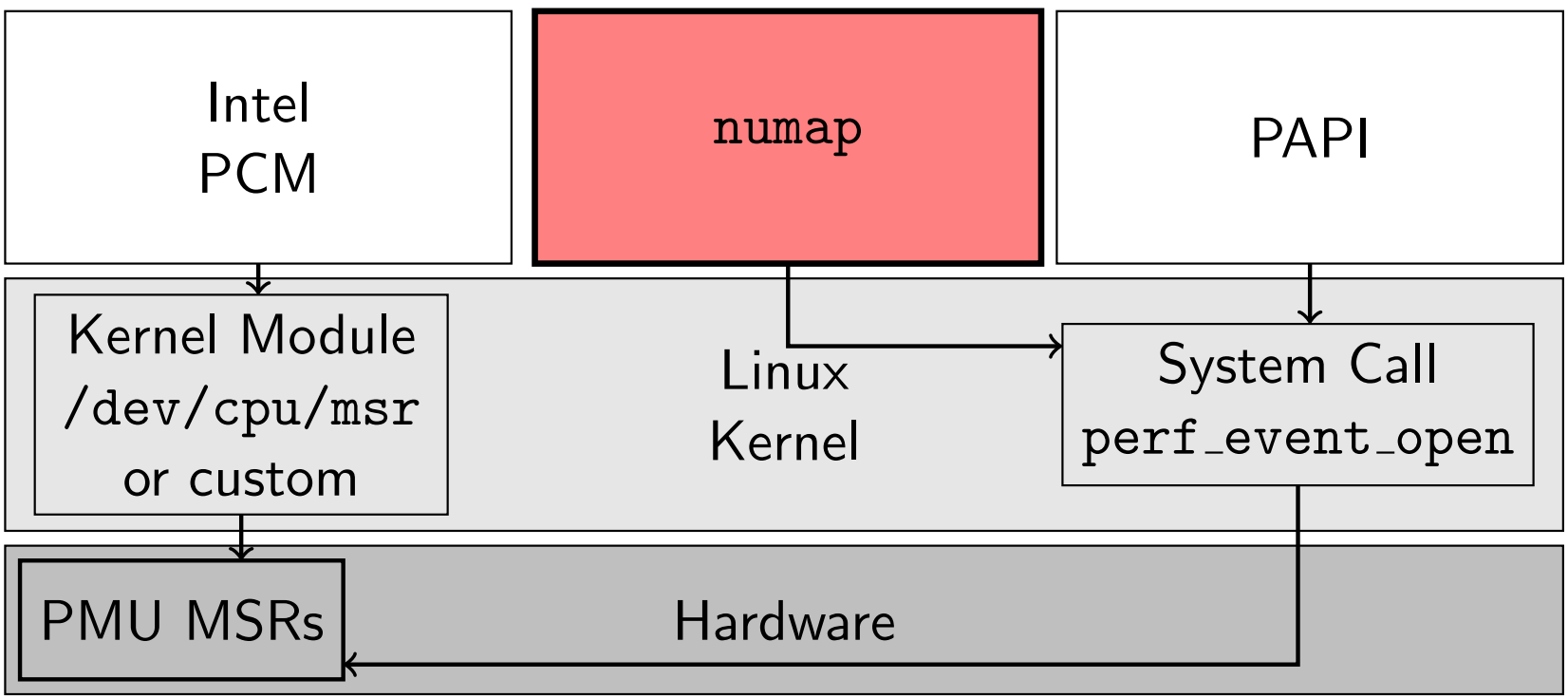
How To Use The PMU



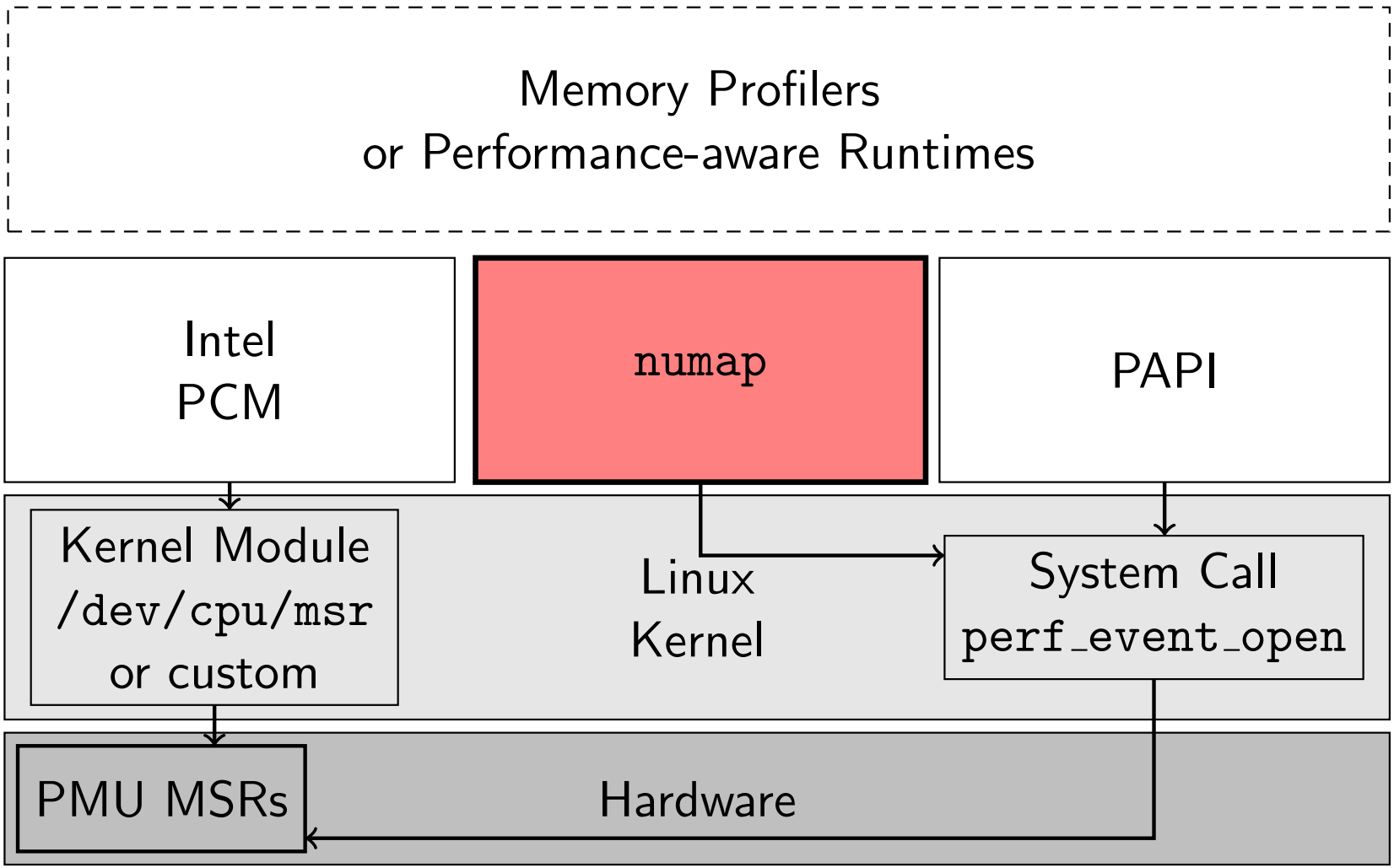
How To Use The PMU



How To Use The PMU



How To Use The PMU



numap Sampling API

```
// Init sampling parameters
int init_samp_session(
    struct samp_session *ses, int nb_threads,
    int sampling_rate, int mmap_pages_count);

// Start and stop memory sampling
int samp_read_start (struct samp_session *s);
int samp_read_stop  (struct samp_session *s);
int samp_write_start(struct samp_session *s);
int samp_write_stop (struct samp_session *s);

// Analyze the samples
int print_rd(File *f, struct samp_session *s);
int print_wr(File *f, struct samp_session *s);
int cpy_samples(struct samp_session *s,
                struct samp **dest, int *nb_sp);
```

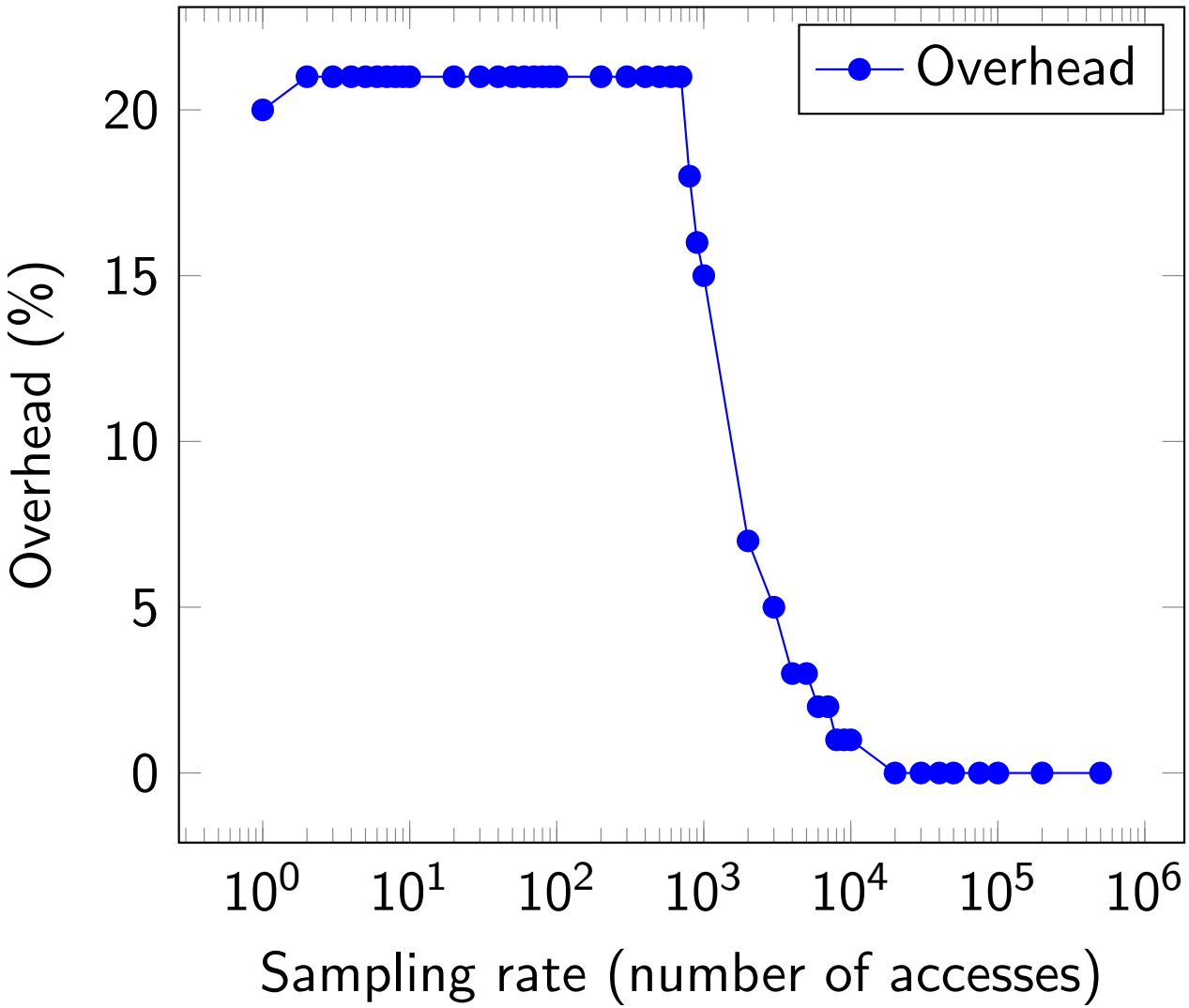
numap Usage

```
// Init read sampling
struct samp_session s;
init_samp_session(&s, 1, 10000, 64);
sm.tids[0] = syscall(SYS_gettid);

// Start memory read access sampling
samp_read_start(&s);

// Code to be profiled here
....

// Stop memory read access sampling
samp_read_stop(&s);

// Print memory read sampling results
print_rd(stdout, &s);
```

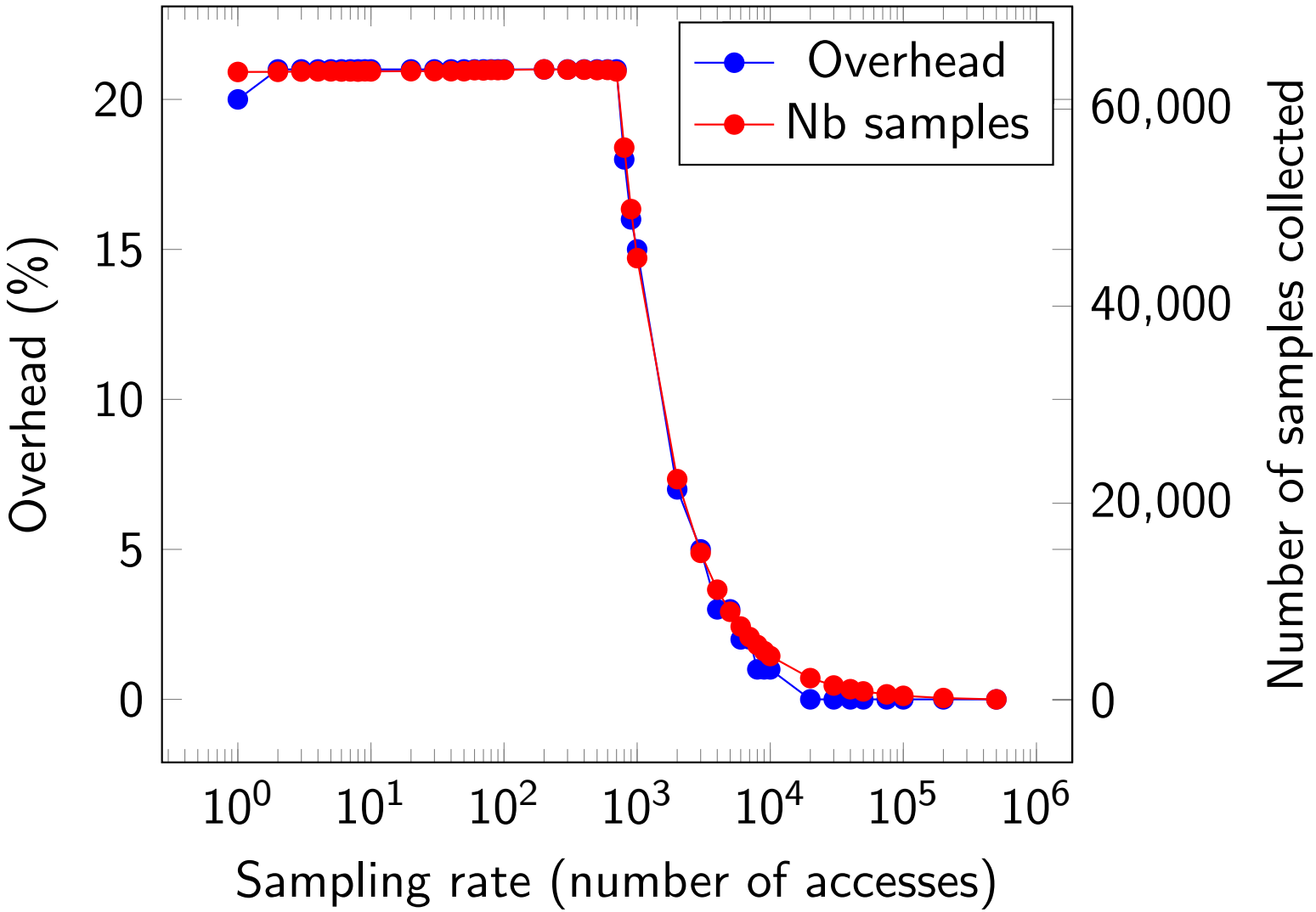
Generated Samples

```
pc=400c48, @=7ffd177a3950, src=L1,  latency=7
pc=400c48, @=7ffd177a3950, src=L1,  latency=7
pc=400c72, @=7ffd177a3968, src=L1,  latency=8
pc=400d3e, @=15164d0,      src=L2,  latency=22
pc=400c48, @=7ffd177a3950, src=L1,  latency=7
pc=400c4f, @=1450bc8,      src=L1,  latency=8
pc=400d3e, @=14ad700,      src=RAM, latency=269
```

Impact Of Sampling On Performance



Impact Of Sampling On Performance



`numap` Implementation

Rely on `perf_event_open`

Currently supports many Intel micro-architectures:

Micro-architecture Name	Read Sampling	Write Sampling
Nehalem - Lynfield decline	Yes	No
Westmere - EP decline	Yes	No
Sandy Bridge	Yes	Yes
Sandy Bridge - EP decline	Yes	Yes
Ivy Bridge	Yes	Yes
Ivy Bridge - E decline	Yes	Yes
Haswell - E decline	Yes	Yes
Haswell - DT decline	Yes	Yes

Conclusion

`numap`

Portable (Intel) and **simple** interface for low level memory sampling

Available: `https://github.com/numap-library/numap`

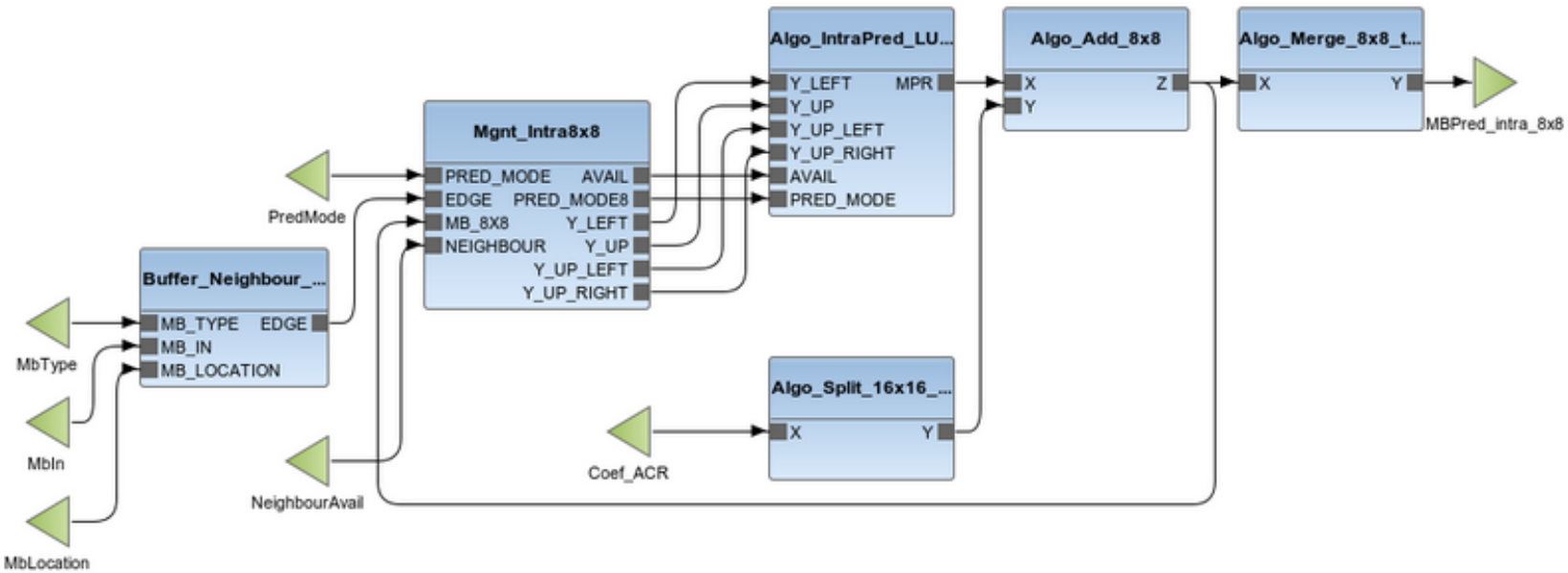
On Going/Future Work

Support for AMD processors

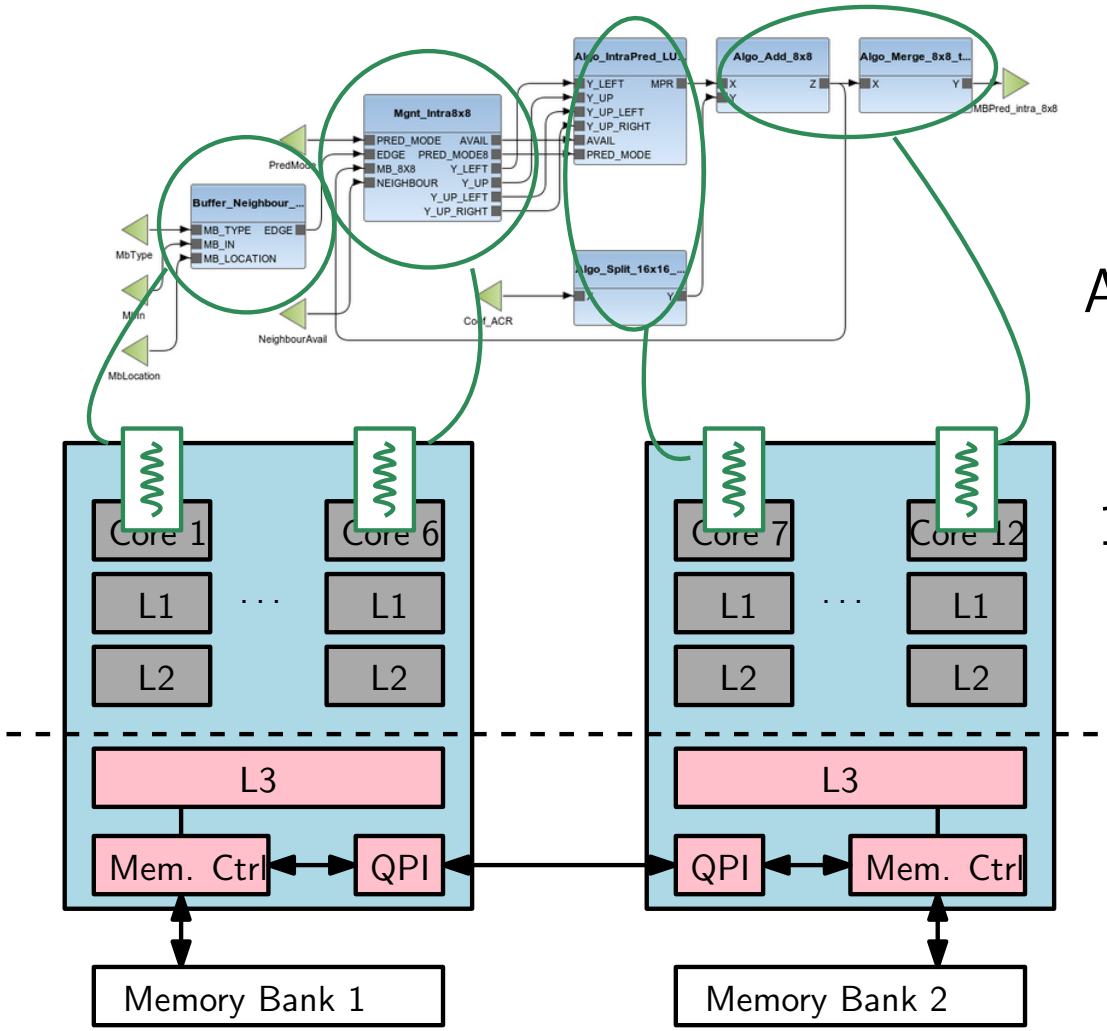
Integrate in PAPI

Support sampling for long lived applications

Dataflow Programming Models



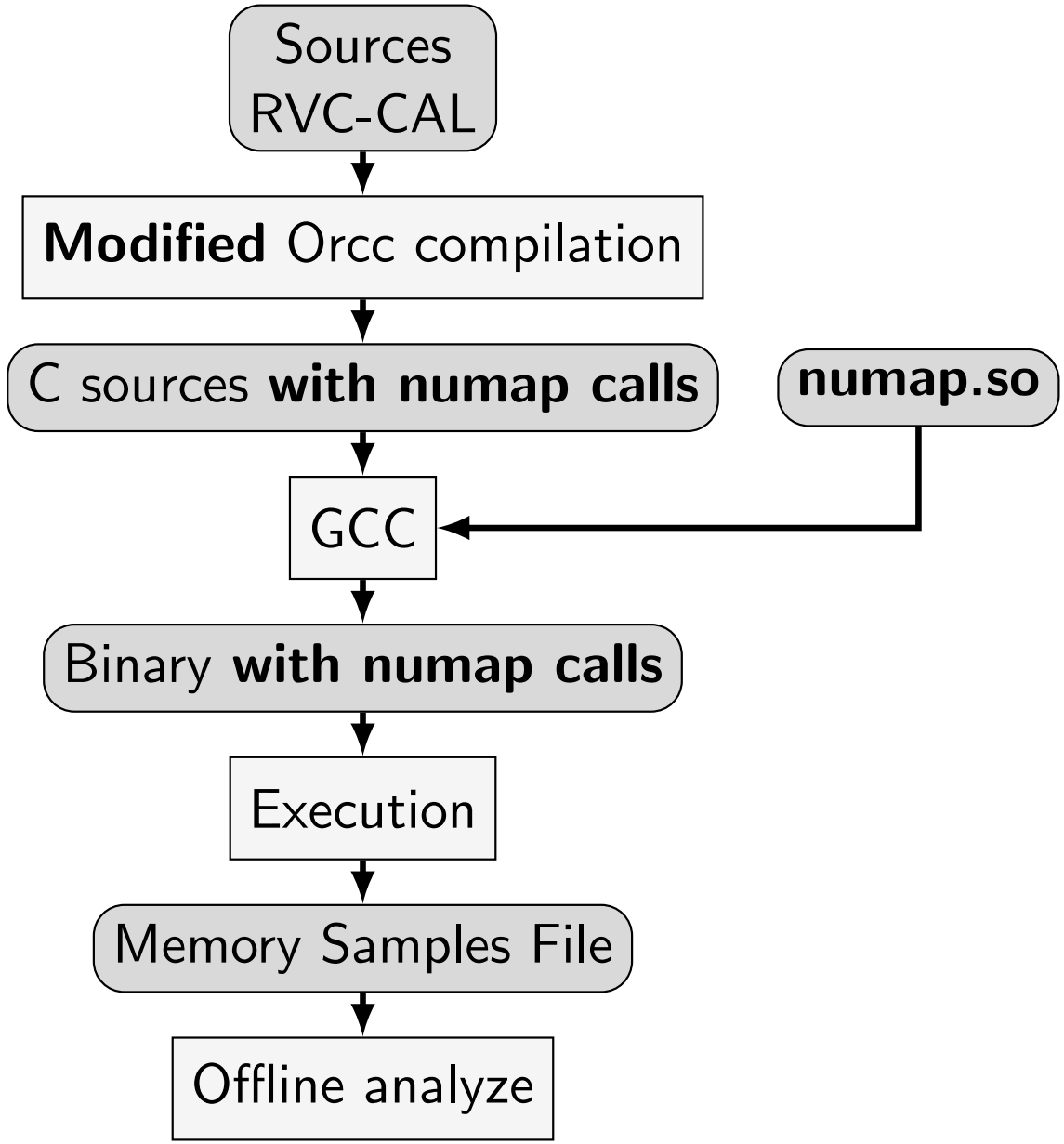
Runtimes for Dataflow Programs



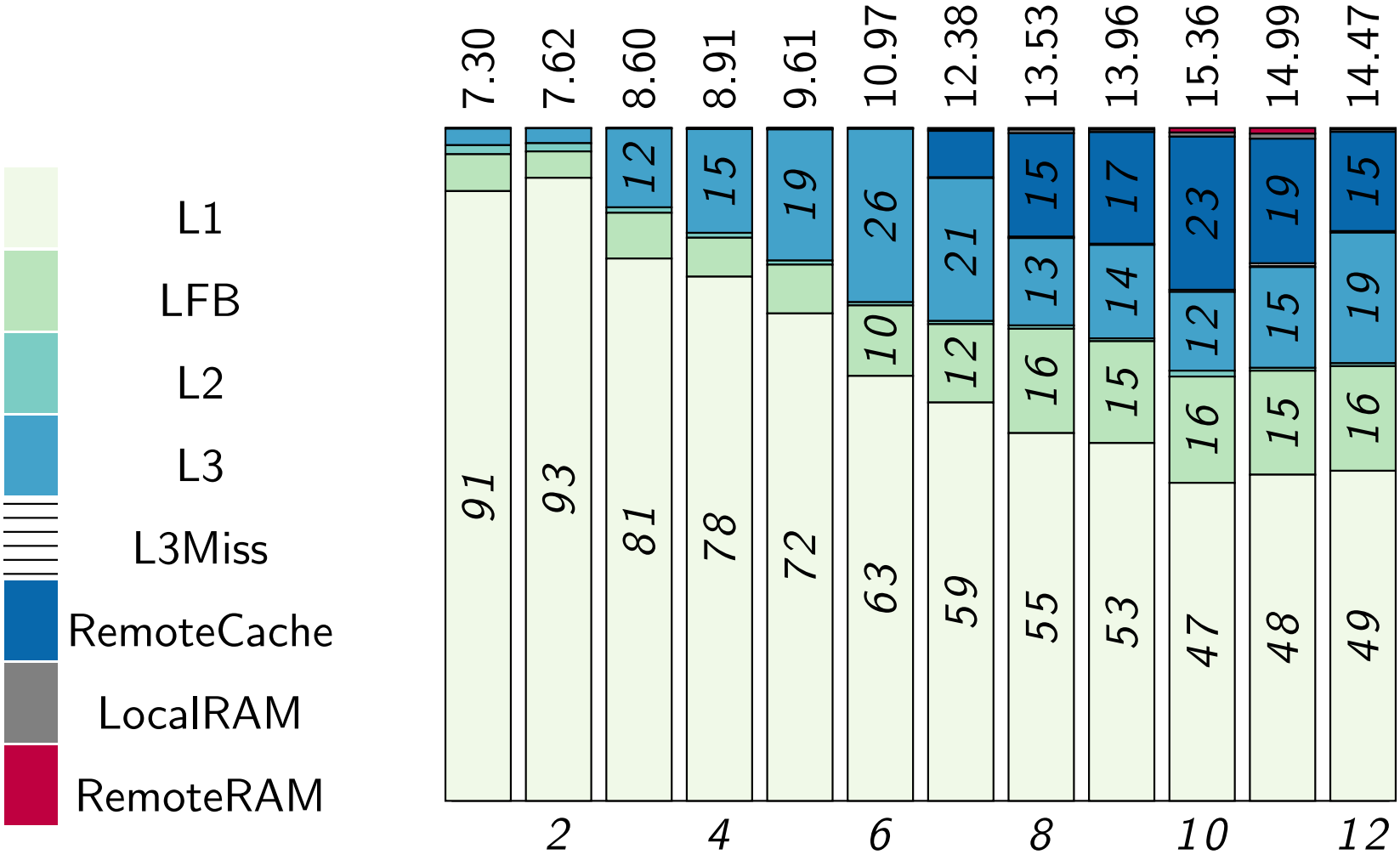
Actor to thread mapping

1 thread per core

Dataflow Memory Profiler - Toolchain



Dataflow Memory Profiler - Results

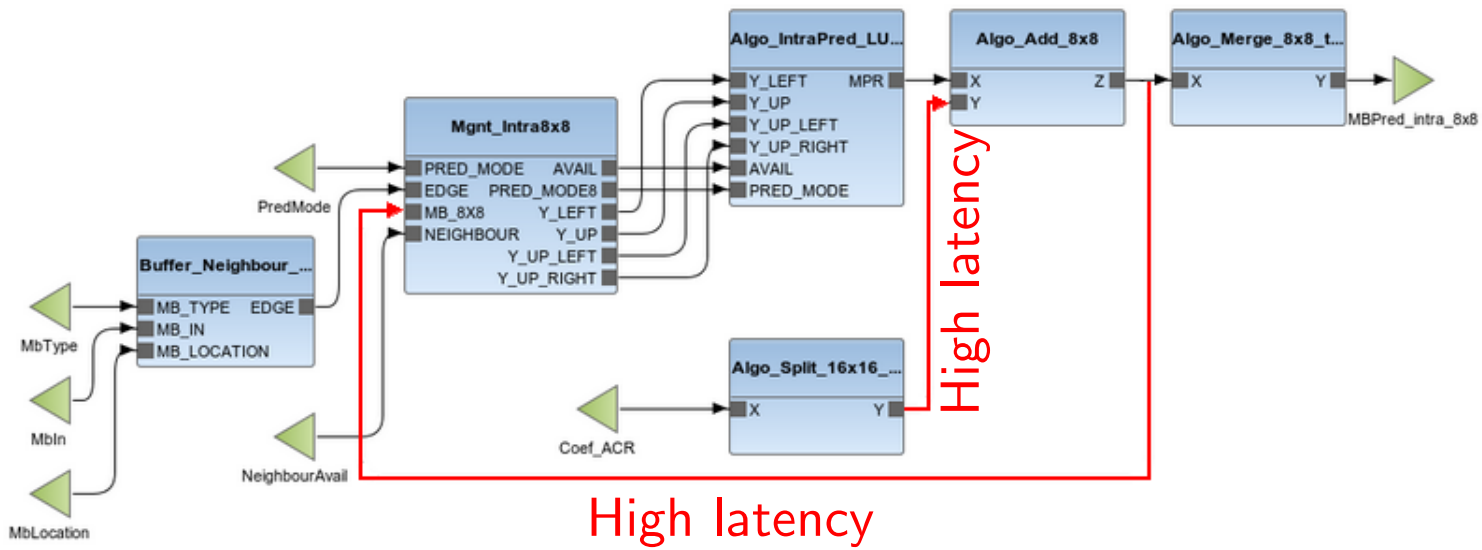


Distribution of memory accesses, average latency (in clock cycles), depending on number of cores.

Dataflow Memory Profiler - Results

Cost (%)	Src	L1 (%)	...	RC1 (%)
3.92	sched	94	...	0
3.86	FIFO-18	89	...	5
3.08	FIFO-9	71	...	10

Cost and memory location of accesses to FIFOs.



Identification of costly dataflow connexions.

numap vs PAPI

```
int PAPI_accum_counters (...)
int PAPI_num_counters (...)
int PAPI_num_components (...)
int PAPI_read_counters (...)
int PAPI_start_counters (...)
int PAPI_stop_counters (...)
int PAPI_flips (...)
int PAPI_flops (...)
int PAPI_ipc (...)
int PAPI_epc (...)
```

```
// Init sampling parameters
int init_samp_session(
    struct samp_session *ses, int nb_
    int sampling_rate, int mmap_pag

// Start and stop memory sampling
int samp_read_start (...);
int samp_read_stop  (...);
int samp_write_start(...);
int samp_write_stop (...);
```

Samples Accuracy vs Sampling Rate