Multicore Performance and Tools

Part 1: Topology, affinity control, clock speed
Tools for Node-level Performance Engineering

- **Node Information**
  /proc/cpuinfo, numactl, hwloc, `likwid-topology`, likwid-powermeter

- **Affinity control** and data placement
  OpenMP and MPI runtime environments, hwloc, numactl, `likwid-pin`

- **Runtime Profiling**
  Compilers, gprof, perf, HPC Toolkit, Intel Amplifier, …

- **Performance Analysis**
  Intel VTune, `likwid-perfctr`, PAPI-based tools, HPC Toolkit, Linux perf

- **Microbenchmarking**
  STREAM, `likwid-bench`, Imbench, uarch-bench
LIKWID performance tools

LIKWID tool suite:

Like I Knew What I’m Doing

Open source tool collection (developed at RRZE):


https://github.com/RRZE-HPC/likwid

https://youtu.be/6uFl1HPq-88
Reporting topology

likwid-topology

https://youtu.be/mxMWjNe73SI
Output of `likwid-topology -g` on one node of Intel Haswell-EP

---

CPU name: Intel(R) Xeon(R) CPU E5-2695 v3 @ 2.30GHz
CPU type: Intel Xeon Haswell EN/EP/EX processor
CPU stepping: 2

********************************************************************************

Hardware Thread Topology
********************************************************************************

Sockets: 2
Cores per socket: 14
Threads per core: 2

<table>
<thead>
<tr>
<th>HWThread</th>
<th>Thread</th>
<th>Core</th>
<th>Socket</th>
<th>Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>*</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>*</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>*</td>
</tr>
<tr>
<td>44</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>*</td>
</tr>
</tbody>
</table>

Socket 0: ( 0 28 1 29 2 30 3 31 4 32 5 33 6 34 7 35 8 36 9 37 10 38 11 39 12 40 13 41 )
Socket 1: ( 14 42 15 43 16 44 17 45 18 46 19 47 20 48 21 49 22 50 23 51 24 52 25 53 26 54 27 55 )

********************************************************************************

Cache Topology
********************************************************************************

Level: 1
Size: 32 kB
Cache groups: ( 0 28 ) ( 1 29 ) ( 2 30 ) ( 3 31 ) ( 4 32 ) ( 5 33 ) ( 6 34 ) ( 7 35 ) ( 8 36 ) ( 9 37 ) ( 10 38 ) ( 11 39 ) ( 12 40 ) ( 13 41 ) ( 14 42 ) ( 15 43 ) ( 16 44 ) ( 17 45 ) ( 18 46 ) ( 19 47 ) ( 20 48 ) ( 21 49 ) ( 22 50 ) ( 23 51 ) ( 24 52 ) ( 25 53 ) ( 26 54 ) ( 27 55 )

Level: 2
Size: 256 kB
Cache groups: ( 0 28 ) ( 1 29 ) ( 2 30 ) ( 3 31 ) ( 4 32 ) ( 5 33 ) ( 6 34 ) ( 7 35 ) ( 8 36 ) ( 9 37 ) ( 10 38 ) ( 11 39 ) ( 12 40 ) ( 13 41 ) ( 14 42 ) ( 15 43 ) ( 16 44 ) ( 17 45 ) ( 18 46 ) ( 19 47 ) ( 20 48 ) ( 21 49 ) ( 22 50 ) ( 23 51 ) ( 24 52 ) ( 25 53 ) ( 26 54 ) ( 27 55 )

Level: 3
Size: 17 MB
Cache groups: ( 0 28 1 29 2 30 3 31 4 32 5 33 6 34 ) ( 7 35 8 36 9 37 10 38 11 39 12 40 13 41 ) ( 14 42 15 43 16 44 17 45 18 46 19 47 20 48 ) ( 21 49 22 50 23 51 24 52 25 53 26 54 27 55 )
Output of `likwid-topology` continued

NUMA Topology

NUMA domains: 4

---

Domain: 0
Processors: ( 0 28 1 29 2 30 3 31 4 32 5 33 6 34 )
Distances: 10 21 31 31
Free memory: 13292.9 MB
Total memory: 15941.7 MB

---

Domain: 1
Processors: ( 7 35 8 36 9 37 10 38 11 39 12 40 13 41 )
Distances: 21 10 31 31
Free memory: 13514 MB
Total memory: 16126.4 MB

---

Domain: 2
Processors: ( 14 42 15 43 16 44 17 45 18 46 19 47 20 48 )
Distances: 31 31 10 21
Free memory: 15025.6 MB
Total memory: 16126.4 MB

---

Domain: 3
Processors: ( 21 49 22 50 23 51 24 52 25 53 26 54 27 55 )
Distances: 31 31 21 10
Free memory: 15488.9 MB
Total memory: 16126 MB
Graphical Topology
********************************************************************************
Cluster on Die (CoD) mode and SMT enabled!
Enforcing thread/process affinity under the Linux OS

likwid-pin  
https://youtu.be/PSJKNQaqwB0
STREAM benchmark on 16-core Sandy Bridge
Anarchy vs. thread pinning

There are several reasons for caring about affinity:

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

OpenMP-parallel
\[ A(\cdot) = B(\cdot) + s \cdot C(\cdot) \]
Vector triad on dual-socket 18-core node

Filling cores from left to right ("compact" pinning)

Filling both sockets simultaneously ("scattered" or "spread" pinning)
More thread/process affinity ("pinning") options

- Highly OS-dependent system calls
  But available on all systems
  
  Linux: `sched_setaffinity()`
  Windows: `SetThreadAffinityMask()`


- Support for "semi-automatic" pinning
  All modern compilers with OpenMP support
  Generic Linux: `taskset`, `numactl`, `likwid-pin` (see below)
  OpenMP 4.0 (`OMP_PLACES`, `OMP_PROC_BIND`)
  Slurm Batch scheduler

- Affinity awareness in MPI libraries
  OpenMPI
  Intel MPI …

[https://youtu.be/IKW0kRLnhyc](https://youtu.be/IKW0kRLnhyc)
Overview 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

Simple usage with logical IDs (“thread groups expressions”):

$ likwid-pin -c S0:0-7 ./myApp params
$ likwid-pin -c C1:0-2 ./myApp params
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 HWthreads sharing a topological entity (e.g. socket or shared cache)
- A **thread domain** is defined by a single **character + index**

- Example for likwid-pin:
  
  ```
  $ likwid-pin -c S1:0-3 ./a.out
  ```

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

Physical cores first!
LIKWARD: Currently available thread domains

Possible unit prefixes

**N** node

**S** socket

**M** NUMA domain

**C** outer level cache group

Default if `–c` is not specified!
Advanced options for pinning: Expression syntax

- The Expression syntax is more powerful in situations where the pin mask would be very long or clumsy

Compact pinning (counting through HW threads):
$ likwid-pin -c E:<thread domain>:\n <number of threads>\n [:<chunk size>:<stride>] ...

Scattered pinning across all domains of the designated type:
$ likwid-pin -c <domaintype>:scatter

- Examples:
  $ likwid-pin -c E:N:8:1:2 ...
  $ likwid-pin -c E:N:120:2:4 ...

- Scatter across all NUMA domains:
  $ likwid-pin -c M:scatter

"Compact" placement!
OMP_PLACES and Thread Affinity

Processor: smallest entity able to run a thread or task (hardware thread)
Place: one or more processors → thread pinning is done place by place
Free migration of the threads on a place between the processors of that place.

<table>
<thead>
<tr>
<th>OMP_PLACES</th>
<th>Place ==</th>
</tr>
</thead>
<tbody>
<tr>
<td>threads</td>
<td>Hardware thread (hyper-thread)</td>
</tr>
<tr>
<td>cores</td>
<td>All HW threads of a single core</td>
</tr>
<tr>
<td>sockets</td>
<td>All HW threads of a socket</td>
</tr>
<tr>
<td>abstract_name(num_places)</td>
<td>Restrict # of places available</td>
</tr>
</tbody>
</table>

Or use explicit numbering, e.g. 8 places, each consisting of 4 processors:
- OMP_PLACES="{0,1,2,3},{4,5,6,7},{8,9,10,11}, ... {28,29,30,31}"
- OMP_PLACES="{0:4},{4:4},{8:4}, ... {28:4}"
- OMP_PLACES="{0:4}:8:4"

Caveat: Actual behavior is implementation defined!
OMP_PROC_BIND variable / proc_bind() clause

Determines how places are used for pinning:

<table>
<thead>
<tr>
<th>OMP_PROC_BIND</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FALSE</td>
<td>Affinity disabled</td>
</tr>
<tr>
<td>TRUE</td>
<td>Affinity enabled, implementation defined strategy</td>
</tr>
<tr>
<td>CLOSE</td>
<td>Threads bind to consecutive places</td>
</tr>
<tr>
<td>SPREAD</td>
<td>Threads are evenly scattered among places</td>
</tr>
<tr>
<td>MASTER</td>
<td>Threads bind to the same place as the master thread that was running before the parallel region was entered</td>
</tr>
</tbody>
</table>

If there are more threads than places, consecutive threads are put into individual places ("balanced")
Some simple OMP_PLACES examples

Intel Xeon w/ SMT, 2x10 cores, 1 thread per physical core, fill 1 socket
OMP_NUM_THREADS=10
OMP_PLACES=cores
OMP_PROC_BIND=close

Always prefer abstract places instead of HW thread IDs!

Intel Xeon Phi with 72 cores,
32 cores to be used, 2 threads per physical core
OMP_NUM_THREADS=64
OMP_PLACES=cores(32)
OMP_PROC_BIND=close  # spread will also do

Intel Xeon, 2 sockets, 4 threads per socket (no binding within socket!)
OMP_NUM_THREADS=8
OMP_PLACES=sockets
OMP_PROC_BIND=close  # spread will also do

Intel Xeon, 2 sockets, 4 threads per socket, binding to cores
OMP_NUM_THREADS=8
OMP_PLACES=cores
OMP_PROC_BIND=spread
MPI startup and hybrid pinning: \texttt{likwid-mpirun}

- How do you manage affinity with MPI or hybrid MPI/threading?
- In the long run a unified standard is needed
- Till then, \texttt{likwid-mpirun} provides a portable/flexible solution
- The examples here are for Intel MPI/OpenMP programs, but are also applicable to other threading models

Pure MPI:

\texttt{$likwid-mpirun$ -np 16 -nperdomain S:2 ./a.out}

Hybrid:

\texttt{$likwid-mpirun$ -np 16 -pin S0:0,1_S1:0,1 ./a.out}
likwid-mpirun 1 MPI process per socket

$ likwid-mpirun -np 4 -pin S0:0-5_S1:0-5 ./a.out

Intel MPI+compiler:
OMP_NUM_THREADS=6 mpirun -ppn 2 -np 4 \\ 
   -env I_MPI_PIN_DOMAIN socket -env KMP_AFFINITY scatter ./a.out
Clock speed under the Linux OS

Turbo steps and likwid-powermeter
likwid-setFrequencies
Which clock speed steps are there?

Uses the Intel RAPL interface (Sandy Bridge++)

$ likwid-powermeter -i

---

CPU name: Intel(R) Xeon(R) CPU E5-2695 v3 @ 2.30GHz
CPU type: Intel Xeon Haswell EN/EP/EX processor
CPU clock: 2.30 GHz

Base clock: 2300.00 MHz
Minimal clock: 1200.00 MHz

Turbo Boost Steps:
C0 3300.00 MHz
C1 3300.00 MHz
C2 3100.00 MHz
C3 3000.00 MHz
C4 2900.00 MHz
[...]
C13 2800.00 MHz

---

Info for RAPL domain PKG:
Thermal Spec Power: 120 Watt
Minimum Power: 70 Watt
Maximum Power: 120 Watt
Maximum Time Window: 46848 micro sec

Info for RAPL domain DRAM:
Thermal Spec Power: 21.5 Watt
Minimum Power: 5.75 Watt
Maximum Power: 21.5 Watt
Maximum Time Window: 44896 micro sec

Note: AVX code on HSW+ may execute even slower than base freq.

likwid-powermeter can also measure energy consumption, but likwid-perfctr can do it better (see later)
Setting the clock frequency

- The “Turbo Mode” feature makes reliable benchmarking harder
  CPU can change clock speed at its own discretion
- Clock speed reduction may save a lot of energy
- So how do we set the clock speed?
  → LIKWID to the rescue!

$ likwid-setFrequencies -l
Available frequencies:
1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2 2.1 2.2 2.3
$ likwid-setFrequencies -p
Current CPU frequencies:
CPU 0: governor performance min/cur/max 2.3/2.301/2.301 GHz Turbo 1
CPU 1: governor performance min/cur/max 2.3/2.301/2.301 GHz Turbo 1
CPU 2: governor performance min/cur/max 2.3/2.301/2.301 GHz Turbo 1
CPU 3: governor performance min/cur/max 2.3/2.301/2.301 GHz Turbo 1
[...]
$ likwid-setFrequencies -f 2.0  # min=max=2.0
[...]
$ likwid-setFrequencies -turbo 0  # turbo off

Turbo mode