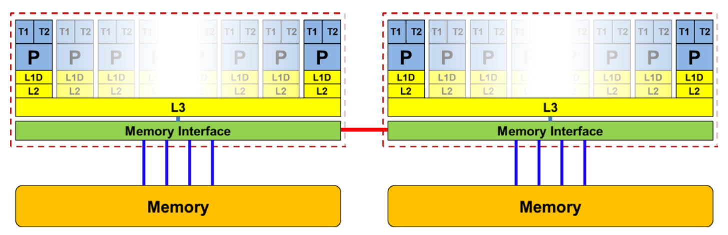
LIKWID 4: Lightweight Performance Tools

Jan Eitzinger, Thomas Röhl, Georg Hager and Gerhard Wellein
Erlangen Regional Computing Center (RRZE), 91058 Erlangen, Germany

LIKWID is a collection of command-line tools for performance-aware programmers of multicore and manycore CPUs. It follows the UNIX design philosophy of "one task, one tool". Among its many capabilities are system topology reporting, enforcement of thread-core affinity for threading, MPI, and hybrid programming models, setting clock speeds, hardware performance event counting, energy measurements, and low-level benchmarking. It currently supports x86 CPUs; ports to ARM and Power8 are work in progress.



Multi-/manycore challenges only get worse: where-to-run-what, complex topologies, hierarchical (cc?)NUMA, resource sharing, hardware threading, many cores, multiple bottlenecks, system configuration nightmares.

Full documentation, examples, FAQs, publications, source code, event validation data, and more:

Topology Challenge

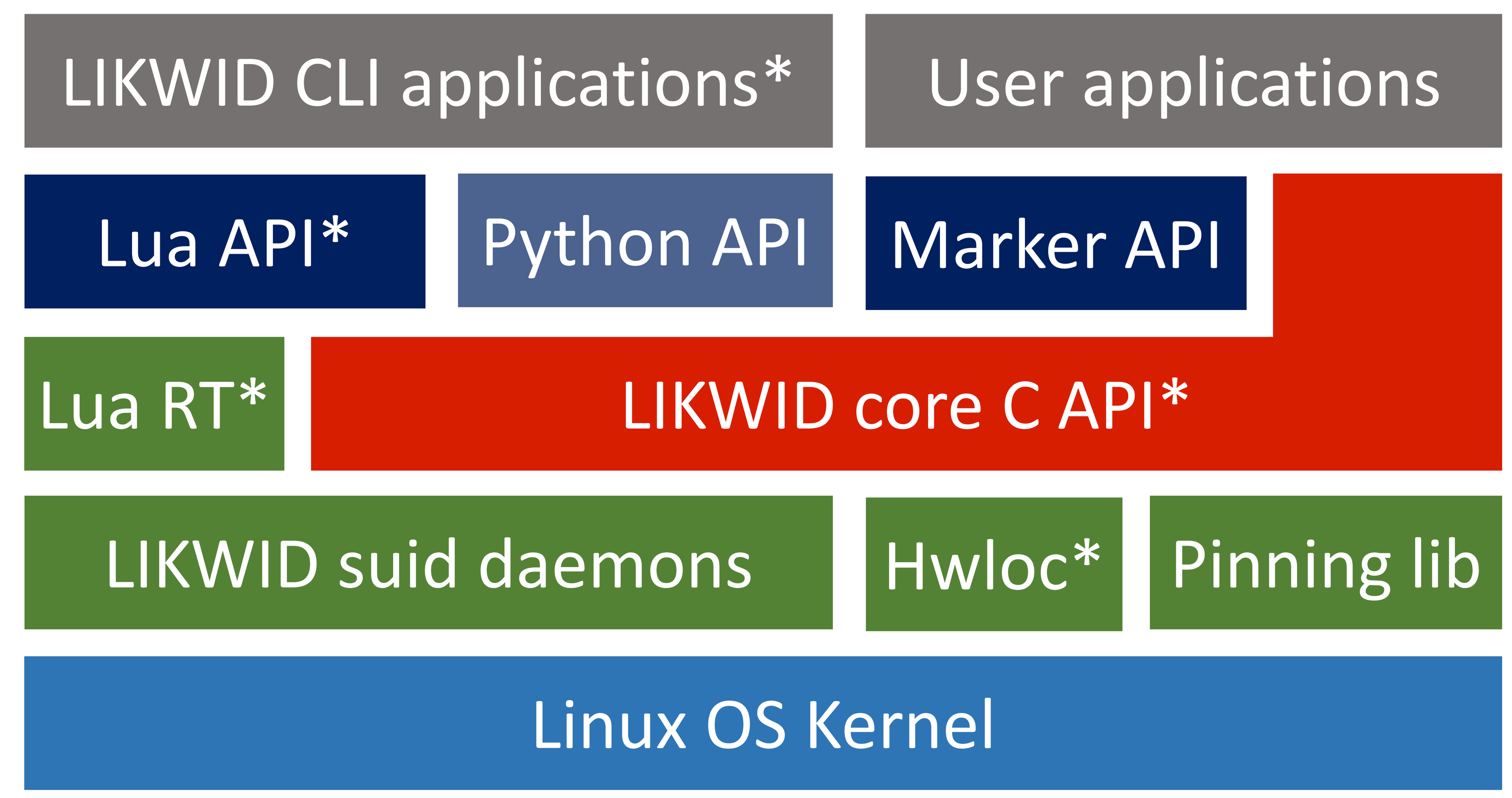
Topology features are steadily added to the hardware, making it harder to find out "what is where" in the machine. How do we get the full information in order to leverage the full power of the system?

LIKWID Solution

Full topology, cache and NUMA information provided via the core C API and **likwid-topology**. All LIKWID tools can access the data.

```
$ likwid-topology -g
[…]
*********************************************************************
Graphical Topology
*********************************************************************
Socket 0:
+----------------------------------------------------------------------
| +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +------
| |  0  7  | |  1  8  | |  2  9  | |  3 10  | |  4 11  | |  5 12  | |  6 13
| +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +------
| +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +------
| |  32kB  | |  32kB  | |  32kB  | |  32kB  | |  32kB  | |  32kB  | |  32kB
| +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +------
| +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +------
| |  256kB | |  256kB | |  256kB | |  256kB | |  256kB | |  256kB | |  256kB
| +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +------
| +--------------------------------------+ +-------------------------------
| |                 27MB                 | |
| +--------------------------------------+ +-------------------------------
+----------------------------------------------------------------------
```

LIKWID 4 Tools Architecture



*New in LIKWID 4.x

Configuration Challenge

What are the settings of performance-relevant hardware features (CPU features, prefetchers, Cluster on Die, Uncore frequency and freq. scaling, Turbo Mode, clock speed, power capping, …)? Can we change these how? How do we change these?

LIKWID Solution

Requesting and setting the status of the hardware prefetchers ; Cluster on Die (CoD) setting is part of node topology; changable clock speed and Turbo Mode settings; reading Uncore frequency limits; Read supported CPU features

```
# obtain CPU type, cache, topology and NUMA info
likwid-topology -c -g
# clean Linux FS buffer cache, evict dirty CL from LLC
likwid-memsweeper
# scan available frequencies
for f in `likwid-setFrequencies -l`; do
  # set CPI frequency for first socket (S0)
  likwid-setFrequencies -c S0 -f $f
  # scan DCU prefetcher off & on
  for pref in "-d DCU_PREFETCHER" "-e DCU_PREFETCHER"; do
    likwid-features $pref -c 0-15
    # scan core count
    for threads in `seq 1 16`; do
      # run w/ thread pinning & HW events & marker API
      # measure memory traffic and DP FLOP/s
      likwid-perfctr -C S0:0-$((threads-1)) \
                     -g MEM_DP -m ./a.out
    done
  done
done
```

LIKWID Tools support

Requesting and setting the status of the four hardware prefetchers with **likwid-features**; Cluster on Die (CoD) via **likwid-topology**; clock speed and Turbo Mode settings via **likwid-setFrequencies**. Uncore frequency and power limits by **likwid-powermeter**. The combination of LIKWID tools contributes to reproducible benchmarking by allowing users to take full control.

Thread Affinity Challenge

How do we make sure that threads/processes run where they should? Many specific solutions exist, but there is no common nomenclature.

LIKWID Solution

Common numbering schemes across all LIKWID tools. Physical (OS-based) and logical (entity-based) numbering. Supports all pthreads-based threading models (OpenMP, C++11, TBB, Cilk+, …) and several combinations of MPI & OpenMP implementations.

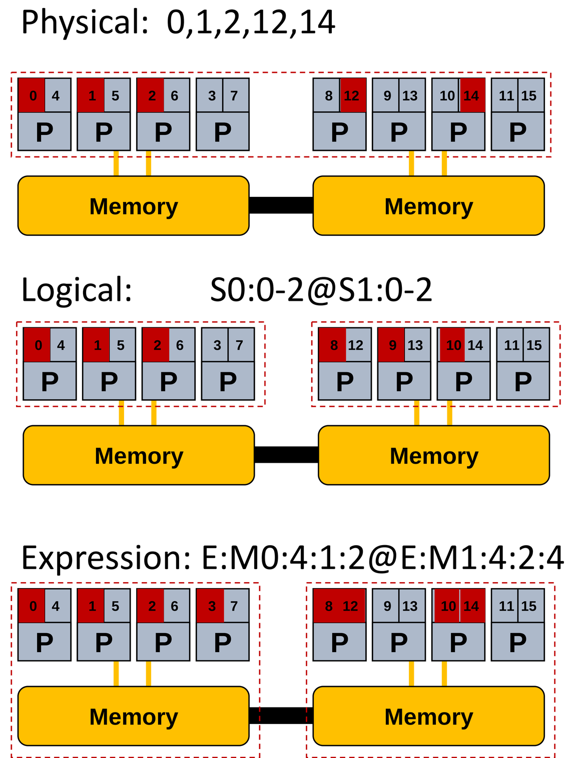
LIKWID Tools support

likwid-pin
pin threads to resources
likwid-mpirun
pin threads and processes in MPI or MPI+X programs
likwid-perfctr
measure HW performance events
likwid-memsweeper
clean FS buffer cache

Physical: 0,1,2,12,14

Logical: S0:0-2@S1:0-2

Expression: E:M0:4:1:2@E:M1:4:2:4




```
$ likwid-pin -p  # show affinity domains

Domain N:
0,4,1,5,2,6,3,7,8,12,9,13,10,14,11,15
Domain S0:
0,4,1,5,2,6,3,7
Domain S1:
8,12,9,13,10,14,11,15
```

```
Domain C0:
0,4,1,5,2,6,3,7
Domain C1:
8,12,9,13,10,14,11,15
Domain M0:
0,4,1,5,2,6,3,7
Domain M1:
8,12,9,13,10,14,11,15
```

```
$ likwid-perfctr -a
 Group name     Description
------------------------------------------------------
     L2CACHE     L2 cache miss rate/ratio
    TLB_DATA     L2 data TLB miss rate/ratio
    FLOPS_DP     Double Precision MFLOP/s
      MEM_DP     Arithmetic and main memory performance
     L3CACHE     L3 cache miss rate/ratio
      ICACHE     Instruction cache miss rate/ratio
        DATA     Load to store ratio
          L2     L2 cache bandwidth in MBytes/s
       CLOCK     Clock frequency
      BRANCH     Branch prediction miss rate/ratio
   FLOPS_AVX     Packed AVX MFLOP/s
         QPI     QPI Link Layer data
          HA     Main memory bandwidth from Home Agent
      CACHES     Cache bandwidth in MBytes/s
 FALSE_SHARE     False sharing
          L3     L3 cache bandwidth in MBytes/s
         MEM     Main memory bandwidth in MBytes/s
   TLB_INSTR     L1 Instruction TLB miss rate/ratio
        NUMA     Local and remote data transfers
      ENERGY     Power and Energy consumption
      MEM_SP     Arithmetic and main memory performance
    FLOPS_SP     Single Precision MFLOP/s
```

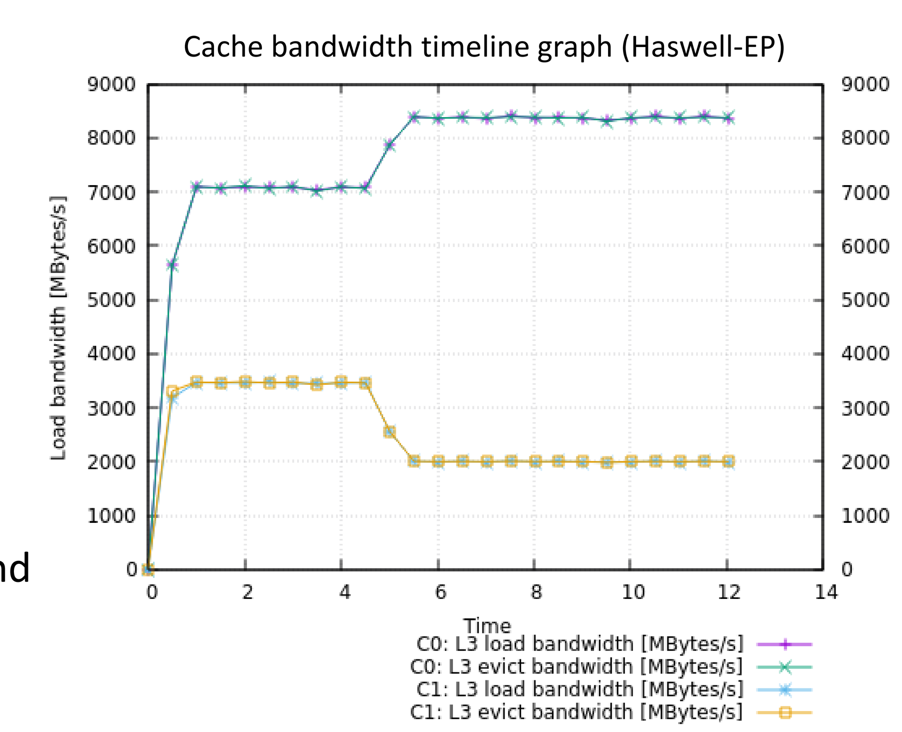
Hardware-Software Interaction Challenge

What is going on while code is being executed on the cores? What are the relevant bottlenecks? Are resources well utilized? Can measurements point to promising code optimizations? Are the measurements correct at all?

LIKWID Solution

Provides user-extensible performance groups that address interesting combinations of metrics; single event specification; Marker API to enable/disable/multiplex counting; live monitoring of derived metrics; C/C++, Fortran and Lua APIs for building tools and applications.

`$ likwid-perfscope -C 0,1 -g L3 ./a.out`



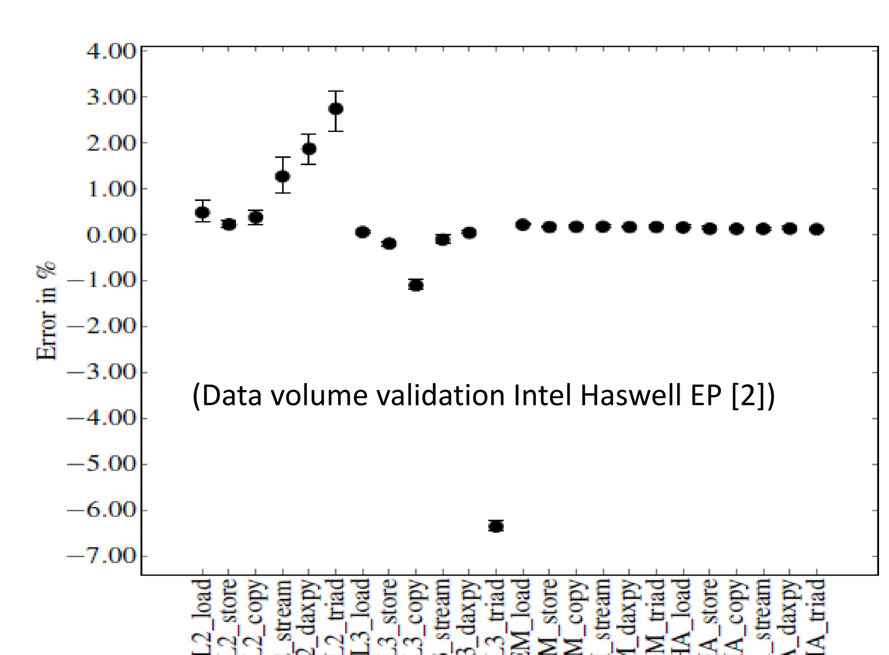
LIKWID Tools support

likwid-perfctr / likwid-perfscope
measure HW performance events, timeline display
likwid-mpirun
hybrid affinity with integrated event counting
likwid-powermeter
measure energy consumption, power, temperature

```
$ likwid-perctr -C 0,1 -g L3 -g FLOPS_AVX -T 1s ./a.out
[…]
Event Group 1: L3
+--------------------------------+------------+------------+
|             Metric             |   Core 0   |   Core 1   |
+--------------------------------+------------+------------+
|      Runtime (RDTSC) [s]       |   18.0017  |   18.0017  |
|      Runtime unhalted [s]      |   11.7513  |   11.7433  |
|          Clock [MHz]           |  2494.2335 |  2494.2323 |
|              CPI               |   0.7731   |   0.7728   |
|    L3 bandwidth [MBytes/s]     | 11401.3294 | 11429.3703 |
+--------------------------------+------------+------------+

Event Group 2: FLOPS_AVX
+----------------------+-----------+-----------+
|        Metric        |   Core 0  |   Core 1  |
+----------------------+-----------+-----------+
|  Runtime (RDTSC) [s] |  16.5222  |  16.5222  |
| Runtime unhalted [s] |  11.1478  |  11.1048  |
|      Clock [MHz]     | 2494.2336 | 2494.2317 |
|          CPI         |   0.7740  |   0.7735  |
|   Packed DP MFLOP/s  | 4410.1420 | 4410.1798 |
+----------------------+-----------+-----------+
```

Event/derived metric counts are validated against a wide range of assembly benchmarks with known behavior and calculable event counts for comparison.



New Features

CPU frequency manipulation in C/C++ library

Full support of Intel's Xeon Phi (KNL)

Uncore support for desktop chips

Upcoming Features

ARM v7/v8 and IBM POWER8 support

Linux perf_event as HPM backend

Support for Intel Kaby Lake

Support for 8 counters per core w/o SMT

Future Work

Generic plugin interface for other measurement facilities (GPU, libraries, …)

More derived metrics (Intel TMAM, RRZE performance patterns)

Reduced overhead for event count (de)activation

Integration in higher-level applications

Increase flexibility of benchmarking tool (latency, data structures, data types)

Benchmarking Challenge

What are the basic limitations of the hardware? How does it react to subtle code changes? Can we reverse engineer relevant features? Can we build micro benchmarks without trusting the compiler? Can we ensure a specific SIMD width is used?

LIKWID Solution

Assembly-level loop bench-marking tool **likwid-bench** comes with many standard preconfigured benchmark kernels; full control of threading & data placement; easily extensible; automatic boiler-plate code generation; calculation of benchmark metrics; integrated performance event counting (optional)

```
STREAMS 3             # STR[0-2] variables
TYPE DOUBLE
FLOPS 2               # per element
BYTES 24              # per element
LOADS 2               # per element
STORES 1              # per element
INSTR_LOOP 7          # whole loop
UOPS 8                # whole loop
vmovaps ymm5, [rip+SCALAR]
LOOP 4                # increment for GPR1
vmovaps    ymm1, [STR2 + GPR1*8]
vmulpd     ymm1, ymm1, ymm5
vaddpd     ymm1, ymm1, [STR1 + GPR1*8]
vmovntpd   [STR0 + GPR1*8], ymm1
```

Available benchmark kernels: copy, daxpy, ddot, load, store, stream, sum, triad, update

Kernel versions: Double, float, and int data types

Scalar, SSE, AVX, AVX512, NT Stores, FMA ops

```
$ likwid-bench -t stream_mem_avx -w N:1GB1
(stream triad with nt stores & AVX)
Test: stream_mem_avx
-------------------------------------------
Cycles:                         4472035767
Time:                       1.318346e+00 sec
Number of Flops:                2133332992
MFlops/s:                          1618.19
Data volume (Byte):            25599995904
MByte/s:                          19418.27
Cycles per update:                4.192534
Cycles per cacheline:            33.540274
Instructions:                   1866665489
UOPs:                           2133331968
-------------------------------------------
```

See it in action

Thanks to

Federal Ministry of Education and Research

Grant Nr. 01IH13009

References

[1] Treibig, Jan, Georg Hager, and Gerhard Wellein. "Likwid: A lightweight performance-oriented tool suite for x86 multicore environments." 2010 39th International Conference on Parallel Processing Workshops. IEEE, 2010.
[2] Röhl, Thomas, et al. "Validation of Hardware Events for Successful Performance Pattern Identification in High Performance Computing." Tools for High Performance Computing 2015. Springer International Publishing, 2016. 17-28.