Multicore Performance and Tools

Part 1: Topology, affinity, 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**, lmbench, uarch-bench*

LIKWID performance tools

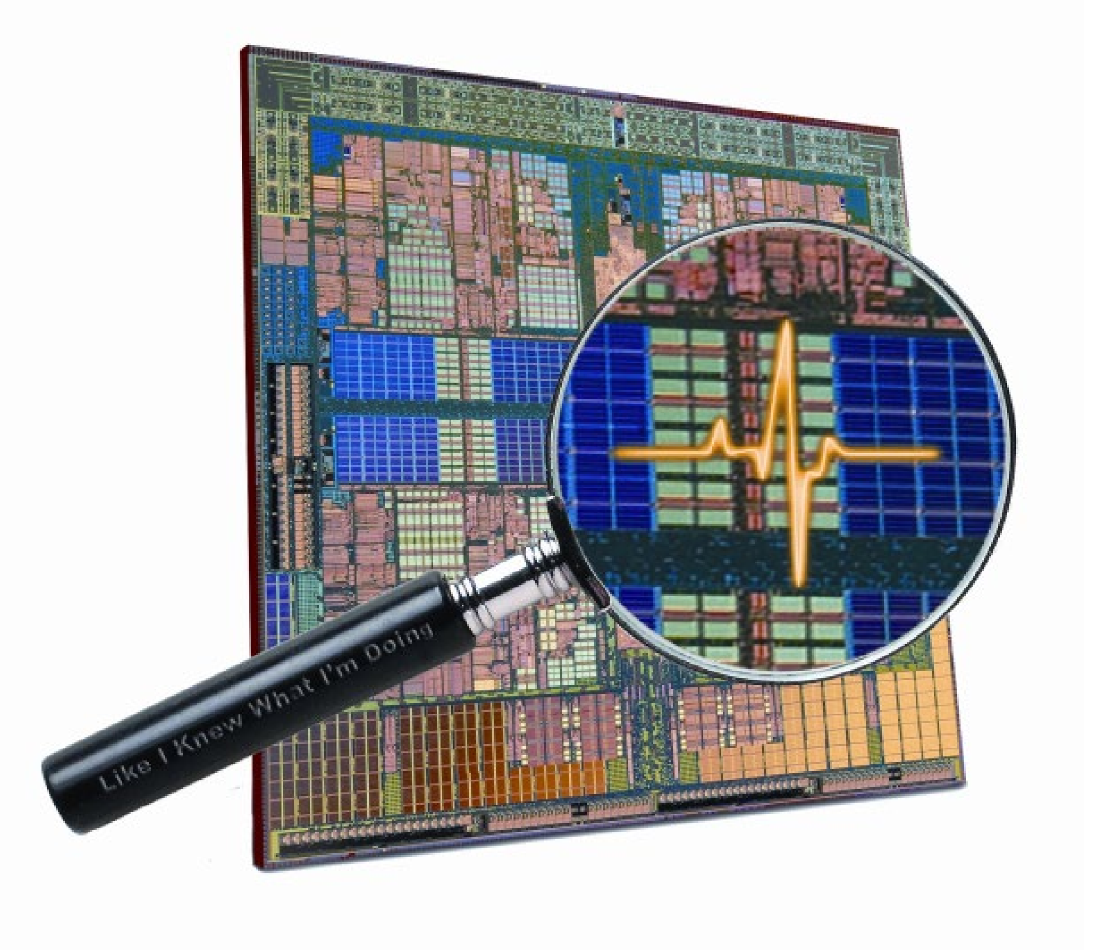
LIKWID tool suite:

Like
I
Knew
What
I’m
Doing

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

Open source tool collection
(developed at RRZE):

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



J. Treibig, G. Hager, G. Wellein: *LIKWID: A lightweight performance-oriented tool suite for x86 multicore environments.* PSTI2010, Sep 13-16, 2010, San Diego, CA. DOI: 10.1109/ICPPW.2010.38

LIKWID Tool Suite

- Command line tools for Linux:

  easy to install

  works with standard Linux kernel

  simple and clear to use

  supports most X86 CPUs

  (also ARMv8, POWER9 and
  Nvidia GPUs)

- Current 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

… some more

Reporting topology

likwid-topology

https://youtu.be/mxMWjNe73SI

Output of `likwid-topology`

on one node of Intel Icelake SP

```
$ 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             *
[…]
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 )
--------------------------------------------------------------------------------
********************************************************************************
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
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 )
--------------------------------------------------------------------------------
```

All OS hardware thread IDs

Output of `likwid-topology` continued

```
********************************************************************************
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
--------------------------------------------------------------------------------
```

Output similar to
`numactl --hardware`

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

Sub-NUMA clustering (SNC) enabled, SMT disabled!

Output of `likwid-topology -g`

Intel Haswell EP, 14 cores per socket
Cluster on Die (CoD) mode and SMT enabled!

```
********************************************************************************
Graphical Topology
********************************************************************************
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  | |
| +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ |
| +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ |
| |  32kB  | |  32kB  | |  32kB  | |  32kB  | |  32kB  | |  32kB  | |  32kB  | |  32kB  | |  32kB  | |  32kB  | |  32kB  | |  32kB  | |  32kB  | |  32kB  | |
| +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ |
| +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ |
| |  256kB | |  256kB | |  256kB | |  256kB | |  256kB | |  256kB | |  256kB | |  256kB | |  256kB | |  256kB | |  256kB | |  256kB | |  256kB | |  256kB | |
| +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ |
| +--------------------------------------------------------------------------+ +--------------------------------------------------------------------------+ |
| |                                   17MB                                   | |                                   17MB                                   | |
| +--------------------------------------------------------------------------+ +--------------------------------------------------------------------------+ |
+---------------------------------------------------------------------------------------------------------------------------------------------------------------+
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  | |
| +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ |
| +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ |
| |  32kB  | |  32kB  | |  32kB  | |  32kB  | |  32kB  | |  32kB  | |  32kB  | |  32kB  | |  32kB  | |  32kB  | |  32kB  | |  32kB  | |  32kB  | |  32kB  | |
| +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ |
| +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ |
| |  256kB | |  256kB | |  256kB | |  256kB | |  256kB | |  256kB | |  256kB | |  256kB | |  256kB | |  256kB | |  256kB | |  256kB | |  256kB | |  256kB | |
| +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ |
| +--------------------------------------------------------------------------+ +--------------------------------------------------------------------------+ |
| |                                   17MB                                   | |                                   17MB                                   | |
| +--------------------------------------------------------------------------+ +--------------------------------------------------------------------------+ |
+---------------------------------------------------------------------------------------------------------------------------------------------------------------+
```

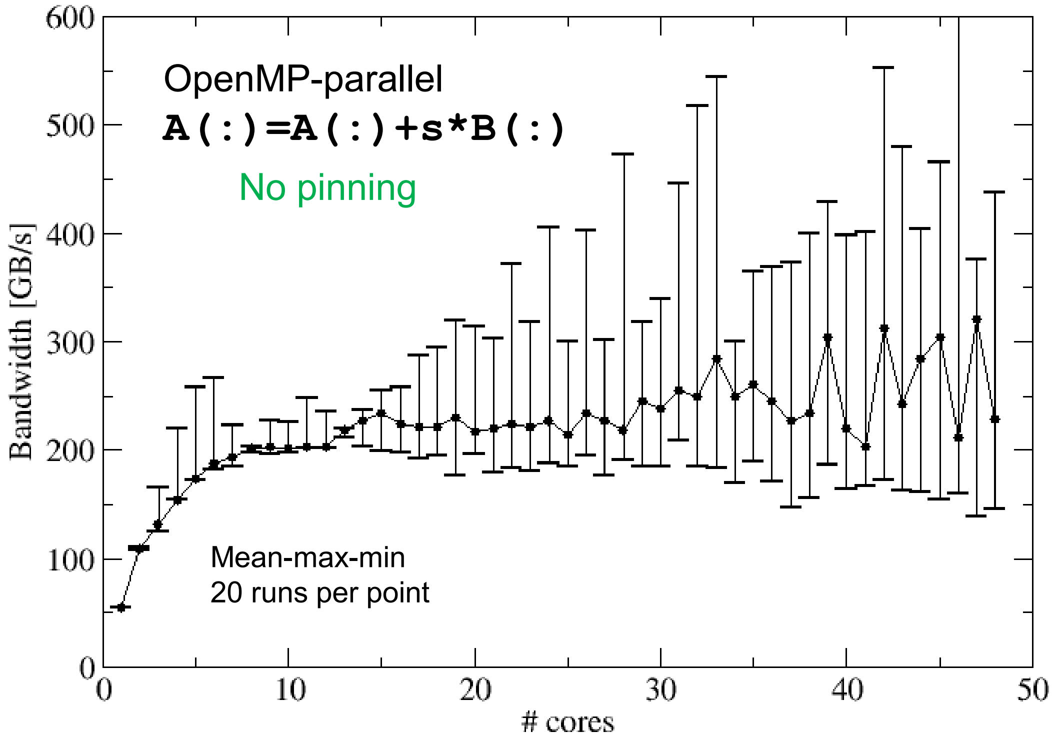
Enforcing thread/process affinity under the Linux OS

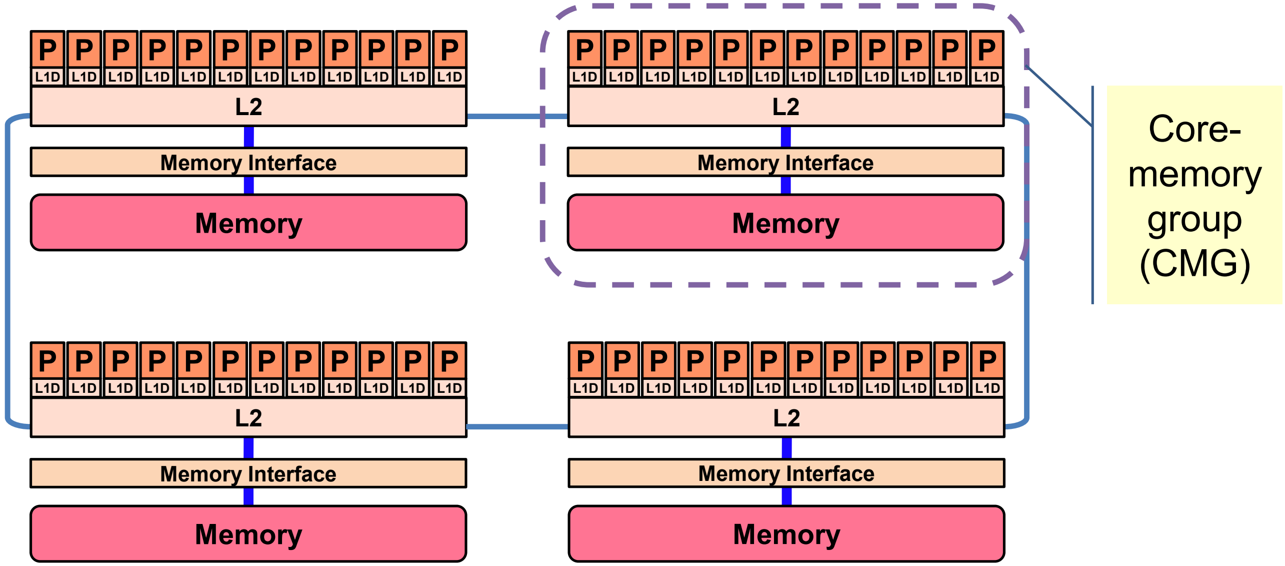
likwid-pin

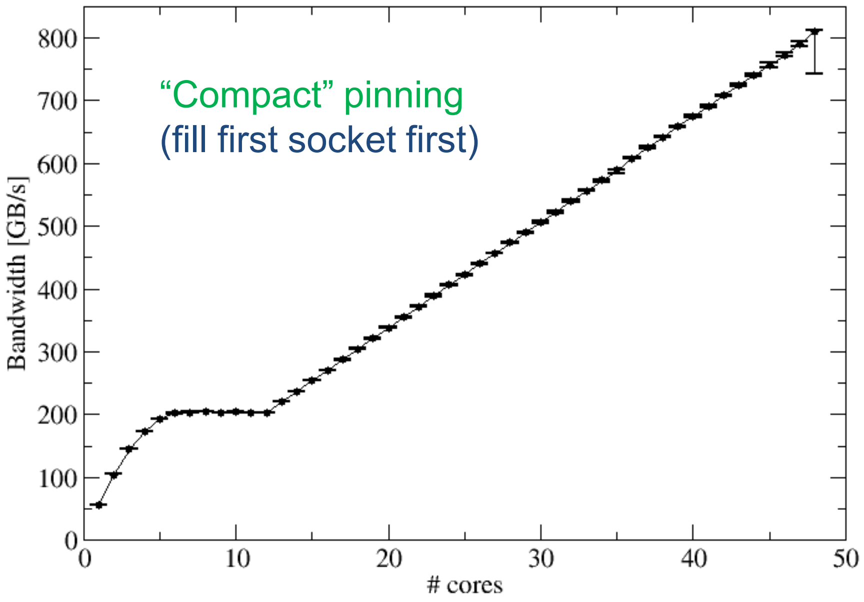
https://youtu.be/PSJKNQaqwB0

DAXPY test on A64FX

Anarchy vs. thread pinning



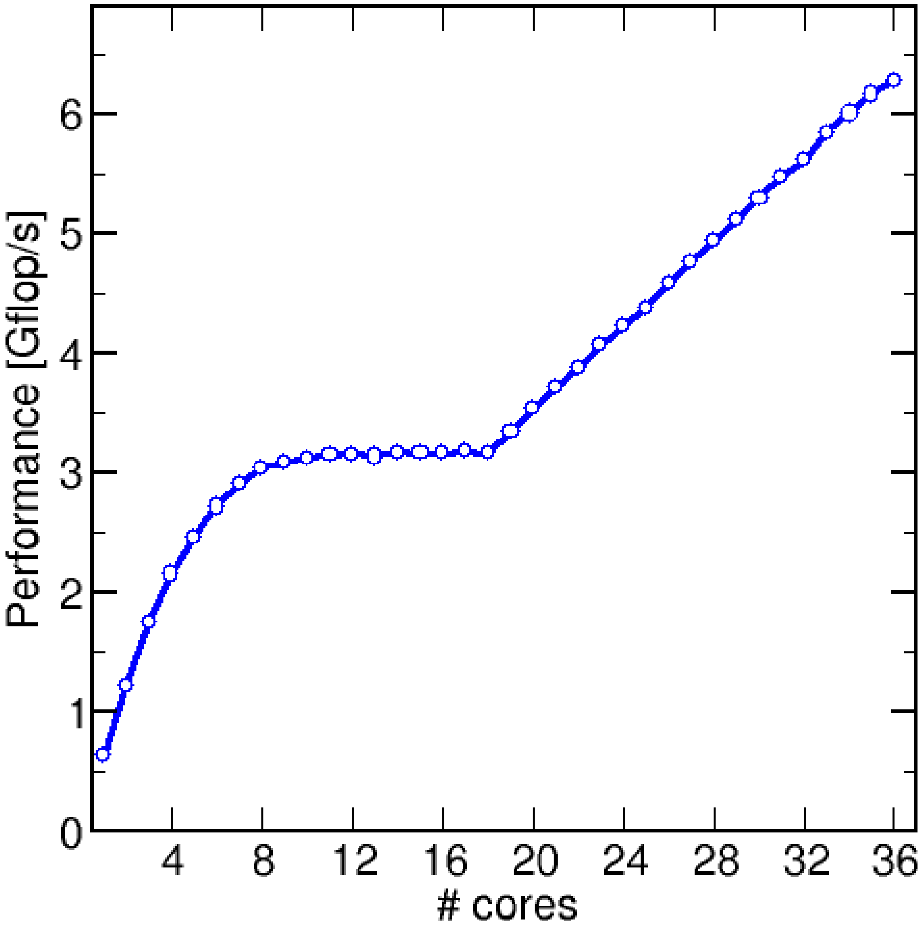




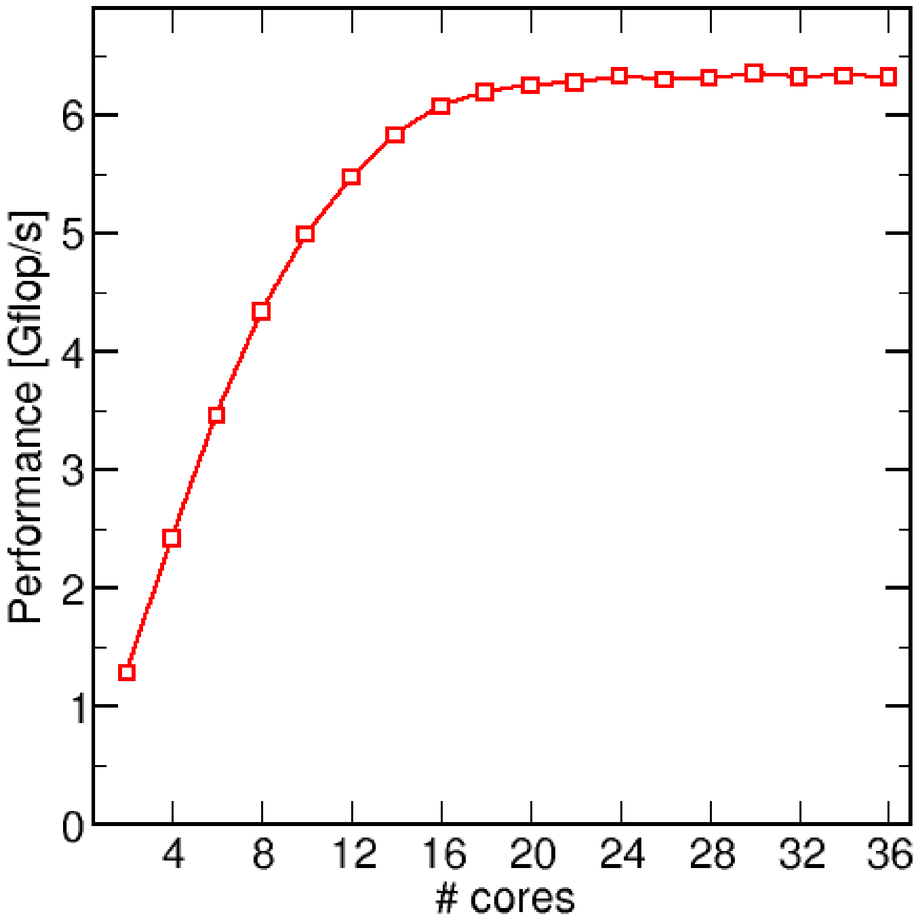
There are several reasons for caring about affinity:

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

Vector triad on 2x 18-core node: Compact vs. spread pinning

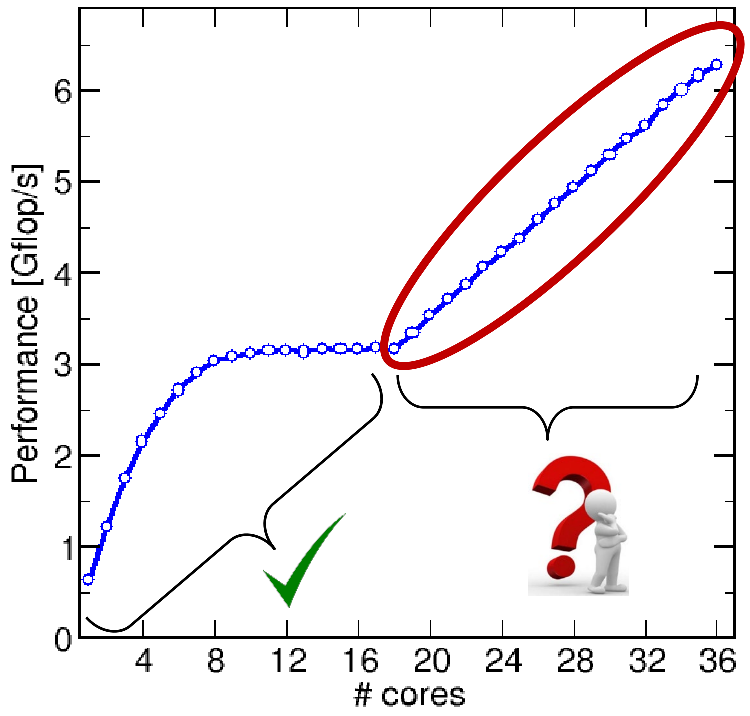


Filling cores from left to right
("compact" pinning)



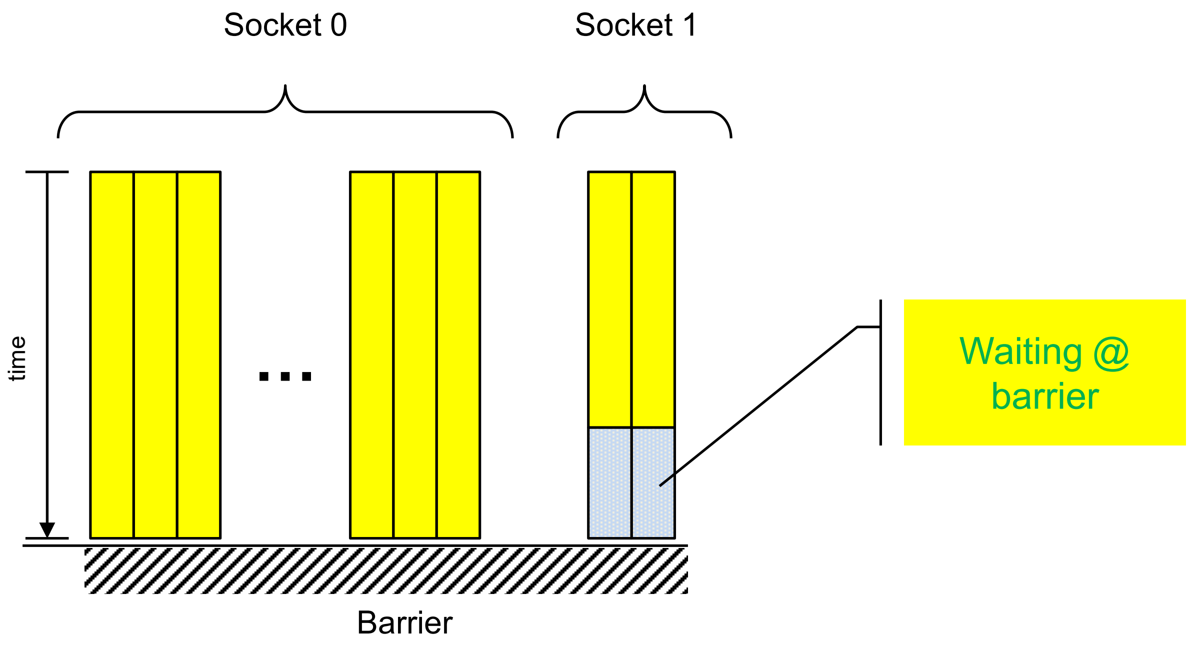
Filling both sockets simultaneously
("scattered" or "spread" pinning)

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

More thread/process affinity ("pinning") options

- Highly OS-dependent system calls but available on all systems
  - Linux: `sched_setaffinity()`
  - Windows: `SetThreadAffinityMask()`
  - OSX: `thread_policy_set()` (only L2Cache granularity)

- Hwloc project (http://www.open-mpi.de/projects/hwloc/)

- Support for "semi-automatic" pinning
  - All modern compilers with OpenMP support
    OpenMP 4.0 (**`OMP_PLACES`**, **`OMP_PROC_BIND`**)
  - CPUset reduction utils: **`taskset`** or **`numactl`**
  - Job scheduler like **`SLURM`**

- Affinity awareness in MPI libraries (OpenMPI, Intel MPI, …)

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 core IDs (“thread groups”):

```
$ 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 hardware threads (called “processors” in the OS) on a node
- The numbering is enforced at boot time by the BIOS
- LIKWID introduces **thread domains** consisting of HW 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`

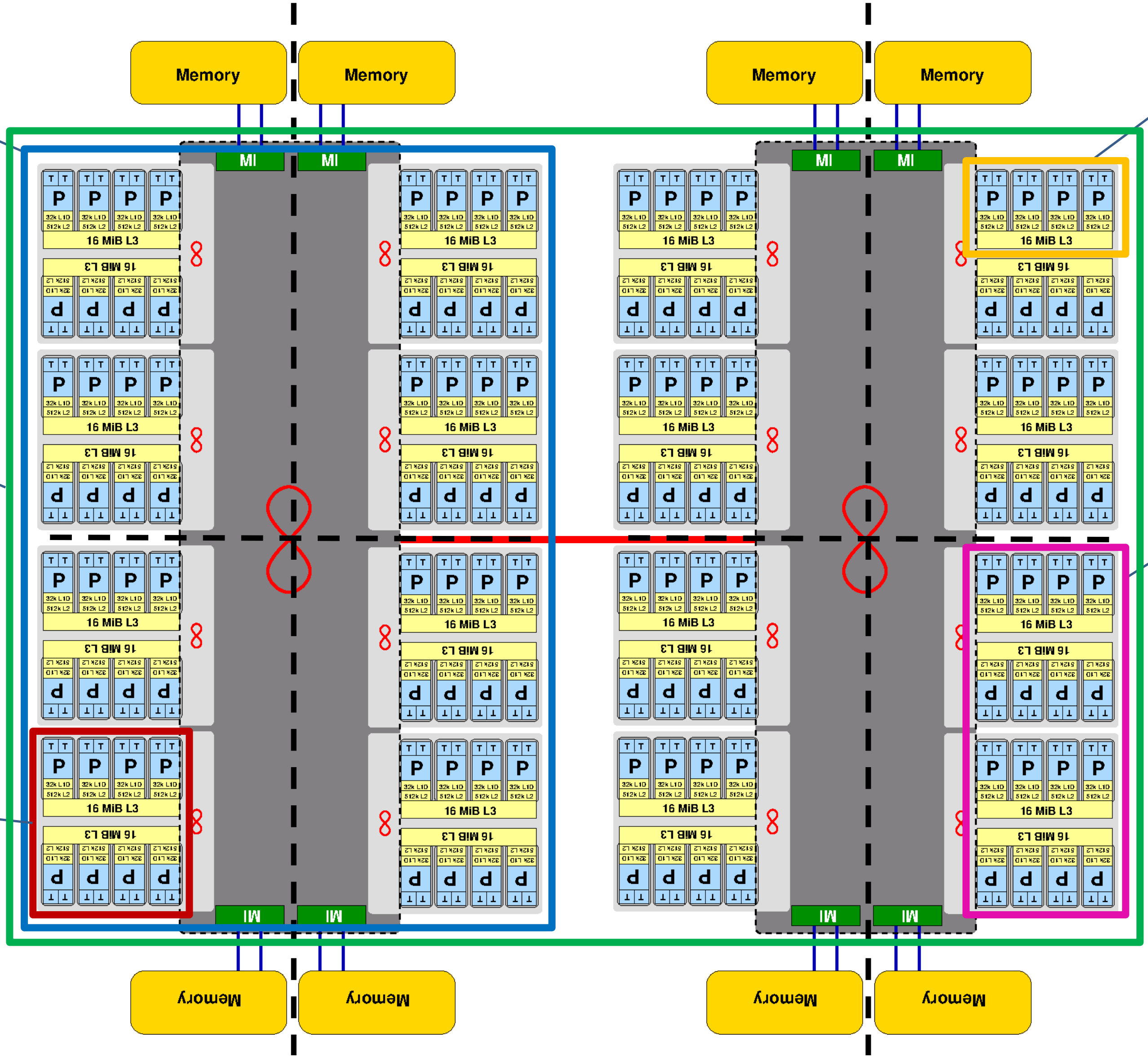
```
+-------------------------------------++-------------------------------------+
| +------+ +------+ +------+ +------+ || +------+ +------+ +------+ +------+ |
| | 0  8 | | 1  9 | | 2 10 | | 3 11 | || | 4 12 | | 5 13 | | 6 14 | | 7 15 | |
| +------+ +------+ +------+ +------+ || +------+ +------+ +------+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

Available thread domains/unit prefixes (as of LIKWID 5.2)

S socket

N node

D die/chip



C outer-level cache group

M ccNUMA domain

Advanced options for pinning: Expressions

- Expressions are more powerful in situations where the pin mask would be very long or clumsy

  Compact pinning (counting through hardware threads):

```
$ likwid-pin -c E:<thread domain>:\
    <number of threads>\
    [:<chunk size>:<stride>] ...
$ likwid-pin -c E:N:4
```

  "Compact" placement!

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

  Scattered pinning across all domains of the designated type:

```
$ likwid-pin -c <domaintype>:scatter
```

- Examples:

```
$ likwid-pin -c E:N:8:1:2 ...
```

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

- Scatter across all NUMA domains:

```
$ likwid-pin -c M:scatter
```

Example: `likwid-pin` with Intel OpenMP

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

Pin all spawned threads in turn

OMP_PLACES and Thread Affinity

optional

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.

abstract name

`OMP_PLACES`	Place ==
`threads`	Hardware thread (hyper-thread)
`cores`	All HW threads of a single core
`sockets`	All HW threads of a socket
`abstract_name(num_places)`	Restrict # of places available

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"`

<lower-bound>:<number of entries>[:<stride>]

Caveat: Actual behavior is implementation defined!

OMP_PROC_BIND variable / proc_bind() clause

optional

Determines how places are used for pinning:

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

If there are more threads than places, consecutive threads are put into individual places (“balanced”)

Some simple OMP_PLACES examples

optional

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: `likwid-mpirun`

- How do you manage affinity with MPI or hybrid MPI/threading?
- In the long run a unified standard is needed
- Till then, 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:

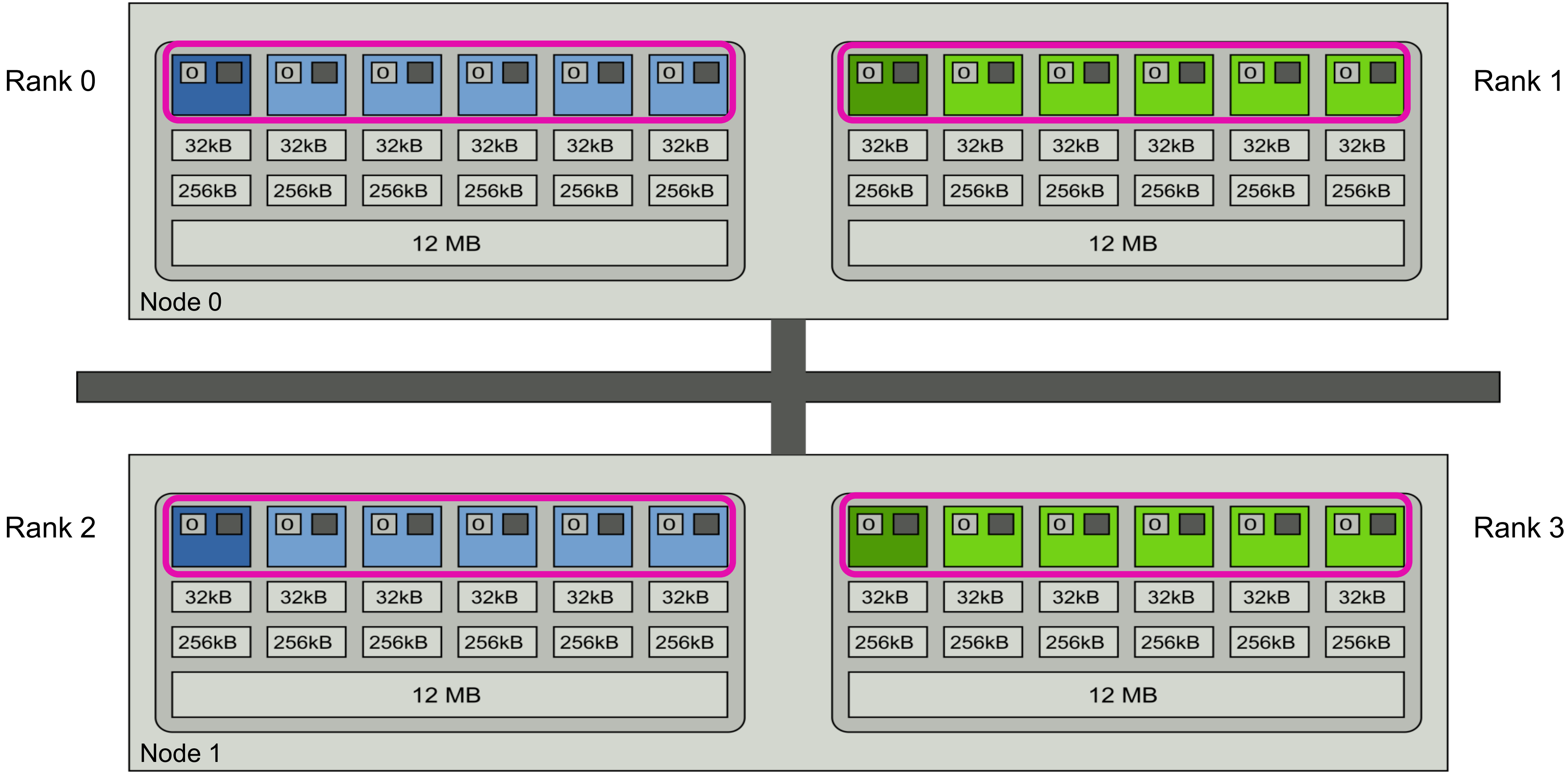
```
$ likwid-mpirun -np 16 -nperdomain S:2 ./a.out
```

Hybrid:

```
$ likwid-mpirun -np 16 -pin S0:0,1_S1:0,1 ./a.out
$ likwid-mpirun -np 16 -nperdomain S:1 -t 2 ./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
---------------------------------
```

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

```
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
```

`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

Uncore clock frequency

optional

Starting with Intel Haswell, the **Uncore** (L3, memory controller, UPI) sits in its own clock domain

```
$ likwid-setFrequencies -p
[...]
CPU 68: governor  performance min/cur/max 2.3/2.301/2.301 GHz Turbo 1
CPU 69: governor  performance min/cur/max 2.3/2.301/2.301 GHz Turbo 1
CPU 70: governor  performance min/cur/max 2.3/2.301/2.301 GHz Turbo 1
CPU 71: governor  performance min/cur/max 2.3/2.301/2.301 GHz Turbo 1

Current Uncore frequencies:
Socket 0: min/max 1.2/3.0 GHz
Socket 1: min/max 1.2/3.0 GHz

$ likwid-setFrequencies --umin 2.3 --umax 2.3
```

Uncore has considerable impact on power consumption

J. Hofmann et al.: *An analysis of core- and chip-level architectural features in four generations of Intel server processors*. Proc. ISC High Performance 2017. DOI: 10.1007/978-3-319-58667-0_16.

J. Hofmann et al.: *On the accuracy and usefulness of analytic energy models for contemporary multicore processors*. Proc. ISC High Performance 2018. DOI: 10.1007/978-3-319-92040-5_2