Machine Architecture Tools (LIKWID, hwloc, perftools)

CSE 6240 – High Performance Parallel Computing

Thomas Gruber Thomas.gruber@fau.de

NHR@FAU

Who am I?

- Thomas Gruber né Roehl

- Apprenticeship as IT-Specialist at
  Regional Computing Center Erlangen (RRZE)
- M.Sc in Computer Science from RWTH Aachen
- Starting with LIKWID development in 2013 at RRZE

- Other projects:
  - ClusterCockpit: Cluster-wide job-specific monitoring
  - MachineState: System settings and runtime environment recorder

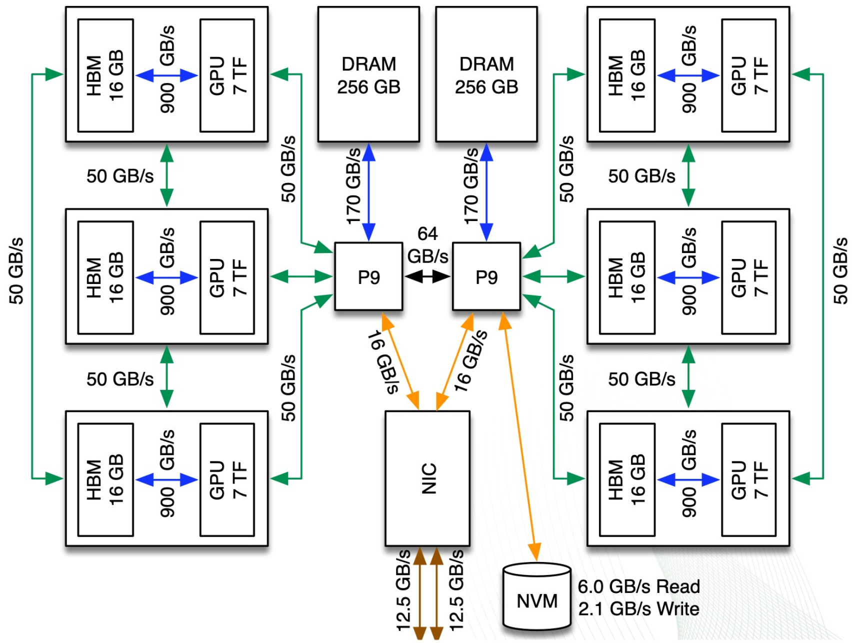
The NHR Alliance

Provide nationwide HPC resources for researchers of German universities

- Powerful and reliable HPC infrastructure
- Expert user support and user training
- NHR@FAU fields of expertise within NHR
  - Atomistic Simulations
  - Performance Engineering & Tools
- Long-term funding: 2021 – 2030
  (Federal govt. & State of Bavaria; FAU)

Motivation

- System architecture more and more complex
  - Not only from the hardware perspective (more subsystems)
  - But also from software POV (task to resource affinity)



(src: CSE6240 - Performance Modeling - 2/7/23)

How to programmatically get relation of components?

- Portable? Multi-arch? Multi-OS?
- Discover relations between devices
- Interaction with other libraries
- Control compute & memory affinity?

Portable Hardware Locality (hwloc)

Discover hardware resources in parallel architectures

Portable Hardware Locality (hwloc)

- OpenMPI sub-project but used in various libraries/tools/applications
- Consists of hwloc (local topology) and netloc (network topology)
- Mainly developed by the TADaaM team at Inria (Bordeaux, France)
- Features:
  - portable abstraction (architectures, OS, co-processors, …)
  - hierarchical topology
  - CPU and memory binding
  - C/C++-API
  - Active development (new topological quirks added quickly)

Example topologies

AMD EPYC 9654

Intel Xeon Platinum 8360Y

Hierarchical topology

- Hwloc organizes the system topology in an imperfect tree

```
hwloc_topology_t topo = NULL;
err = hwloc_topology_init(&topo);
// error check
err = hwloc_topology_set_type_filter(topo, …, …);
// error check
err = hwloc_topology_set_flags(topo, …);
// error check
err = hwloc_topology_load(topo);
// error check
[…]
hwloc_topology_destroy(topo);
```

Default filters do not detect some devices, e.g. PCI devices

Control device discovery

```
depth 0:                1 Machine (type #0)
 depth 1:               2 Package (type #1)
  depth 2:              2 L3Cache (type #6)
   depth 3:             4 Group0 (type #12)
    depth 4:            72 L2Cache (type #5)
     depth 5:           72 L1dCache (type #4)
      depth 6:          72 L1iCache (type #9)
       depth 7:         72 Core (type #2)
        depth 8:        144 PU (type #3)
Special depth -3:       4 NUMANode (type #13)
Special depth -4:       10 Bridge (type #14)
Special depth -5:       9 PCIDev (type #15)
Special depth -6:       271 OSDev (type #16)
```

- System topology updates require reloading of tree, no under-the-hood updates of the tree

Portable abstraction

- Each topological entity is an **object (`hwloc_obj_t`)**

```
hwloc_obj_t obj = NULL, next = NULL;
count = hwloc_get_nbobj_by_type(topo, <type>);
obj = hwloc_get_obj_by_type(topo, <type>, <logical_idx>);
next = hwloc_get_next_obj_by_type(topo, <type>, obj);

count = hwloc_get_nbobj_by_depth(topo, <depth>);
obj = hwloc_get_obj_by_depth(topo, <depth>, <logical_idx>);
next = hwloc_get_next_obj_by_depth(topo, < depth >, obj);
```

- Information per object:
  - ID given by the operating system (**`os_index`**)
  - Object type (Lx cache, …) maybe with subtype (unified, instruction or data cache)
  - Relation to parent(s), sibling(s) and cousin(s) objects
  - Type-specific information (key-value pairs)
  - …

Bridging the gap between hardware and software

Hardware performance counters

(Short) History of hardware counters

- Already available in ‘old’ architectures but proprietary
- Reverse-engineered for Intel Pentium in 1994[1]
- Introduced (publicly) by AMD with AMD K6 (1997)
- Nowadays available in all systems

- Logic that runs besides demand computation
  - Great observability
  - (Almost) No overhead
  - Originally used for verification by chip vendors

[1] *“Pentium Secrets: Undocumented features of the Intel Pentium can give you all the information you need to optimize Pentium code”*
Terje Mathisen, *eByte Magazine, July 1994, Page 191*

History

Counts from LIKWID

Arcitecture	Num PMCs	Num Events	AccessModes
Intel Pentium	2	30	MSR
Intel Pentium MMX	2	30	MSR, RDPMC
AMD K8	4	163	MSR, RDPMC
Intel Core2	5	423	MSR, RDPMC
Intel Nehalem Intel Nehalem EX	16 105	498 2623	MSR, RDPMC
IBM POWER8 / POWER9	6 / 48	996 / 818	MCR
AMD Interlagos (Kabini)	12	428	MSR, RDPMC
Intel Sandybridge Intel Sandybridge EP	31 102	422 788	MSR, RDPMC MSR, RDPMC, PCI
Intel Skylake Intel Skylake SP	37 337	444 2061	MSR, RDPMC MSR, RDPMC, PCI
ARM Neoverse N1	6	122	LDR
AMD Zen3	21	303	MSR, RDPMC
Intel Icelake SP	408	3033	MSR, RDPMC, PCI, MMIO

Issues?

- Due to verification history
  - (partly) very specific events
  - Event names are written from hardware architect POV
- Generational differences
  - Same event (name) might count differently
  - Important events missing (Intel Haswell does not support counting FP ops)
  - New access modes increase space for errors
- Security
  - MSRs (x86) are used not only for hardware counting
  - Access commonly restricted to kernel space
  - Monitoring might reveal user code (behavior)

`perf_event` and `perf`

Focus on `perf_event`

For `perf` see CSE6240 – Profiling - 2/14/23

Short overview

- History
  - Prior to `perf_event` only kernel patches ([1], [2]) existed
  - `perf_event` introduced with Linux 2.6.31 (2009)
- Linux kernel interface for performance event monitoring
  - Hardware performance counters
  - Kernel internal structure & event monitoring
- Single system call [`perf_event_open`] for setup

```
static long perf_event_open(struct perf_event_attr *config,
                            pid_t pid, int cpu, int group_fd,
                            unsigned long flags) {
  return syscall(__NR_perf_event_open, config, pid, cpu, group_fd, flags);
}
```

- Access control through `/proc/sys/kernel/perf_event_paranoid`

Configuration (I)

- Version dependent configuration structure **`struct perf_event_attr`**
- **`size`** field always **`sizeof(struct perf_event_attr)`**
- Counting modes
  - User-controlled start/stop/reset
  - Instruction/time/… based sampling (Intel PEBS, AMD IBS, …)
  - Result access (file descriptors, grouped FDs, MMAP, ring buffer)
- Unit configuration
  - Each unit exports a sysfs folder: **`/sys/bus/event_source/devices/<unit>`**
  - Each unit has a **`type`**
  - Config struct contains one or more **`config`** field(s)
    How to populate these fields, check unit's **`format`** folder
  - Other flags in the struct: counting scope (kernel, userspace, VMs, …), inherit to child processes, start disabled, addresses to allocated space, …

Configuration (III)

- Configuration left:
  - Which process(es) should be counted
  - Which part of the system (CPUs) should be measured

PID	CPU	Description
0	-1	Calling process/thread on any CPU
0	>= 0	Calling process/thread when running on specified CPU
> 0	-1	Specified PID on any CPU
> 0	>= 0	Specified PID on specified CPU
-1	>= 0	All processes/thread on specified CPU **`perf_event_paranoid <= 1`** **or** **`CAP_PERFMON/CAP_SYS_ADMIN`**

Usage

- `foreach event:`
  - `struct perf_event_attr config = create_config(event);`
  - `fd = perf_event_open(config, <pid>, <cpu>, -1, 0);`
- `foreach fd: ioctl(fd, PERF_EVENT_IOC_RESET, 0);`
- `foreach fd: ioctl(fd, PERF_EVENT_IOC_ENABLE, 0);`
- `<Code region>`
- `foreach fd: ioctl(fd, PERF_EVENT_IOC_DISABLE, 0);`
- `foreach fd OR foreach group_fd as fd:`
  - `read(fd, &result, sizeof(long long));`
  - `add(total_result, result);`

Use first FD of unit(!) event as **`group_fd`** to reduce reads

With **`group_fd`**, read all group data with a single read. **`read_format`** in config, read multiple **`long long's`**

Usage

- Different access modes require more configuration
  - User allocation for ring buffer with kernel for samples → Interrupt when full
  - Group FDs for less overhead when accessing (group per unit)

- If events exceed physical counters, multiplexing is applied
  Get enabled vs. running time ratio with different **`read_format`**

- Some units are per CPU, others per socket or other topological entity (check **`cpumask`** in sysfs)
- Only few events pre-configured

Pros & Cons

- Pro
  - Vendor support → Available on almost all systems
  - Get infos from the kernel (eBPF, events, …)
  - “Simple” API (one system call, some IOCTLs, common SysProg)
  - All required configuration information published via sysfs
  - Usage control via procfs and/or capabilities
  - Process/thread support (limit counting to PIDs)
- Con
  - Almost similar between archs but vendors partly do their own stuff
  - Overhead not really known and hard to measure
  - Scalability issues (PID × CPUs × Users × …)
  - Intransparent event scheduling (did it multiplex?)
  - Insufficient error handling and almost non-existent documentation

LIKWID

NHR@FAU: https://hpc.fau.de/research/tools/likwid/

High Performance Computing

LIKWID tool suite

- LIKWID tool suite:
  Like I Knew What I'm Doing
- Support for x86, ARM and PPC and Nvidia GPUs
  (upcoming AMD GPUs, Intel GPUs and recent CPUs)
- Works with standard kernel interfaces
- C/C++ library with command-line tools
  - Lua interface (builtin)
  - Julia interface
  - Python interface

- Repo: **https://github.com/RRZE-HPC/likwid**
- Docs: **https://github.com/RRZE-HPC/likwid/wiki**
- Zenodo: **https://doi.org/10.5281/zenodo.4275676**

DOI: 10.1109/ICPPW.2010.38

LIKWID tools

- **`likwid-topology`** - Print thread and cache topology
- **`likwid-pin`** - Pin threaded application without touching code
- **`likwid-perfctr`** - Measure performance counters
- **`likwid-powermeter`** - Measure energy consumption
- **`likwid-bench`** - Microbenchmarking tool and environment
- **`likwid-mpirun`** - MPI wrapper to **`likwid-pin`** and **`likwid-perfctr`**
- **`likwid-features`** - Manipulation of hardware feature flags
- **`likwid-setFrequencies`** - Manipulation of various frequencies

likwid-topology

- Get information about the current system from different sources: hwloc, procfs, sysfs and CPUID (x86)

- Hardware thread topology
  How many sockets? HW Thread → socket mapping? SMT?
- Cache topology
  How many cache levels? Sizes? Inclusive/exclusive? Cacheline size?
- NUMA topology
  How is the memory distributed? HW Thread → NUMA node mapping?
- Nvidia GPU topology (if built with Nvidia support)
  How many GPU? How much GPU memory?
  GPU → NUMA node mapping?

Output of `likwid-topology`

DEMO

```
$ likwid-topology
--------------------------------------------------------------------------------
CPU name: Intel(R) Xeon(R) Platinum 8360Y CPU @ 2.40GHz
CPU type: Intel Icelake SP processor
CPU stepping:  6
********************************************************************************
Hardware Thread Topology
********************************************************************************
Sockets:        2
Cores per socket:   36
Threads per core:   1
--------------------------------------------------------------------------------
HWThread        Thread          Core        Die         Socket          Available
0               0               0           0           0               *
1               0               1           0           0               *
2               0               2           0           0               *
[…]
69              0               69          0           1               *
70              0               70          0           1               *
71              0               71          0           1               *
--------------------------------------------------------------------------------
Socket 0:       ( 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 … 23 24 25 26 27 28 29 30 31 32 33 34 35 )
Socket 1:       ( 36 37 38 39 40 41 42 43 44 45 46 47 48 … 59 60 61 62 63 64 65 66 67 68 69 70 71 )
--------------------------------------------------------------------------------
```

All physical processor IDs

Output of `likwid-topology`

DEMO

```
********************************************************************************
Cache Topology
********************************************************************************
Level:              1
Size:               48 kB
Cache groups:       ( 0 ) ( 1 ) ( 2 ) ( 3 ) ( 4 ) ( 5 ) … ( 64 ) ( 65 ) ( 66 ) ( 67 ) ( 68 ) ( 69 ) ( 70 ) ( 71 )
--------------------------------------------------------------------------------
Level:              2
Size:               1.25 MB
Cache groups:       ( 0 ) ( 1 ) ( 2 ) ( 3 ) ( 4 ) ( 5 ) … ( 64 ) ( 65 ) ( 66 ) ( 67 ) ( 68 ) ( 69 ) ( 70 ) ( 71 )
--------------------------------------------------------------------------------
Level:              3
Size:               54 MB
Type:               Unified cache
Associativity:      12
Number of sets:     73728
Cache line size:    64
Cache type:         Non Inclusive
Shared by threads:  36
Cache groups:       ( 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 … 23 24 25 26 27 28 29 30 31 32 33 34 35 )
                     ( 36 37 38 39 40 41 42 43 44 45 46 47 48 … 59 60 61 62 63 64 65 66 67 68 69 70 71 )
--------------------------------------------------------------------------------
```

Additional cache info with `-c` option

Output of `likwid-topology`

```
********************************************************************************
NUMA Topology
********************************************************************************
NUMA domains:              4
--------------------------------------------------------------------------------
Domain:                    0
Processors:                ( 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 )
Distances:                  10 11 20 20
Free memory:               119059 MB
Total memory:              128553 MB
--------------------------------------------------------------------------------
Domain:                    1
Processors:                ( 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 )
Distances:                  11 10 20 20
Free memory:               128196 MB
Total memory:              129020 MB
--------------------------------------------------------------------------------
Domain:                    2
Processors:                ( 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 )
Distances:                 20 20 10 11
Free memory:               128033 MB
Total memory:              128978 MB
--------------------------------------------------------------------------------
Domain:                    3
Processors:                ( 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 )
Distances:                 20 20 11 10
Free memory:               128719 MB
Total memory:              129017 MB
```

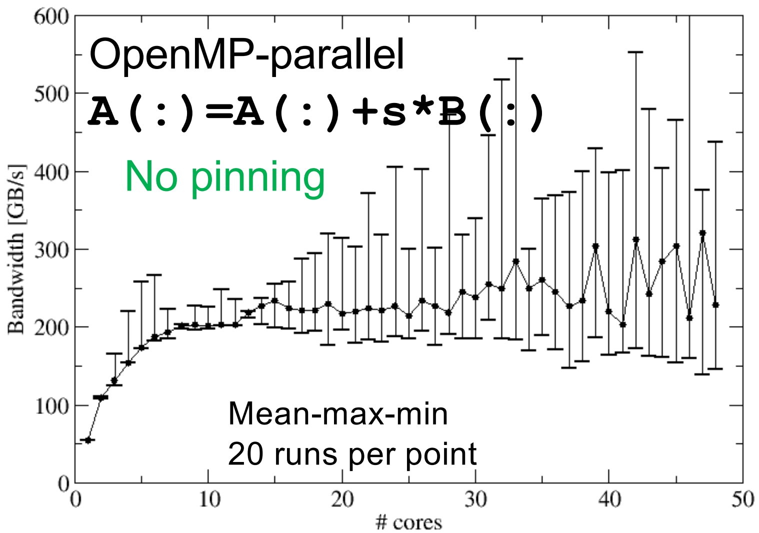
DEMO

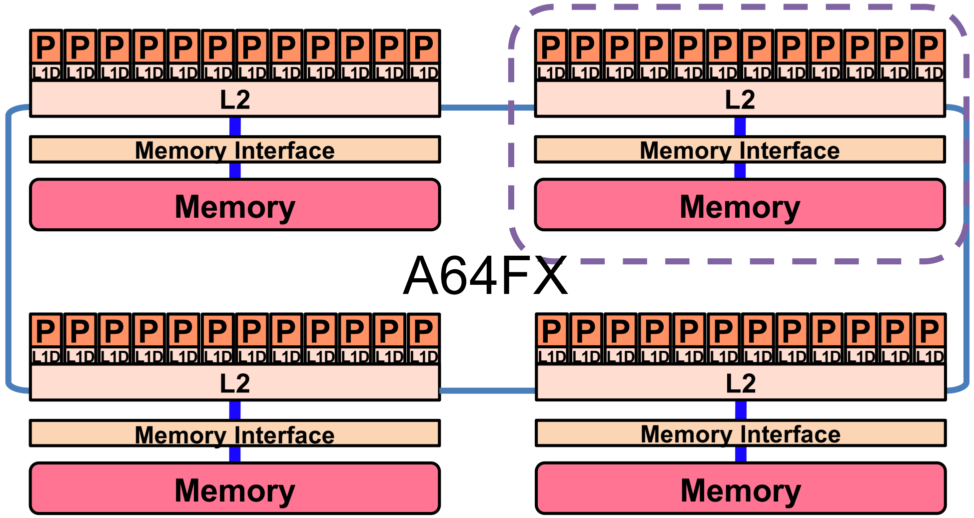
Output similar to
`numactl --hardware`

`Sockets:` `2`
`Threads per core:` `1`

Cluster on Die (CoD) mode and SMT disabled!

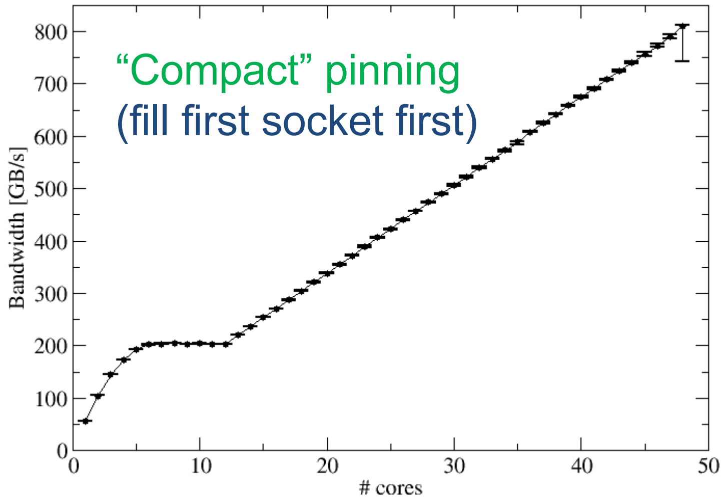
Importance of affinity control aka pinning



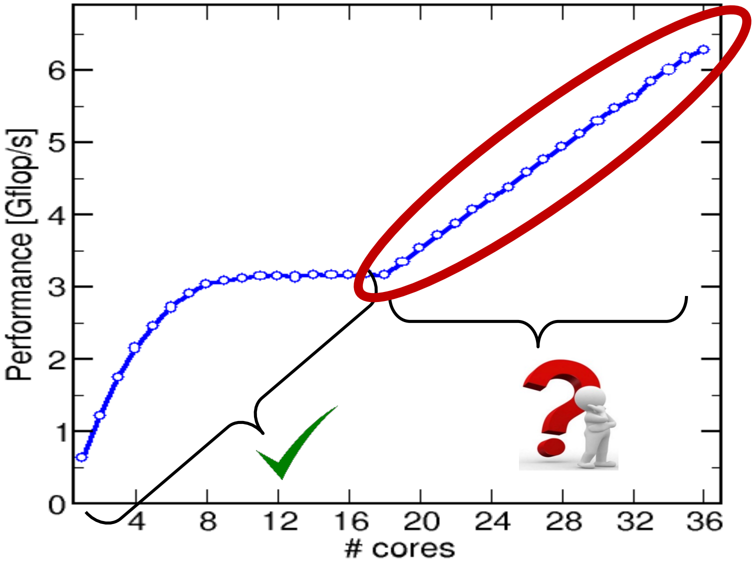


There are several reasons for caring about affinity:

- Eliminating performance variation
- Making use of architectural features
- Avoiding resource contention

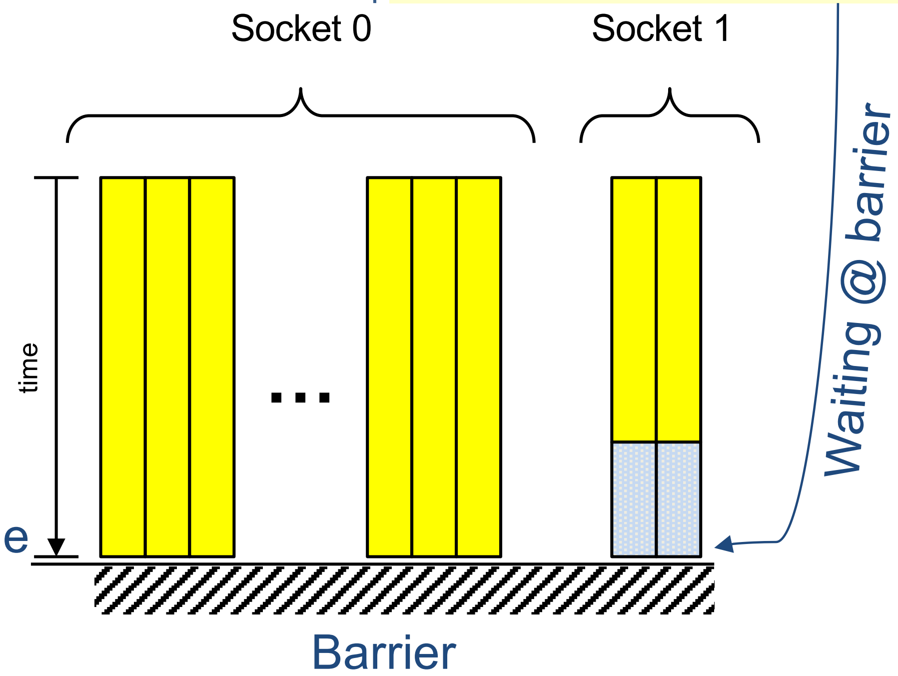


Interlude: Why the weird scaling behavior?




```
!$omp parallel do schedule(static)
 do i = 1,N
   a(i) = b(i) + s * c(i)
!$omp end parallel do
```

implicit barrier



- Every thread has the same workload
- Performance of left socket is saturated
- Barrier enforces waiting of “speeders” at sync point
- Average performance of each “right” core == average performance of each “left” core → linear scaling

likwid-pin

- Pins processes and threads to specific cores without touching code
- Directly supports pthreads, gcc OpenMP, Intel OpenMP
- Based on combination of wrapper tool together with overloaded pthread library
  → binary must be dynamically linked!
- Supports logical core numbering within topological entities (thread domains)

- Simple usage with physical (kernel) core IDs:

```
$ likwid-pin -c 0-3,4,6  ./myApp parameters
$ OMP_NUM_THREADS=4 likwid-pin -c 0-9 ./myApp params
```

No overwriting of existing env variables

LIKWID terminology: Thread group syntax

- The OS numbers all processors (hardware threads) on a node
- The numbering is enforced at boot time by the BIOS

- LIKWID introduces thread domains consisting of hardware threads sharing a topological entity (e.g. socket or shared cache)
- A thread domain is defined by a single character + index

Physical HW threads first!

```
+-------------------------------------+
| +------+ +------+ +------+ +------+ |
| | 0  4| | 1  5| | 2  6 | | 3  7 | |
| +------+ +------+ +------+ +------+ |
+-------------------------------------+
```

- Example for likwid-pin:
  `$ likwid-pin -c S0:0-3 ./a.out`
- Thread group expressions may be chained with `@`:
  `$ likwid-pin -c S0:0-2@S1:0-2 ./a.out`

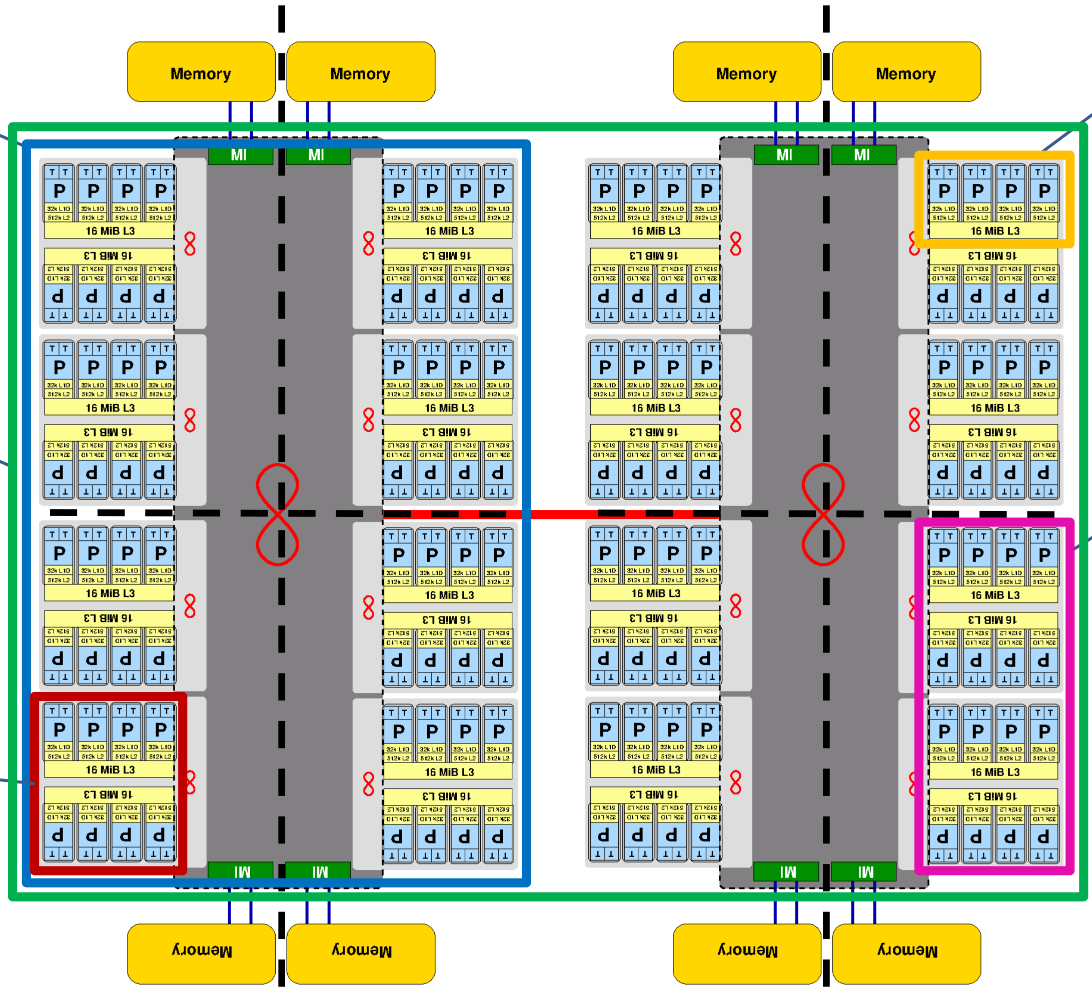
```
+-------------------------------------++-------------------------------------+
| +------+ +------+ +------+ +------+ || +------+ +------+ +------+ +------+ |
| | 0  8 | | 1  9 | | 2 10 | | 3 11 | || | 4 12 | | 5 13 | | 6 14 | | 7 15 | |
| +------+ +------+ +------+ +------+ || +------+ +------+ +------+ +------+ |
+-------------------------------------++-------------------------------------+
```

Available thread domains/unit prefixes (LIKWID 5.2)

S socket

N node

D die/chip



C outer-level cache group

M ccNUMA domain

Example: **`likwid-pin`** with Intel OpenMP

DEMO

Running the STREAM benchmark with **`likwid-pin`**:

```
$ likwid-pin -c S0:0-3 ./stream
----------------------------------------------
 Double precision appears to have 16 digits of accuracy
 Assuming 8 bytes per DOUBLE PRECISION word
----------------------------------------------
 Array size =   20000000
 Offset     =         32
 The total memory requirement is  457 MB
 You are running each test  10 times
 --
 The *best* time for each test is used
 *EXCLUDING* the first and last iterations
[pthread wrapper]
[pthread wrapper] MAIN -> 0
[pthread wrapper] PIN_MASK: 0->1  1->2  2->3
[pthread wrapper] SKIP MASK: 0x0
        threadid 47308666070912 -> core 1 - OK
        threadid 47308670273536 -> core 2 - OK
        threadid 47308674476160 -> core 3 - OK

  [... rest of STREAM output omitted ...]
```

Main PID always pinned

Some threads might need to be skipped (e.g. runtime threads)

Pin all spawned threads in turn

likwid-perfctr

- Commandline application for hardware performance monitoring
- Operating modes
  - Wrapper - coarse profile of whole application
  - Stethoscope - system monitoring
  - Timeline - time-based sampling
  - MarkerAPI - code instrumentation
- Different backends
  - direct access with root privileges
  - accessDaemon mode with privilege-escalation daemon
  - `perf_event` with reduced feature set in other LIKWID tools

Build configuration

- Almost all hardware events supported
- Pre-configured derived metric groups (performance groups)

likwid-perfctr

- Where should be measured?
  - **-c <intlist>**: measure on these HW threads
  - **-C <intlist>**: measure AND pin on/to these HW threads
- What should be measured?
  - **-g <eventlist>**
  - **-g <group>**
- Select operating mode
  - **-m**: activate MarkerAPI mode
  - **-t <time>**: Timeline mode
  - **-S <time>**: Stethoscope mode

-e for list of all events
<event>:<counter>(:opts)

-a for list of all groups

```
CLOCK: Clock frequency of cores
FLOPS_DP: Double Precision MFlops/s
FLOPS_SP: Single Precision MFlops/s
MEM: Main memory bandwidth in MBytes/s
```

- **likwid-perfctr -C 0,1 -g L2 <application> <args>**

`likwid-perfctr` events

DEMO

```
--------------------------------------------------------------------------------
CPU name:Intel(R) Core(TM) i5-8259U CPU @ 2.30GHz
CPU type:Intel Kabylake processor
CPU clock:    2.30 GHz
--------------------------------------------------------------------------------
$ likwid-perfctr -e
This architecture has 33 counters.
Counter tags(name, type<, options>):
FIXC0, Fixed counters, KERNEL|ANYTHREAD
FIXC1, Fixed counters, KERNEL|ANYTHREAD
FIXC2, Fixed counters, KERNEL|ANYTHREAD
PMC0, Core-local general purpose counters, EDGEDETECT|THRESHOLD|INVERT|KERNEL|ANYTHREAD|IN_TRANSACTION
PMC1, Core-local general purpose counters, EDGEDETECT|THRESHOLD|INVERT|KERNEL|ANYTHREAD|IN_TRANSACTION
[…]

This architecture has 445 events.
Event tags (tag, id, umask, counters<, options>):
INSTR_RETIRED_ANY, 0x0, 0x0, FIXC0
CPU_CLK_UNHALTED_CORE, 0x0, 0x0, FIXC1
CPU_CLK_UNHALTED_REF, 0x0, 0x0, FIXC2
ICACHE_16B_IFDATA_STALL, 0x80, 0x4, PMC
ICACHE_64B_IFTAG_HIT, 0x83, 0x1, PMC
```

Counters and events are architecture-dependent

Counter & event option description in [LIKWID wiki]()

Can only be measured on **`FIXCx`**

Can be measured on any **`PMC`** counter

`likwid-perfctr` performance groups

DEMO

```
$ likwid-perfctr -a
Group name            Description
-----------------------------------------------------------
UOPS                  UOPs execution info
L2                    L2 cache bandwidth in MBytes/s
CYCLE_STALLS          Cycle Activities (Stalls)
TLB_INSTR             L1 Instruction TLB miss rate/ratio
L3CACHE               L3 cache miss rate/ratio
ICACHE                Instruction cache miss rate/ratio
[…]
```

`likwid-perfctr` in wrapper mode

DEMO

```
$ likwid-perfctr -C S1:0-3 -g L2 ./a.out
--------------------------------------------------------------------------------
CPU name:       Intel(R) Xeon(R) CPU E5-2695 v3 @ 2.30GHz […]
--------------------------------------------------------------------------------
<<<< PROGRAM OUTPUT >>>>
--------------------------------------------------------------------------------
Group 1: L2
+-----------------------+---------+------------+------------+------------+------------+
|         Event         | Counter |  Core 14   |  Core 15   |  Core 16   |  Core 17   |
+-----------------------+---------+------------+------------+------------+------------+
|   INSTR_RETIRED_ANY   |  FIXC0  | 1298031144 | 1965945005 | 1854182290 | 1862521357 |
| CPU_CLK_UNHALTED_CORE |  FIXC1  | 2353698512 | 2894134935 | 2894645261 | 2895023739 |
|  CPU_CLK_UNHALTED_REF |  FIXC2  | 2057044629 | 2534405765 | 2535218217 | 2535560434 |
|    L1D_REPLACEMENT    |  PMC0   |  212900444 |  200544877 |  200389272 |  200387671 |
|    L2_TRANS_L1D_WB    |  PMC1   |  112464863 |  99931184  |  99982371  |  99976697  |
|     ICACHE_MISSES     |  PMC2   |    21265   |    26233   |    12646   |    12363   |
+-----------------------+---------+------------+------------+------------+------------+
[… statistics output omitted …]
+---------------------------------+------------+------------+------------+------------+
|             Metric              |  Core 14   |  Core 15   |  Core 16   |  Core 17   |
+---------------------------------+------------+------------+------------+------------+
|       Runtime (RDTSC) [s]       |   1.1314   |   1.1314   |   1.1314   |   1.1314   |
|      Runtime unhalted [s]       |   1.0234   |   1.2583   |   1.2586   |   1.2587   |
|           Clock [MHz]           | 2631.6699  | 2626.4367  | 2626.0579  | 2626.0468  |
|               CPI               |   1.8133   |   1.4721   |   1.5611   |   1.5544   |
|   L2D load bandwidth [MBytes/s] | 12042.7388 | 11343.8446 | 11335.0428 | 11334.9523 |
|   L2D load data volume [GBytes] |  13.6256   |  12.8349   |  12.8249   |  12.8248   |
|  L2D evict bandwidth [MBytes/s] | 6361.5883  | 5652.6192  | 5655.5146  | 5655.1937  |
|  L2D evict data volume [GBytes] |   7.1978   |   6.3956   |   6.3989   |   6.3985   |
|     L2 bandwidth [MBytes/s]     | 18405.5299 | 16997.9477 | 16991.2728 | 16990.8453 |
|     L2 data volume [GBytes]     |  20.8247   |  19.2321   |  19.2246   |  19.2241   |
+---------------------------------+------------+------------+------------+------------+
```

Resolves to HW threads 14,15,16 and 17

Fixed-purpose events always measured if possible

Configured events (L2 group)

Derived metrics for L2 group

`likwid-perfctr` with MarkerAPI

- The MarkerAPI can restrict measurements to code regions
- The API only reads counters, configuration performed by **`likwid-perfctr`**
- Multiple named regions support, accumulation over multiple calls
- Inclusive and overlapping regions allowed

See LIKWID wiki for Fortran90 example

```
#include <likwid-marker.h>

LIKWID_MARKER_INIT;                    // must be called from serial region
. . .
LIKWID_MARKER_REGISTER("Compute") // register for each thread
. . .
LIKWID_MARKER_START("Compute");   // start markers for each thread
<code>
LIKWID_MARKER_STOP("Compute");     // stop markers for each thread
. . .
LIKWID_MARKER_CLOSE;                   // must be called from serial region
```

Before LIKWID 5 use **`likwid.h`**

- `$CC -DLIKWID_PERFMON -I/path/to/likwid-marker.h -L/path/to/liblikwid … -llikwid`
- `likwid-perfctr -C <intlist> -g <eventlist|group> -m ./a.out`

Pinning required!

Usage information

- Topo. entity specific units are only counted by one HW thread per entity

```
$ likwid-perfctr -C S0:0-1@S1:0-1 -g MEM ./a,out
+------------------------------------+------------+--------------+--------------+-------------+
|               Metric               | HWThread 0 |  HWThread 1  | HWThread 36  | HWThread 37 |
+------------------------------------+------------+--------------+--------------+-------------+
|    Memory bandwidth [MBytes/s]     |   328.1941 |            0 |     775.7109 |           0 |
|    Memory data volume [GBytes]     |     0.0014 |            0 |       0.0033 |           0 |
+------------------------------------+------------+--------------+--------------+-------------+
```

- Statistics table may contain non-useful data (uncore units)
- No knowledge about PIDs → count anything done by the HW thread(s) (except perf_event backend)
- Pinning recommended!
- Counter access is overhead!
  - Might disturb execution
  - Too short code regions return wrong results

Motivation for Microbenchmarking as a tool

- Isolate small kernels to:
  - Separate influences
  - Determine specific machine capabilities (light speed)
  - Gain experience about software/hardware interaction
  - Determine programming model overhead
  - …
- Possibilities:
  - Readymade benchmark collections (epcc OpenMP, IMB)
  - STREAM benchmark for memory bandwidth
  - Implement own benchmarks (difficult and error prone)
  - **`likwid-bench`** tool: Offers collection of benchmarks and framework for rapid development of assembly code kernels

The parallel vector triad benchmark - *A "swiss army knife" for microbenchmarking*

```
double striad_seq(double* restrict a, double* restrict b, double* restrict c,
double* restrict d, int N, int iter) {
    double S, E;
    S = getTimeStamp();
    for (int j = 0; j < iter; j++) {
#pragma vector aligned
        for (int i = 0; i < N; i++) {
            a[i] = b[i] + d[i] * c[i];
        }
        if (a[N/2] > 2000) printf("Ai = %f\n",a[N-1]);
    }
    E = getTimeStamp();
    return E-S;
}
```

All timing facilities have a distinct resolution. Repeat main loop.

Required to get optimal code with Intel compiler **`icc`**! New **`icx`** unclear

Keeps smarty-pants compilers from doing "clever" stuff

- Report performance for different **`N`**, choose **`iter`** so that accurate time measurement is possible
- This kernel is limited by data transfer performance for all memory levels on all architectures, ever!

A better way – use a microbenchmarking tool

- Microbenchmarking in high-level language is often difficult
- Solution: assembly-based microbenchmarking framework
  - e.g., **`likwid-bench`**

```
$ likwid-bench -t triad_avx512_fma -W S0:28kB:1
```

benchmark type
topological entity (see likwid-pin)
working set
# of threads

- LIKWID MarkerAPI integrated
  `likwid-perfctr -C <MASK> -g <GROUP> -m likwid-bench …`
- Other recommendation: [**`nanobench`**]

Example: `likwid-bench`

DEMO

```
$ likwid-bench -t triad_avx512_fma -W N:2GB:2:1:2
Allocate: Process running on hwthread 0 (Domain N) - Vector length 62499968/499999744 Offset 0 Alignment 512
Initialization: Each thread in domain initializes its own stream chunks
--------------------------------------------------------------------------------
LIKWID MICRO BENCHMARK
Test: triad_avx512_fma
--------------------------------------------------------------------------------
Using 1 work groups
Using 2 threads
--------------------------------------------------------------------------------
Running without Marker API. Activate Marker API with -m on commandline.
--------------------------------------------------------------------------------
Group: 0 Thread 0 Global Thread 0 running on hwthread 0 - Vector length 31249984 Offset 0
Group: 0 Thread 1 Global Thread 1 running on hwthread 1 - Vector length 31249984 Offset 31249984
--------------------------------------------------------------------------------
Cycles:                         2977073662
CPU Clock:                      2593891829
Cycle Clock:                    2593891829
Time:                           1.147725e+00 sec
Iterations:                     32
Iterations per thread:          16
Inner loop executions:          976562
Size (Byte):                    1999998976
Size per thread:                999999488
Number of Flops:                1999998976
MFlops/s:                       1742.58
Data volume (Byte):             31999983616
MByte/s:                        27881.24
Cycles per update:              2.977075
Cycles per cacheline:           23.816601
Loads per update:               3
Stores per update:              1
Load bytes per element:         24
Store bytes per elem.:          8
Load/store ratio:               3.00
Instructions:                   593749712
UOPs:                           812499584
```

Questions?