LIKWD 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.

### LIKWID 4 Tools Architecture

<table>
<thead>
<tr>
<th>LIKWID CLI applications*</th>
<th>User applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lua API*</td>
<td>Python API</td>
</tr>
<tr>
<td>Lua RT*</td>
<td>Marker API</td>
</tr>
</tbody>
</table>

**LIKWD core C API**

**LIKWD suid daemons**

**Hwloc**

**Pinning lib**

**Linux OS Kernel**

*New in LIKWID 4.x*

### 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 Tools support**

- likwid-perfctr / likwid-perfscope
- measure HW performance events, timeline display
- likwid-mpiran
- hybrid affinity with integrated event counting
- likwid-powermeter
- measure energy consumption, power, temperature event/derived metric counts are validated against a wide range of assembly benchmarks with known behavior and calculable event counts for comparison.

### 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; changeable clock speed and Turbo Mode settings; reading Uncore frequency limits; Read supported CPU features

**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.

### 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 (deactivation 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 coding (optional)

### Available benchmark kernels

- Double, float, and int data types
- Scalar, SSE, AVX, AVX512, NT Stores, FMA ops

---

**References**


---

**Graphical Topology**

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

**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++, TBB, Cilk++, ...) and several combinations of MPI & OpenMP implementations.

**LIKWID Tools support**

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

---

**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 CLI applications**

- Lua API*
- Python API
- Marker API
- LIKWID core C API*
- LIKWID suid daemons
- Hwloc*
- Pinning lib

---

**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 Tools support**

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

---

**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; changeable clock speed and Turbo Mode settings; reading Uncore frequency limits; Read supported CPU features

**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.

**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 (deactivation 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 coding (optional)

**Available benchmark kernels**

- Double, float, and int data types
- Scalar, SSE, AVX, AVX512, NT Stores, FMA ops

---

**References**


---

**Graphical Topology**

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