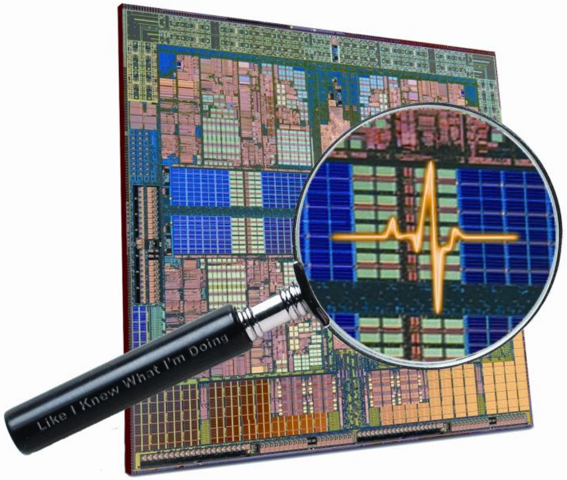
LIKWID:

Lightweight performance tools



J. Treibig

RRZE, University Erlangen

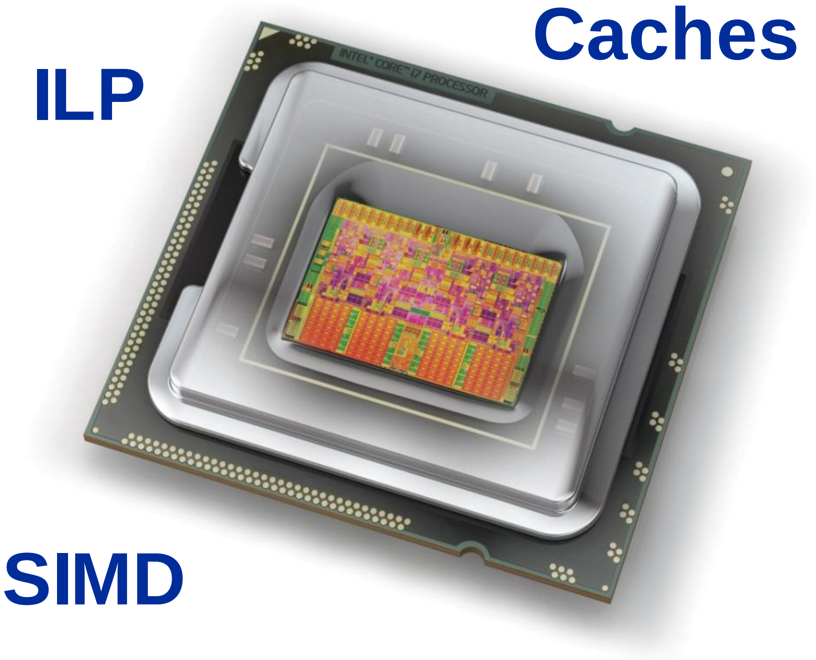
26.9.2011

Challenges

For high efficiency hardware aware programming is required.

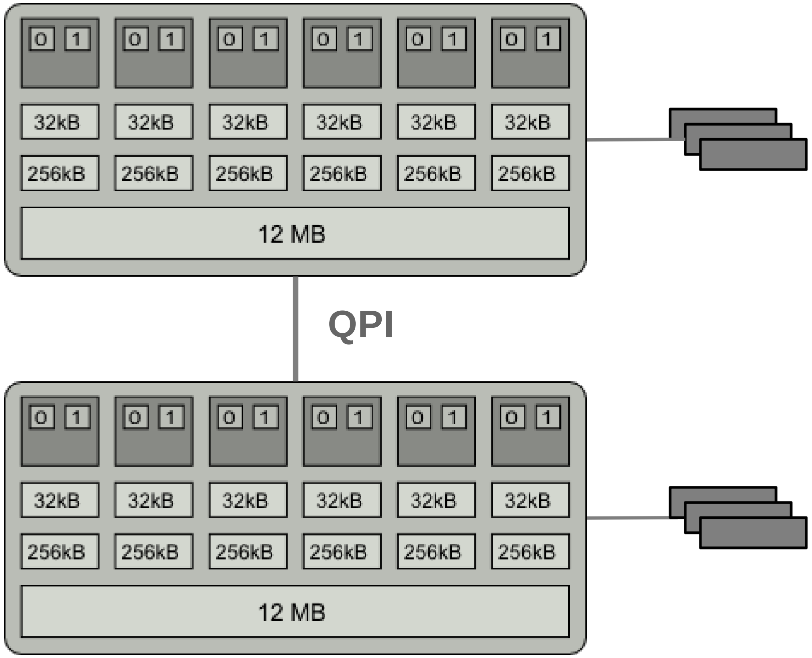
ILP

Caches



SIMD

NUMA



Multicore architectures add complex topologies on the thread and memory level.

Contribution

- **Lightweight command line tools for Linux**
- **Help to face the challenges without getting in the way**
- **Focus on X86 architecture**
- **Philosophy:**
  - Simple
  - Efficient
  - Portable
  - Extensible

Open source project (GPL v2):

http://code.google.com/p/likwid/

Why?

- **Question: There is tool XY? They can do the same thing.**

- **Possible answers:**
  - LIKWID has an unique feature set
  - LIKWID has NO external dependencies
  - LIKWID is easy to build and setup
  - LIKWID is just COOL (OK this is biased)

If you are still not convinced:

It is always good to have some competition.

Even in Open Source tools.

So try it and make your own opinion what suits your needs best.

Scenario 1: Hardware performance monitoring and Node performance characterisation

likwid-perfctr

likwid-perfscope

likwid-bench

likwid-perfctr

Probing performance behavior

- **A coarse overview of hardware performance monitoring data is often sufficient**
  - likwid-perfctr (similar to “perfex” on IRIX, “hpmcount” on AIX, “lipfpm” on Linux/Altix, “craypat” on Cray systems)
  - Simple end-to-end measurement of hardware performance metrics
  - Operating modes:
    - Wrapper
    - Stethoscope
    - Timeline
    - Marker API
  - Preconfigured and extensible metric groups, list with **`likwid-perfctr -a`**

```
BRANCH: Branch prediction miss rate/ratio
CACHE: Data cache miss rate/ratio
CLOCK: Clock of cores
DATA: Load to store ratio
FLOPS_DP: Double Precision MFlops/s
FLOPS_SP: Single Precision MFlops/s
FLOPS_X87: X87 MFlops/s
L2: L2 cache bandwidth in MBytes/s
L2CACHE: L2 cache miss rate/ratio
L3: L3 cache bandwidth in MBytes/s
L3CACHE: L3 cache miss rate/ratio
MEM: Main memory bandwidth in MBytes/s
TLB: TLB miss rate/ratio
```

likwid-perfctr

Example usage for Wrapper mode

```
$ env OMP_NUM_THREADS=4 likwid-perfctr -C N:0-3 -t intel -g FLOPS_DP  ./stream.exe
-------------------------------------------------------------
CPU type:       Intel Core Lynnfield processor
CPU clock:      2.93 GHz
-------------------------------------------------------------
Measuring group FLOPS_DP
-------------------------------------------------------------
YOUR PROGRAM OUTPUT
+-------------------------------------+-------------+-------------+-------------+-------------+
|                Event                |   core 0    |   core 1    |   core 2    |   core 3    |
+-------------------------------------+-------------+-------------+-------------+-------------+
|          INSTR_RETIRED_ANY          | 1.97463e+08 | 2.31001e+08 | 2.30963e+08 | 2.31885e+08 |
|        CPU_CLK_UNHALTED_CORE        | 9.56999e+08 | 9.58401e+08 | 9.58637e+08 | 9.57338e+08 |
|       FP_COMP_OPS_EXE_SSE_FP_PACKED | 4.00294e+07 | 3.08927e+07 | 3.08866e+07 | 3.08904e+07 |
|       FP_COMP_OPS_EXE_SSE_FP_SCALAR |     882     |      0      |      0      |      0      |
| FP_COMP_OPS_EXE_SSE_SINGLE_PRECISION|      0      |      0      |      0      |      0      |
| FP_COMP_OPS_EXE_SSE_DOUBLE_PRECISION| 4.00303e+07 | 3.08927e+07 | 3.08866e+07 | 3.08904e+07 |
+-------------------------------------+-------------+-------------+-------------+-------------+
+---------------------------+------------+-----------+-----------+-----------+
|          Metric           |   core 0   |  core 1   |  core 2   |  core 3   |
+---------------------------+------------+-----------+-----------+-----------+
|        Runtime [s]        |  0.326242  |  0.32672  | 0.326801  | 0.326358  |
|            CPI            |  4.84647   |  4.14891  | 4.15061   |  4.12849  |
| DP MFlops/s (DP assumed)  |  245.399   |  189.108  | 189.024   |  189.304  |
|      Packed MUOPS/s       |  122.698   |  94.554   | 94.5121   |  94.6519  |
|      Scalar MUOPS/s       | 0.00270351 |     0     |    0      |     0     |
|        SP MUOPS/s         |     0      |     0     |    0      |     0     |
|        DP MUOPS/s         |  122.701   |  94.554   | 94.5121   |  94.6519  |
+---------------------------+------------+-----------+-----------+-----------+
```

Always measured

Configured metrics (this group)

Derived metrics

likwid-perfctr
Stethoscope mode

- likwid-perfctr measures on core base and has no notion what runs on the cores

This enables to listen on what currently happens without any overhead:

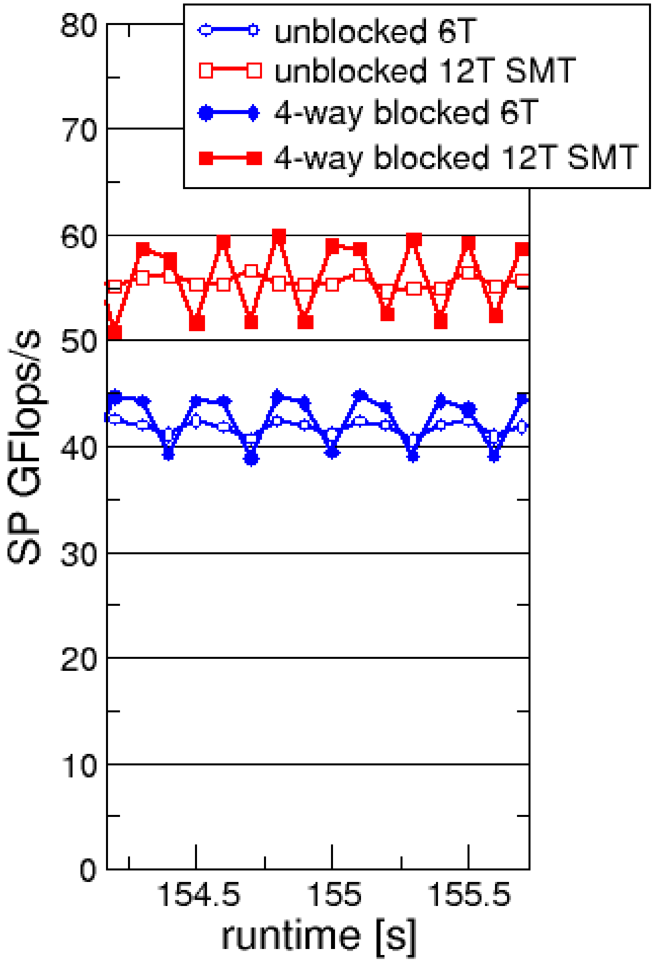
```
likwid-perfctr -c N:0-11 -g FLOPS_DP  sleep 10
```

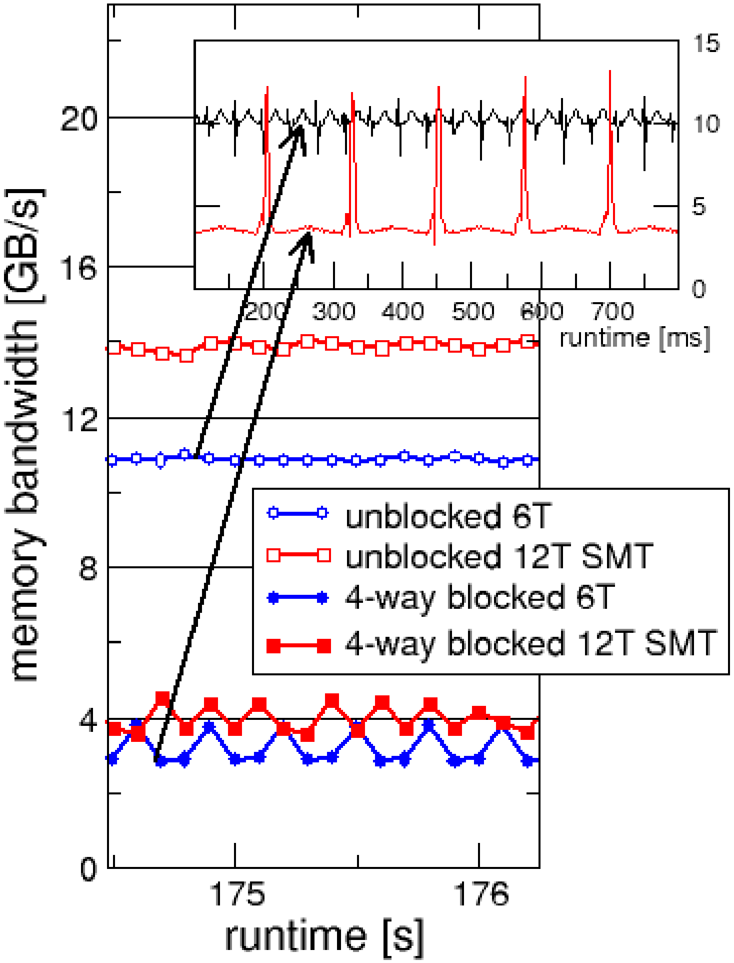
- It can be used as cluster/server monitoring tool
- A frequent use is to measure a certain part of a long running parallel application from outside

likwid-perfctr
Timeline mode

- **likwid-perfctr supports time resolved measurements of full node:**

```
likwid-perfctr -c N:0-11 -g MEM -d 50ms  > out.txt
```

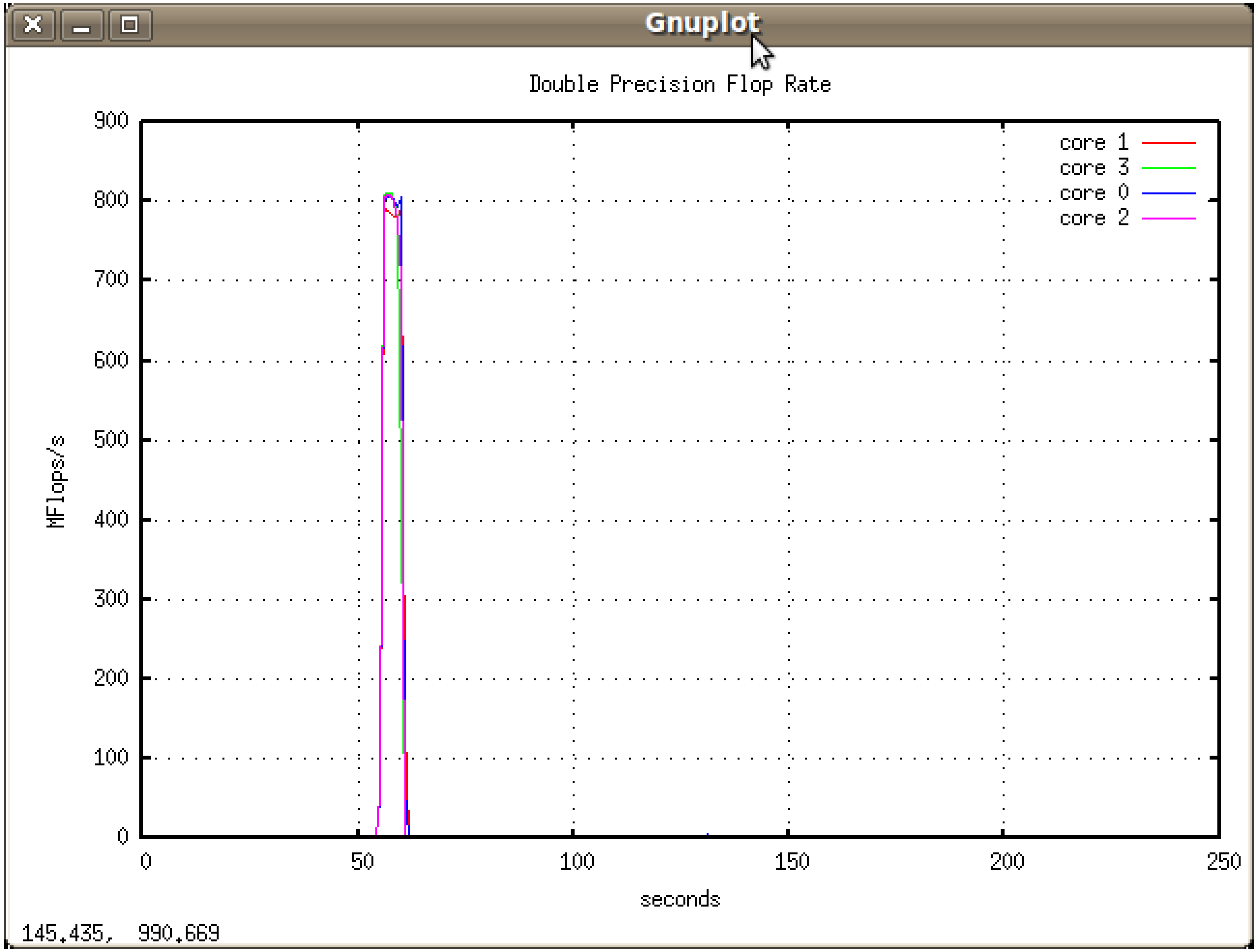




likwid-perfscope

Experimental frontend to Timeline mode

```
likwid-perfscope  -group FLOPS_DP -cores 0-3
```



likwid-perfctr

Marker API

- To measure only parts of an application a marker API is available.
- The API only turns counters on/off. The configuration of the counters is still done by likwid-perfctr application.
- Multiple named regions can be measured
- Results on multiple calls are accumulated
- Inclusive and overlapping Regions are allowed

```
likwid_markerInit();  // must be called from serial region

likwid_markerStartRegion(“Compute”);
. . .
likwid_markerStopRegion(“Compute”);


likwid_markerStartRegion(“postprocess”);
. . .
likwid_markerStopRegion(“postprocess”);


likwid_markerClose();  // must be called from serial region
```

likwid-perfctr
Group files

```
SHORT PSTI
EVENTSET
FIXC0 INSTR_RETIRED_ANY
FIXC1 CPU_CLK_UNHALTED_CORE
FIXC2 CPU_CLK_UNHALTED_REF
PMC0  FP_COMP_OPS_EXE_SSE_FP_PACKED
PMC1  FP_COMP_OPS_EXE_SSE_FP_SCALAR
PMC2  FP_COMP_OPS_EXE_SSE_SINGLE_PRECISION
PMC3  FP_COMP_OPS_EXE_SSE_DOUBLE_PRECISION
UPMC0  UNC_QMC_NORMAL_READS_ANY
UPMC1  UNC_QMC_WRITES_FULL_ANY
UPMC2 UNC_QHL_REQUESTS_REMOTE_READS
UPMC3 UNC_QHL_REQUESTS_LOCAL_READS
METRICS
Runtime [s] FIXC1*inverseClock
CPI  FIXC1/FIXC0
Clock [MHz]  1.E-06*(FIXC1/FIXC2)/inverseClock
DP MFlops/s (DP assumed) 1.0E-06*(PMC0*2.0+PMC1)/time
Packed MUOPS/s   1.0E-06*PMC0/time
Scalar MUOPS/s 1.0E-06*PMC1/time
SP MUOPS/s 1.0E-06*PMC2/time
DP MUOPS/s 1.0E-06*PMC3/time
Memory bandwidth [MBytes/s] 1.0E-06*(UPMC0+UPMC1)*64/time;
Remote Read BW [MBytes/s] 1.0E-06*(UPMC2)*64/time;
LONG
Formula:
DP MFlops/s =  (FP_COMP_OPS_EXE_SSE_FP_PACKED*2 +  FP_COMP_OPS_EXE_SSE_FP_SCALAR)/ runtime.
```

- Groups are architecture specific
- They are defined in simple text files
- During recompile the code is generated
- likwid-perfctr -a outputs list of groups
- For every group an extensive documentation is available

likwid-perfctr

Output filters

Likwid supports to specify an output file with placeholder for:

- %j - PBS_JOBID taken from environment
- %r - MPI Rank as specified by newer Intel MPI versions
- %h - hostname
- %p - process pid

Example:

```
likwid-perfctr -c L:0 -g FLOPS_DP -o test_%h_%p.txt  ./a.out
```

Depending on the file suffix a converter script is called:

- txt Direct output without conversion
- csv Convert to comma separated values format
- xml Convert to xml format

Useful for integration in other tool chains or automated frameworks.

likwid-perfctr

More information

- **Implemented completely in user space (uses msr kernel module)**
- **For security sensitive environments a small proxy application managing a controlled access to the msr device files is available**
- **Supported processors:**
  - Intel Core 2
  - Intel Nehalem /Westmere (all variants) supporting Uncore events
  - Intel NehalemEX/WestmereEX (without Uncore)
  - Intel Sandy Bridge
  - Intel Atom
  - AMD K8/K10
- **likwid-perfctr allows to specify arbitrary event sets on the command line:**

```
likwid-perfctr -c 0-12 -g
  INSTR_RETIRED_ANY:FIXC0,CPU_CLK_UNHALTED_CORE:FIXC1,FP_COMP
  _OPS_EXE_SSE_FP_PACKED:PMC0,UNC_L3_LINES_IN_ANY:UPMC0
  sleep 10
```

likwid-perfctr

Usage with MPI

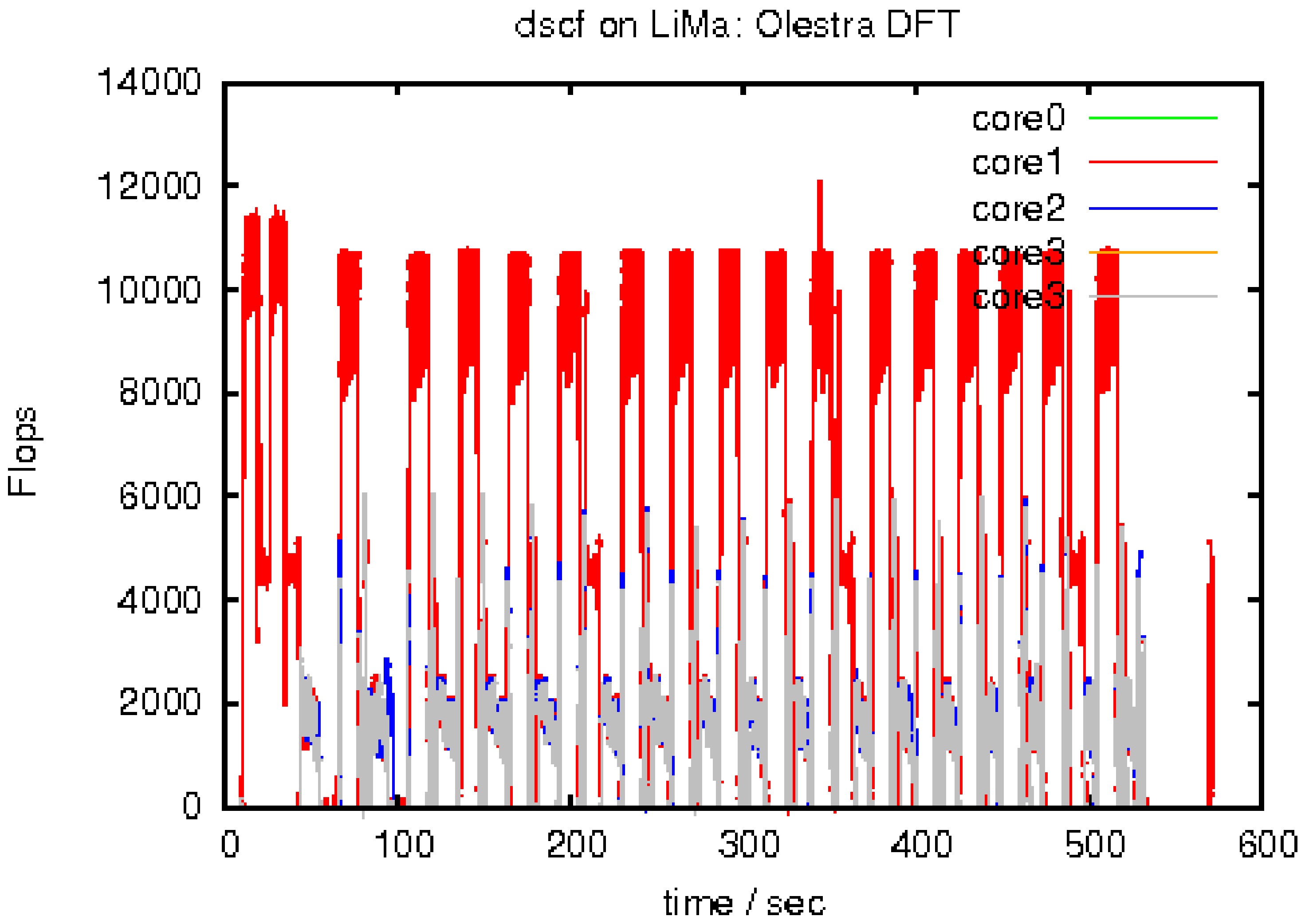
- likwid-perfctr can be used with MPI if processes are pinned
- For hybrid usage a `taskset` cpuset must be established
- To distinguish the output it can be written to separate files

```
likwid-perfctr -c L:0 -g FLOPS_DP -o myTag_%r_%h.txt ./app
```

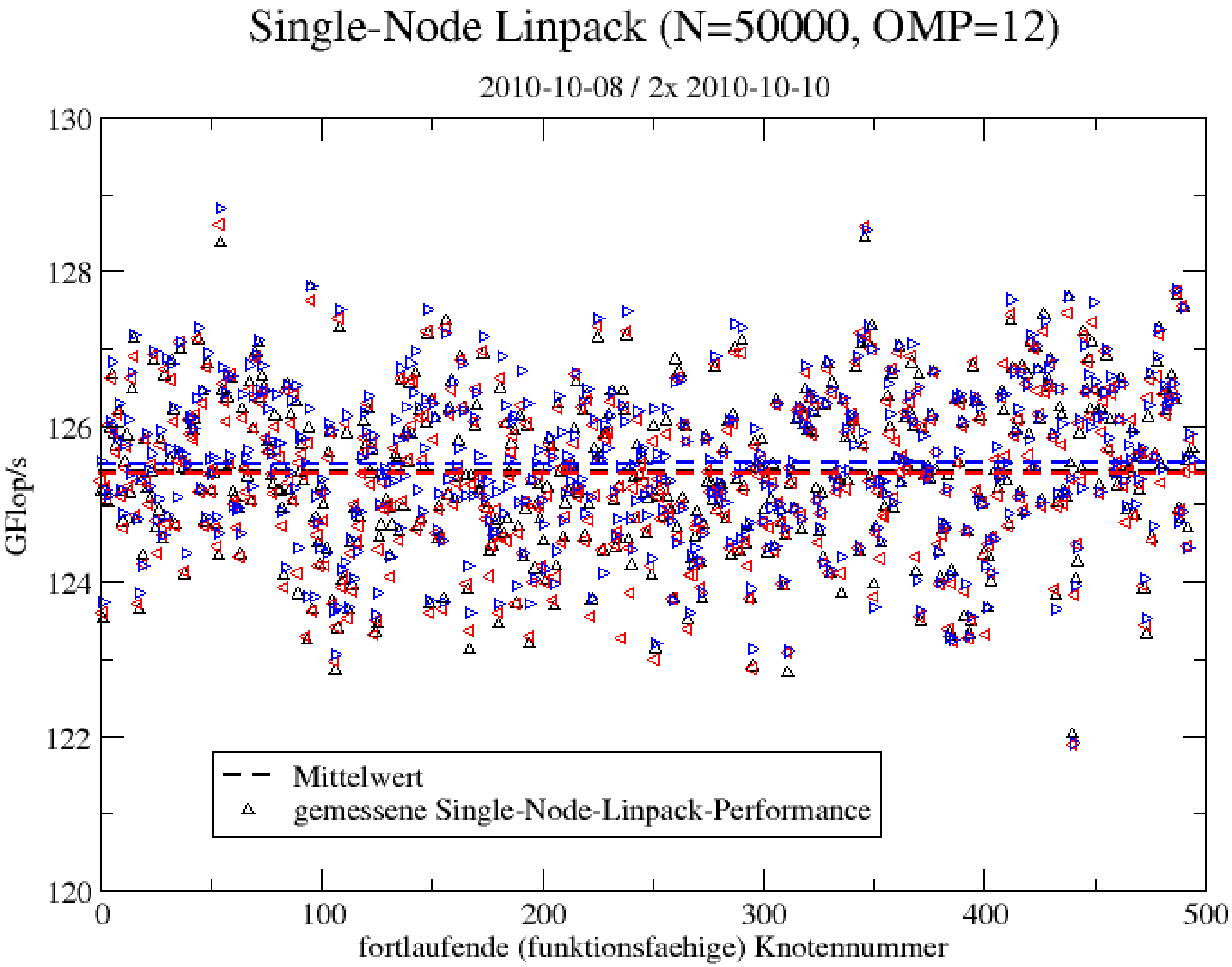
- There are efforts to add likwid support in Scalaska (and Vampir ?)
- likwid-mpirun will have support for perfctr in the future
- Well suited to be integrated into batch job system

likwid-perfctr

Example turbomole (MPI)

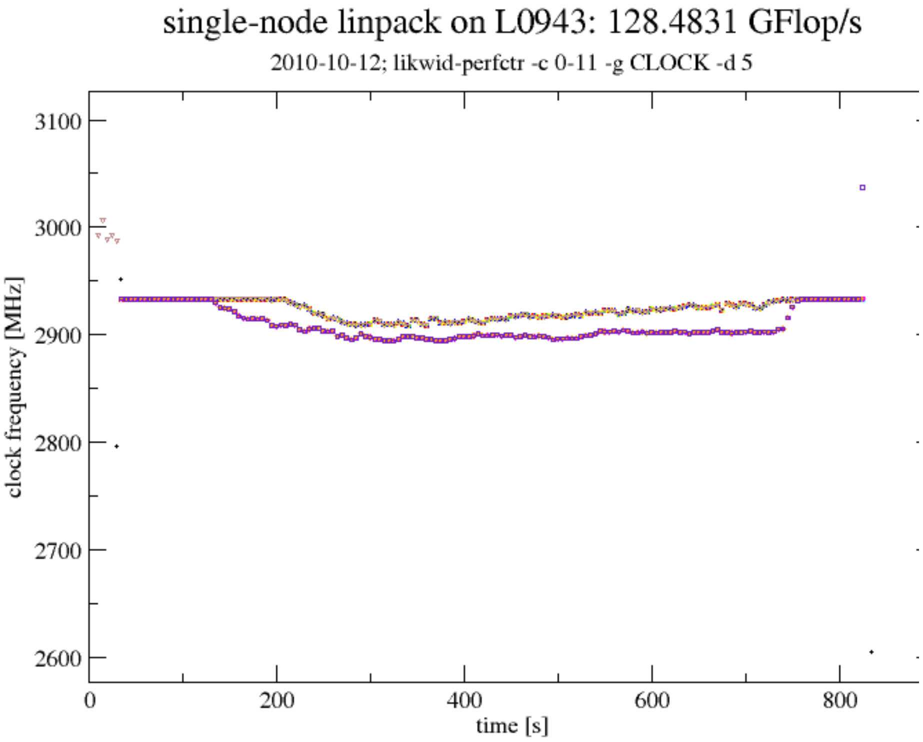


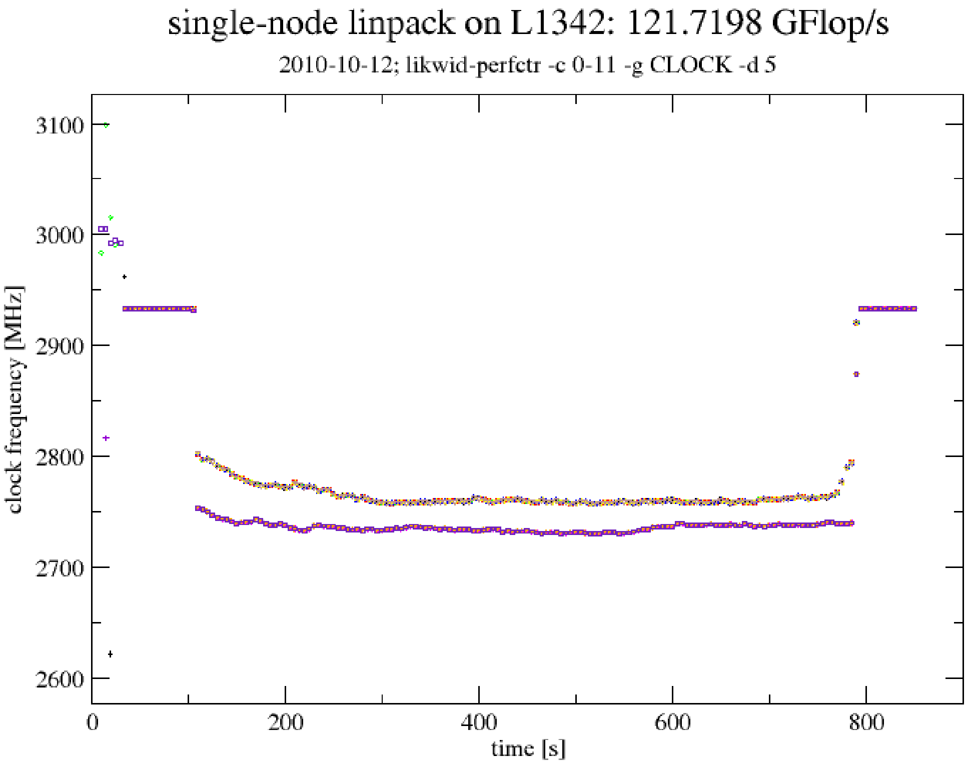
The dangers of Overclocking …



Turbo mode on Westmere

Turbo mode causes volatile performance values





likwid-perfctr
Upcoming features

- Full Uncore support for Intel EX type processors

```
WBOX4 UNCORE_CYCLES
MBOXA0 FVC_EV0_BBOX_CMDS_READS
MBOXB0 FVC_EV0_BBOX_CMDS_READS
BBOXA1 IMT_INSERTS_WR
BBOXB1 IMT_INSERTS_WR
RBOXA0 NEW_PACKETS_RECV_PORT0_IPERF0_ANY_DRS
RBOXA1 NEW_PACKETS_RECV_PORT1_IPERF0_ANY_DRS
RBOXB0 NEW_PACKETS_RECV_PORT4_IPERF0_ANY_DRS
RBOXB1 NEW_PACKETS_RECV_PORT5_IPERF0_ANY_DRS
```

- Full support for AMD Interlagos
- Support for IBM Power 7
- Multiplexing
- Extension of likwid-perfscope
- Porting to Windows and MacOSX

likwid-bench

Microbenchmarking application/platform

- To know the performance properties of a machine is essential for any optimization effort
- Microbenchmarking is an important tool to gain this information

- Extensible, flexible benchmarking framework
- Rapid development of low level kernels
- Already includes many ready to use threaded benchmark kernels
- Benchmarking runtime cares for:
  - Thread management and placement
  - Data allocation and NUMA aware initialization
  - Timing and result presentation

likwid-bench Example

- Implement micro benchmark in abstract assembly
- Add meta information
- The benchmark file is automatically converted, compiled and added to the benchmark application

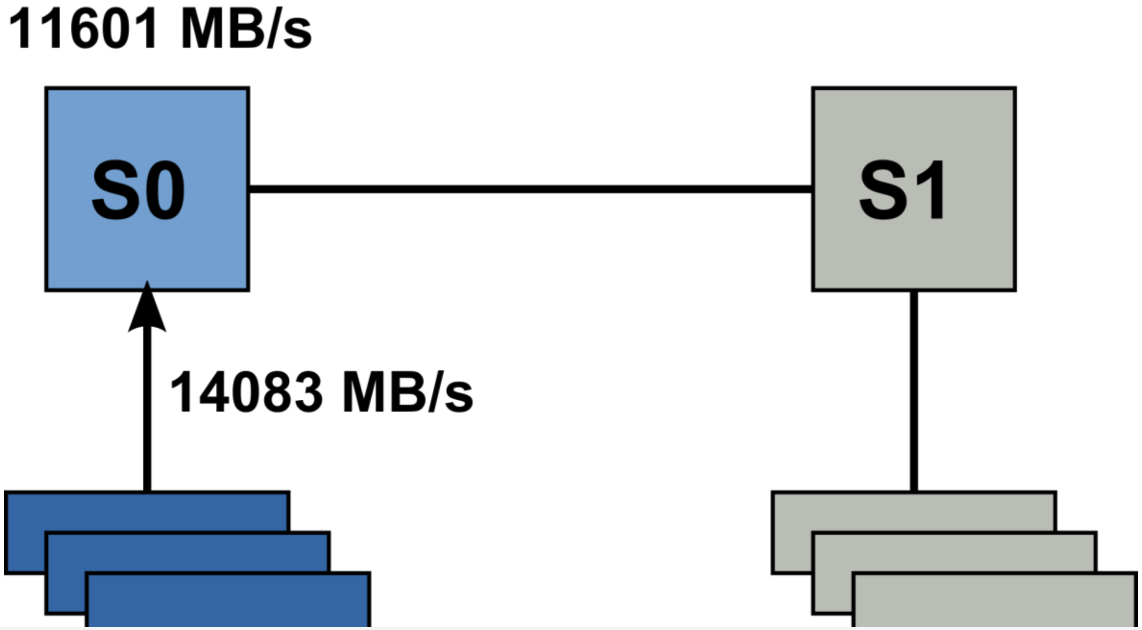
```
$likwid-bench -t clcopy -g 1 -i 1000 -w S0:1MB:2

$likwid-bench -t load -g 2 -i 100 -w S1:1GB  -w S0:1GB-0:S1,1:S0
```

```
STREAMS 2
TYPE DOUBLE
FLOPS 0
BYTES 16
LOOP  32
movaps      FPR1, [STR0 + GPR1 * 8 ]
movaps      FPR2, [STR0 + GPR1 * 8 + 64 ]
movaps      FPR3, [STR0 + GPR1 * 8 + 128 ]
movaps      FPR4, [STR0 + GPR1 * 8 + 192 ]
movaps      [STR1 + GPR1 * 8 ], FPR1
movaps      [STR1 + GPR1 * 8 + 64 ], FPR2
movaps      [STR1 + GPR1 * 8 + 128 ], FPR3
movaps      [STR1 + GPR1 * 8 + 192 ], FPR4
```

Detecting NUMA problems

- Testcase memcpy (without NT stores)
- tested with likwid-bench on dual socket Nehalem node
- Total bandwidth as reported by likwid-bench
- Bandwidths measured from outside with

```
likwid-perfctr -c S0:0@S1:0 -g MEM
```



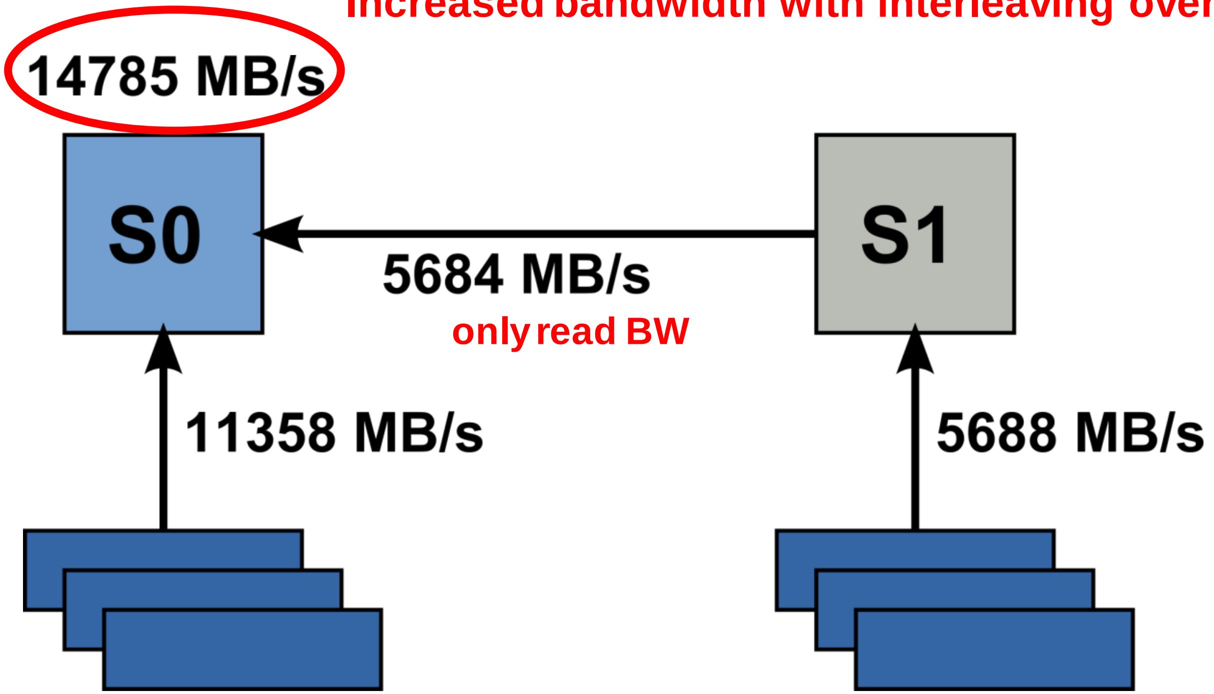
```
likwid-bench -g 1 -i 1000 -t copy -w S0:500MB:4
```

Detecting NUMA problems 2

- Thread group (4 threads) on socket 0
- Store stream to socket 0 memory
- Load stream from socket 1 memory

```
likwid-bench -g 1 -i 1000 -t copy -w S0:500MB:4-0:S1,1:S0
```

Increased bandwidth with interleaving over QPI



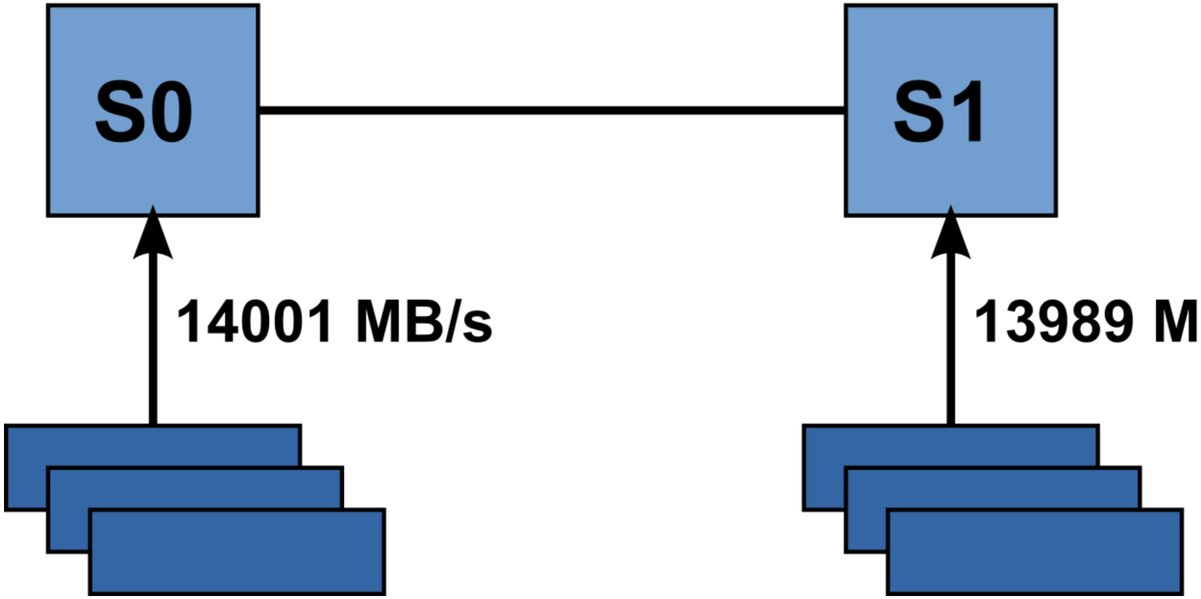
only read BW

Detecting NUMA problems 3

- 2 thread group (2x4 threads) on socket 0/1
- Correct NUMA placement (first touch)

```
likwid-bench -g 2 -i 1000 -t copy -w S0:500MB:4
 -w S1:500MB:4
```

23147 MB/s

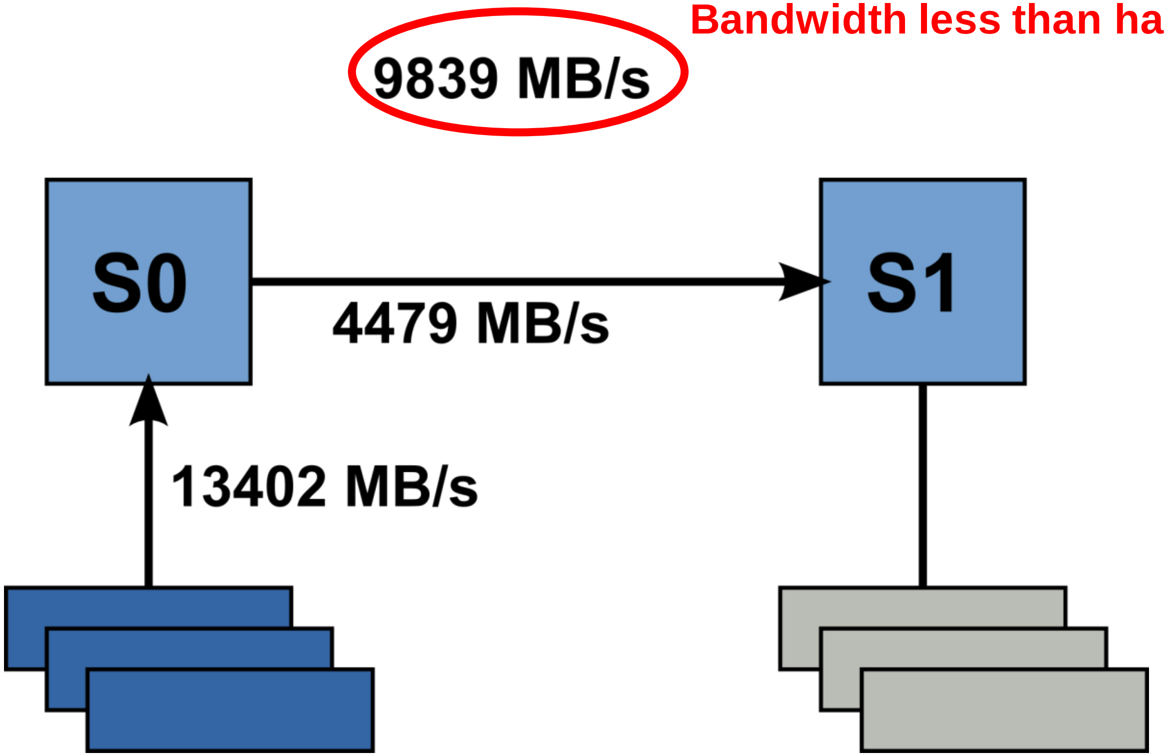


Detecting NUMA problems 4

- 2 thread group (2x4 threads) on socket 0/1
- Common problem: memory placed on one socket only

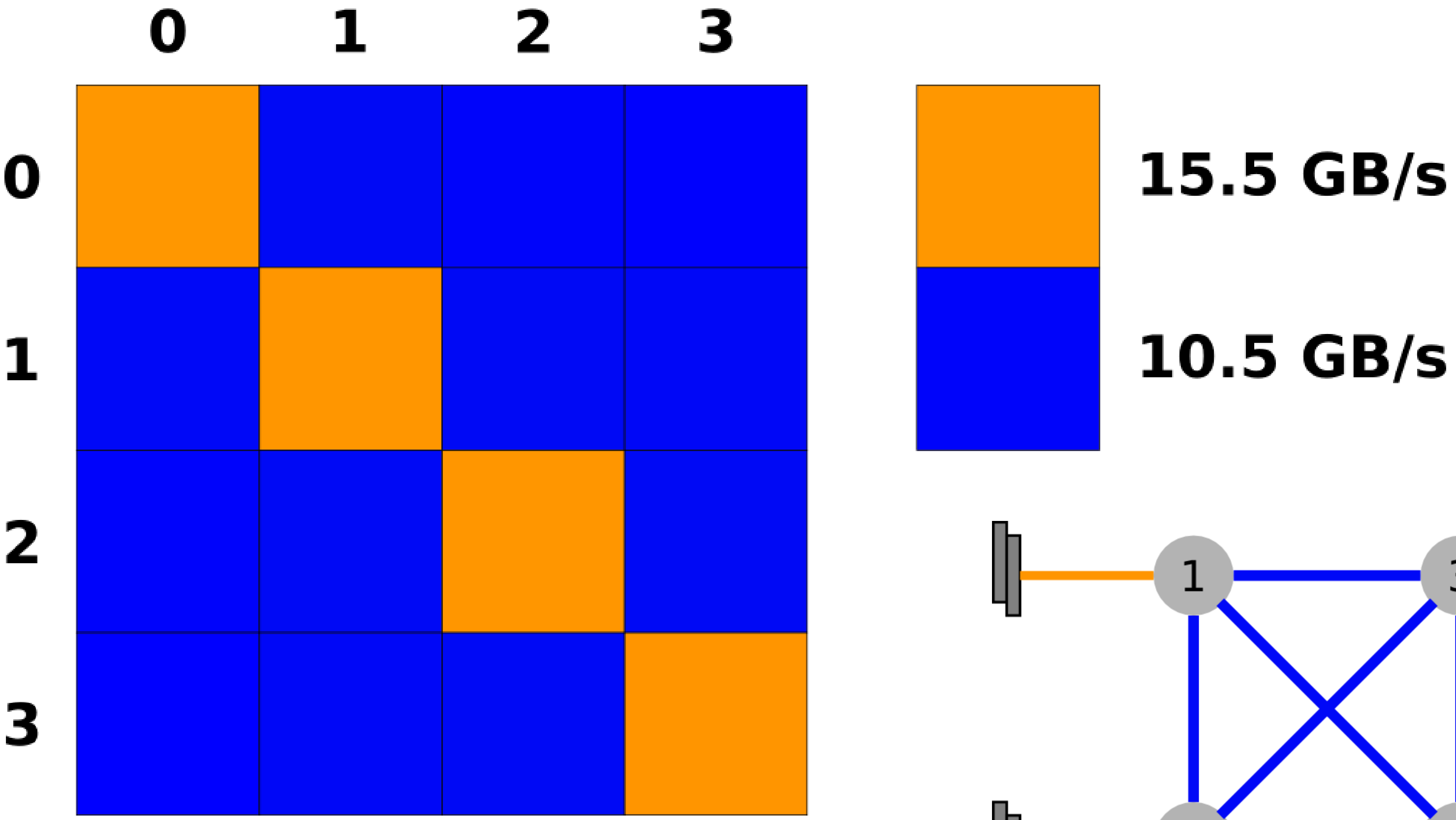
```
likwid-bench -g 2 -i 1000 -t copy -w S0:500MB:4
 -w S1:500MB:4-0:S0,1:S0
```

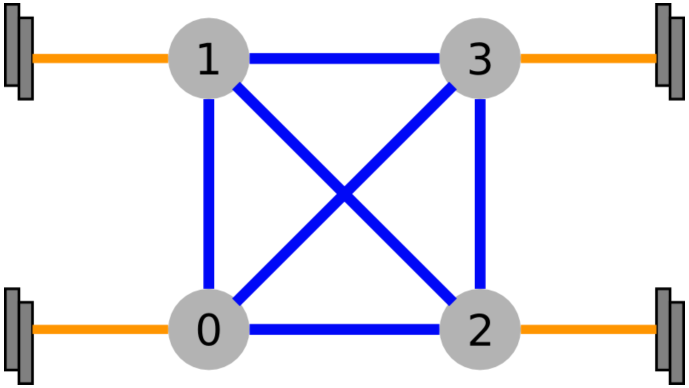
Bandwidth less than half



Intel Nehalem EX 4-socket system

ccNUMA bandwidth map

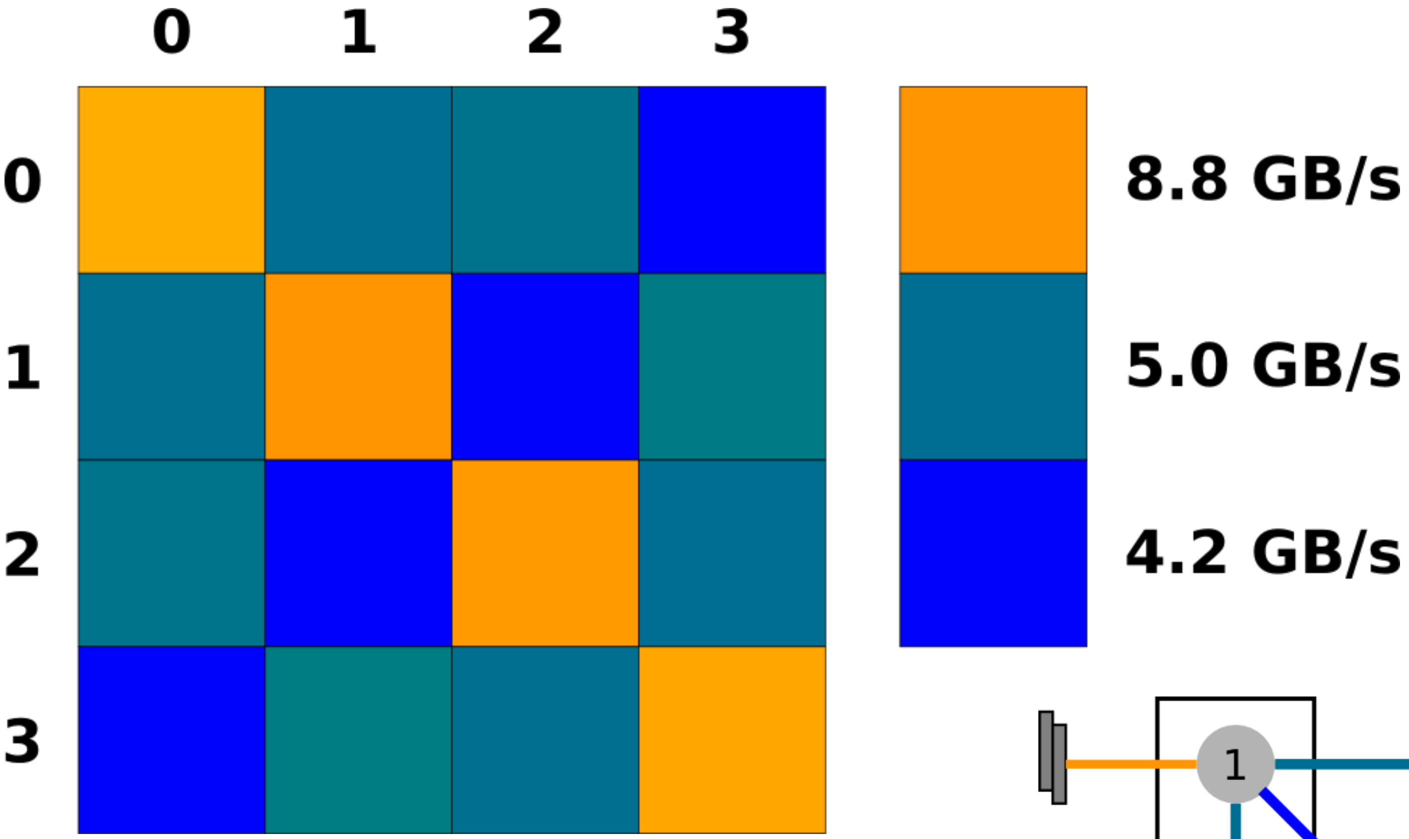


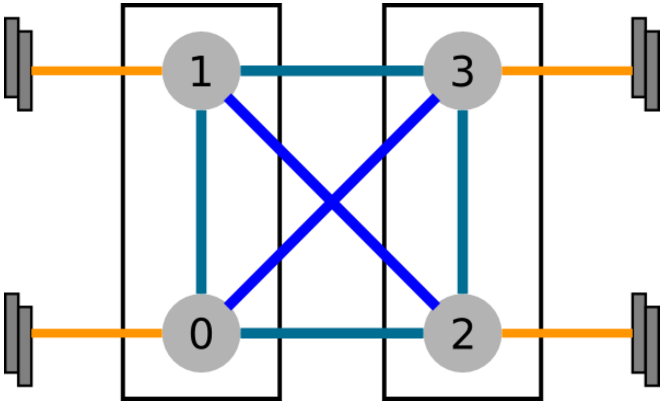


Bandwidth map created with likwid-bench. All cores used in one NUMA domain, memory is placed in a different NUMA domain. Test case: simple copy `A(:)=B(:)`, large arrays

AMD Magny Cours 2-socket system

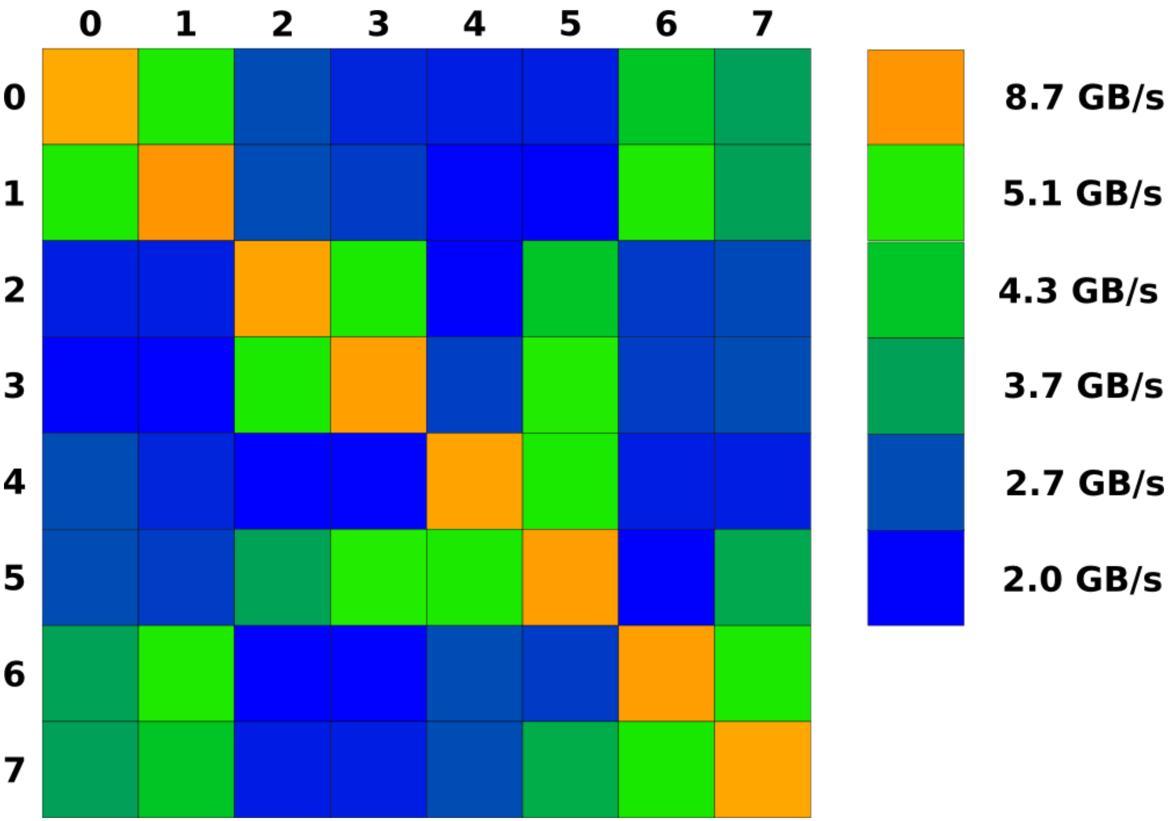
4 chips, two sockets

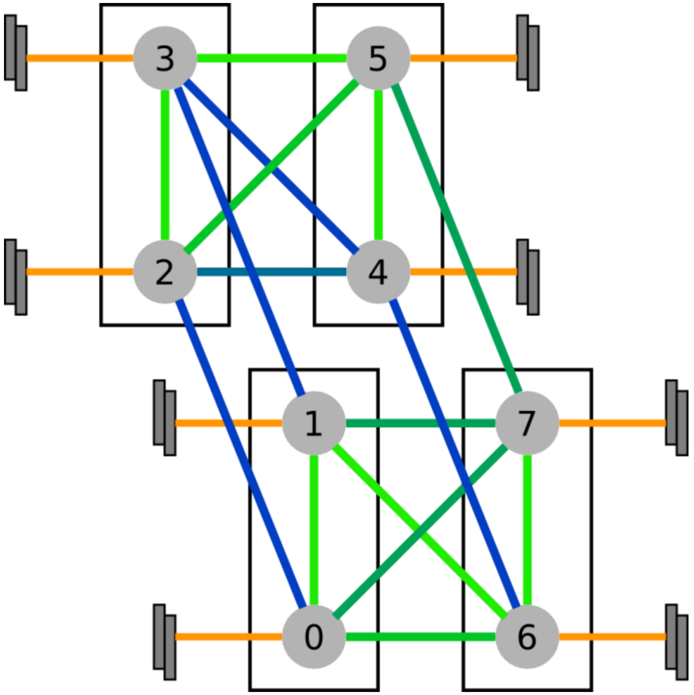




AMD Magny Cours 4-socket system

Topology at its best?





Scenario 2: Dealing with node topology and thread affinity

likwid-topology

likwid-pin

likwid-mpirun

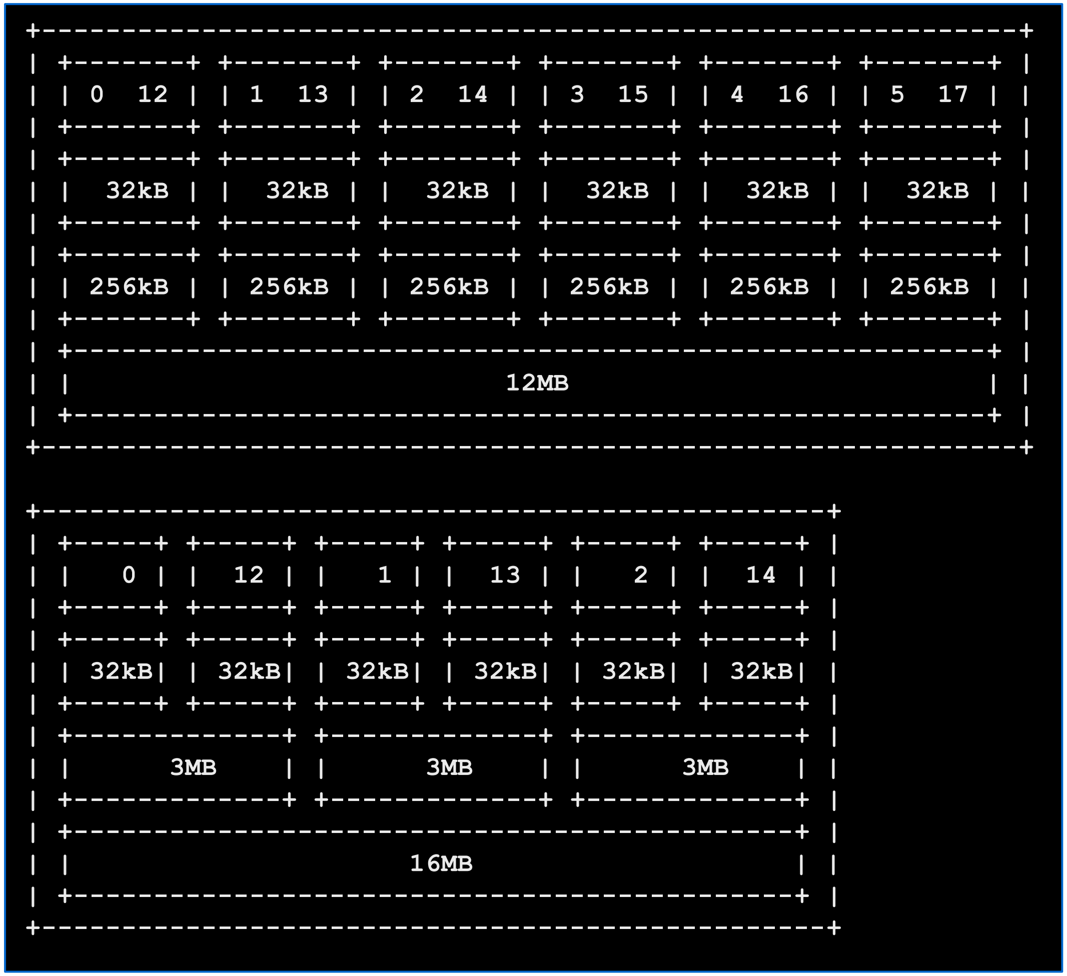
likwid-topology

Single source of node information

- **Node information is usually scattered in various places**
- **likwid-topology provides all information in a single reliable source**
- **All information is based on cpuid directly**
- **Features:**
  - Thread topology
  - Cache topology
  - NUMA topology
  - Detailed cache parameters (-c command line switch)
  - Processor clock (measured)
  - ASCII art output (-g command line switch)

Usage: likwid-topology

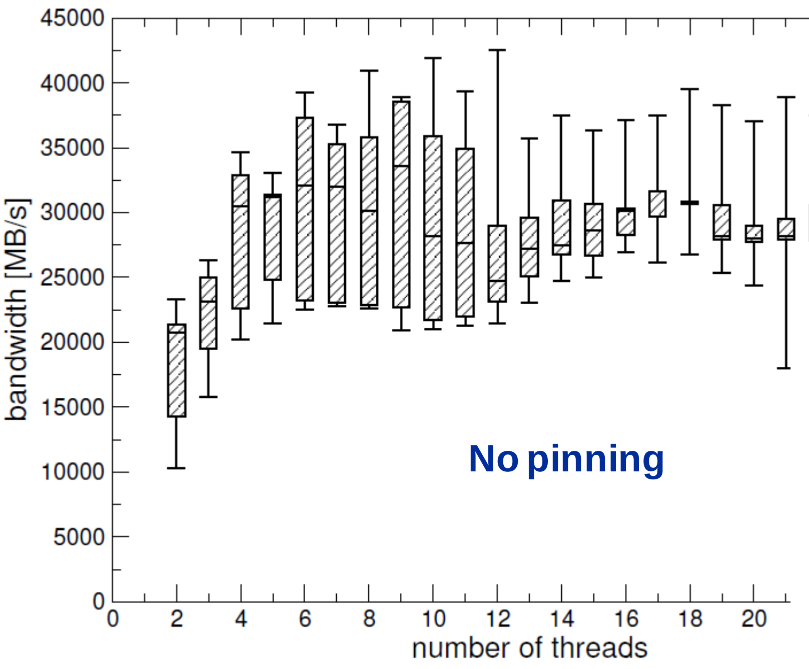
```
-------------------------------------------------------------
CPU type:      Intel Core Westmere processor
*************************************************************
Sockets:                       2
Cores per socket:              6
Threads per core:              2
-------------------------------------------------------------
HWThread       Thread         Core           Socket
0              0              0              0
1              0              1              0
2              0              2              0
-------------------------------------------------------------
Socket 0:  ( 0 12 1 13 2 14 3 15 4 16 5 17 )
Socket 1:  ( 6 18 7 19 8 20 9 21 10 22 11 23 )
-------------------------------------------------------------
Cache Topology
-------------------------------------------------------------
Level:          3
Size:           12 MB
Type:           Unified cache
Associativity:  16
Number of sets: 12288
Cache line size: 64
Non Inclusive cache
Shared among 12 threads
Cache groups:  ( 0 12 1 13 2 14 3 15 4 16 5 17 ) ( 6 18 7 19 8
-------------------------------------------------------------
NUMA Topology
NUMA domains: 2
-------------------------------------------------------------
Domain 0:
 Processors:  0 1 2 3 4 5 12 13 14 15 16 17
Memory: 11615.9 MB free of total 12276.3 MB
-------------------------------------------------------------
Domain 1:
 Processors:  6 7 8 9 10 11 18 19 20 21 22 23
Memory: 12013.9 MB free of total 12288 MB
-------------------------------------------------------------
```



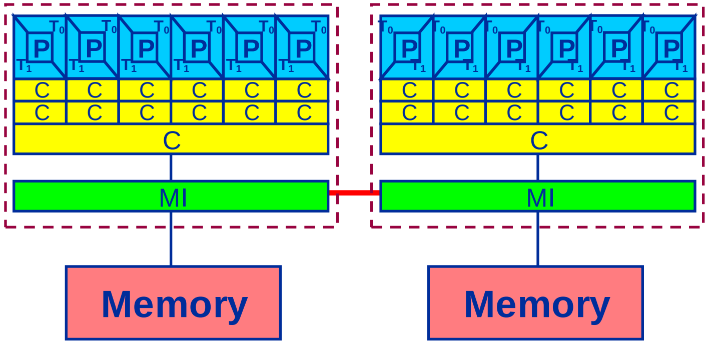
```
+-------------------------------------------------------------------+
| +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ |
| | 0   12 | | 1   13 | | 2   14 | | 3   15 | | 4   16 | | 5   17 | |
| +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ |
| +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ |
| |  32kB  | |  32kB  | |  32kB  | |  32kB  | |  32kB  | |  32kB  | |
| +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ |
| +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ |
| | 256kB  | | 256kB  | | 256kB  | | 256kB  | | 256kB  | | 256kB  | |
| +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ |
| +---------------------------------------------------------------+ |
| |                             12MB                              | |
| +---------------------------------------------------------------+ |
+-------------------------------------------------------------------+

+-------------------------------------------------+
| +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ |
| |   0 | |  12 | |   1 | |  13 | |   2 | |  14 | |
| +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ |
| +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ |
| | 32kB| | 32kB| | 32kB| | 32kB| | 32kB| | 32kB| |
| +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ |
| +-------------+ +-------------+ +-------------+ |
| |     3MB     | |     3MB     | |     3MB     | |
| +-------------+ +-------------+ +-------------+ |
| +---------------------------------------------+ |
| |                    16MB                     | |
| +---------------------------------------------+ |
+-------------------------------------------------+
```

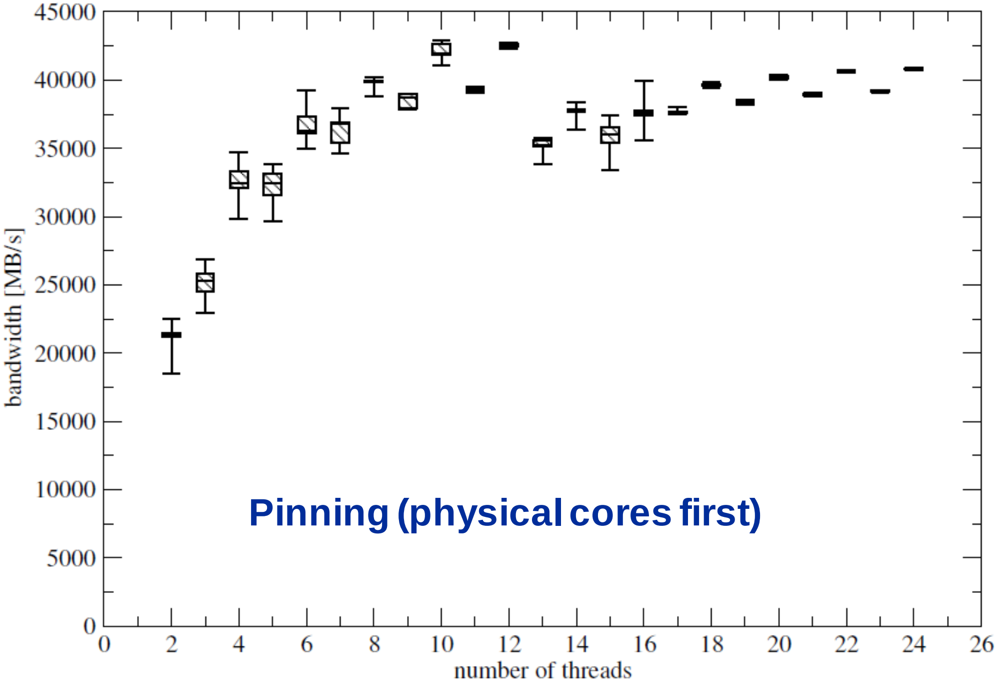
Information can also be queried in library.
NUMA information extracted from Linux sys fs.

Example: STREAM benchmark on 12-core Intel Westmere:
Anarchy vs. thread pinning



No pinning





Pinning (physical cores first)

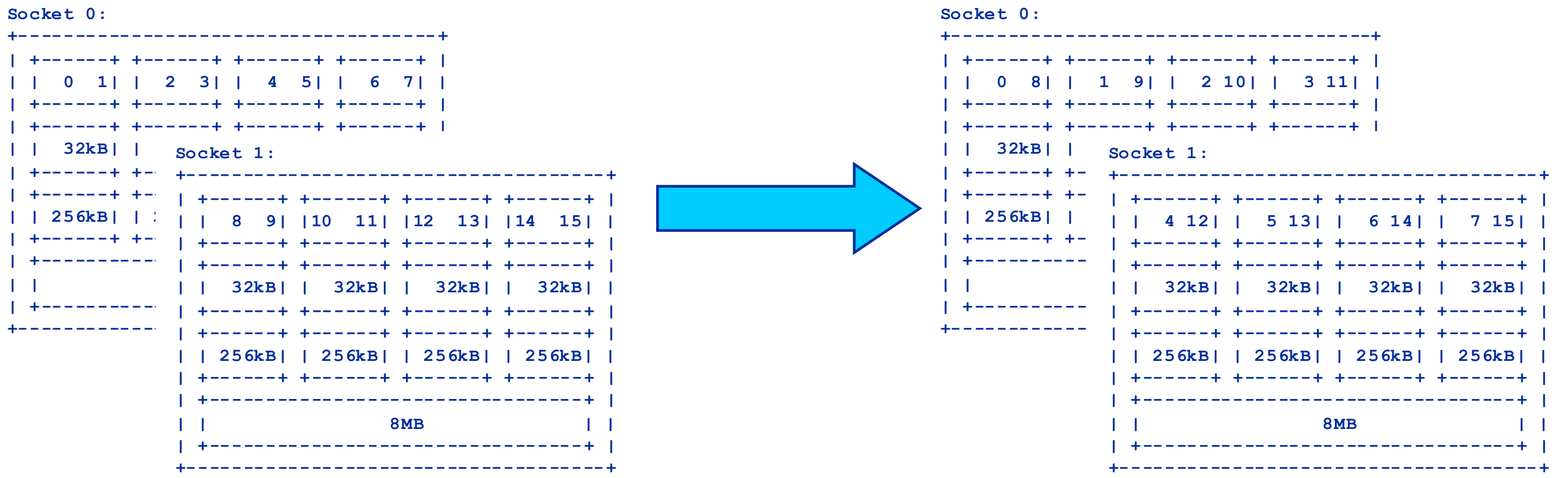
There are several reasons for caring about affinity:

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

Likwid-pin

Using logical core numbering

- **Core numbering may vary from system to system even with identical hardware**
  - likwid-topology delivers this information, which can then be fed into likwid-pin
- **Alternatively, likwid-pin can abstract this variation and provide a purely logical numbering (physical cores first)**



  - Across all cores in the node:

    `OMP_NUM_THREADS=8 likwid-pin -c N:0-7 ./a.out`
  - Across the cores in each socket and across sockets in each node:

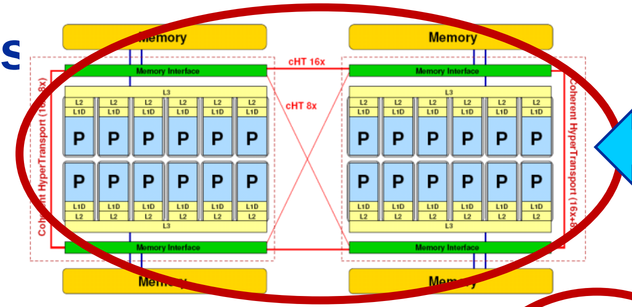
    `OMP_NUM_THREADS=8 likwid-pin -c S0:0-3@S1:0-3 ./a.out`

Likwid-pin

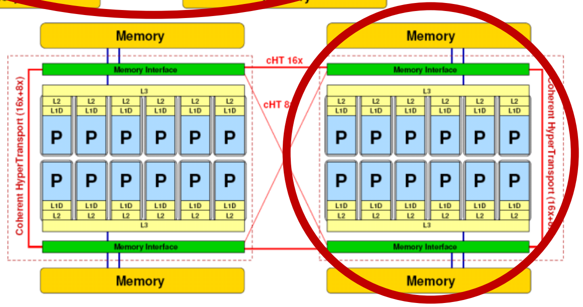
Using logical core numbering

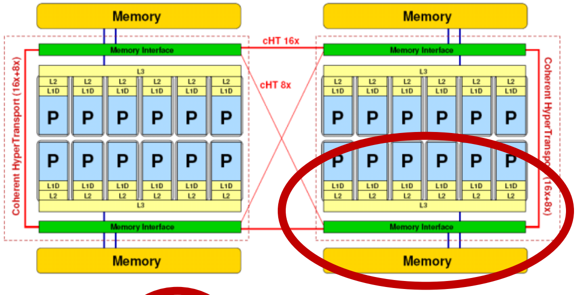
- **Possible unit prefixes**

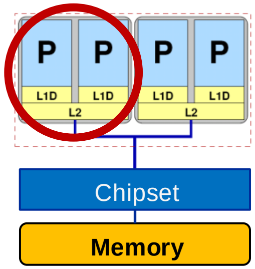
N	**node**
S	**socket**
M	**NUMA domain**
C	**outer level cache group**



Default if –c is not specified!



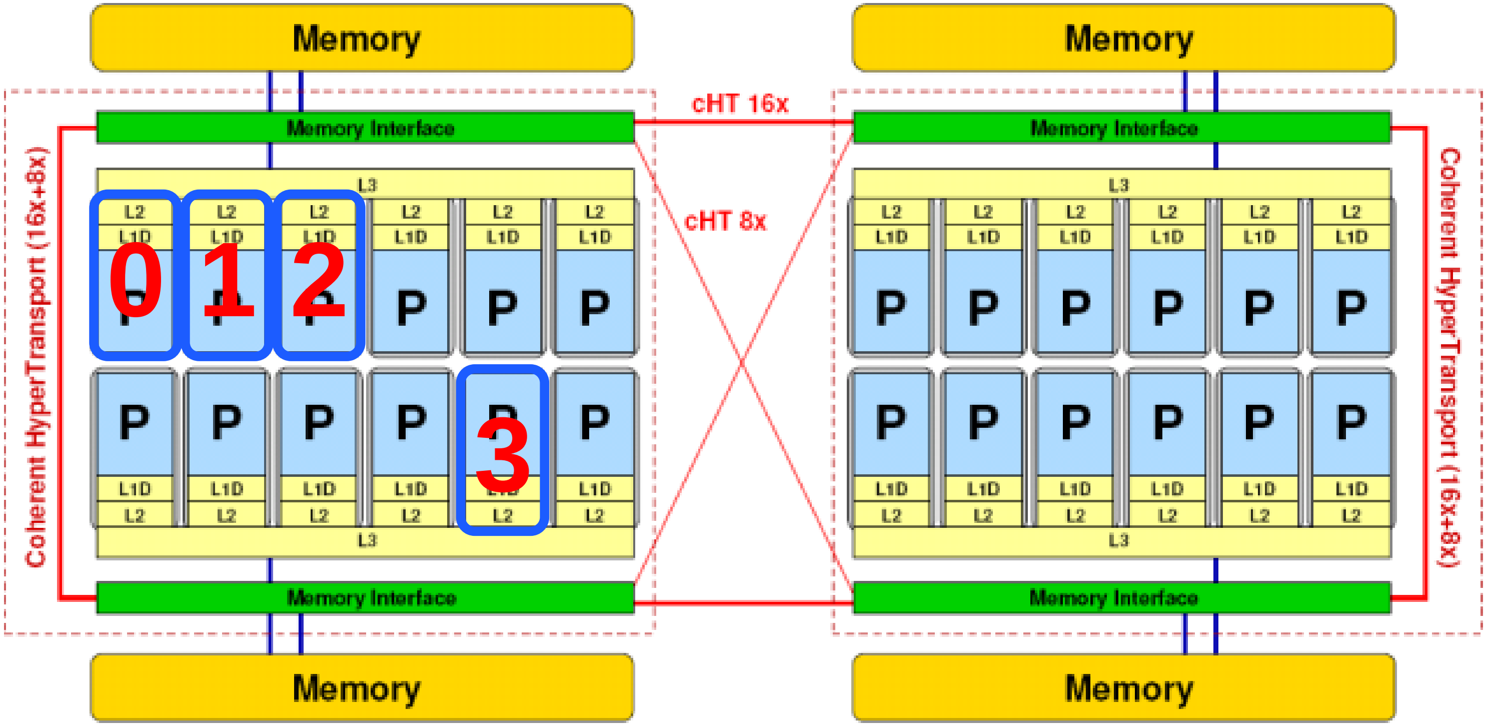




Likwid-pin

Using logical core numbering

- … and: Logical numbering inside a pre-existing cpuset:



- `OMP_NUM_THREADS=4 likwid-pin -c L:0-3 ./a.out`

likwid-pin

Further details

- Pins process and threads to specific cores without touching code
- Directly supports pthreads, gcc OpenMP, Intel OpenMP
- Allows user to specify skip mask (hybrid OpenMP/MPI)
- Can also be used as replacement for taskset

Supported usage modes:

- Physical numbering: `likwid-pin -c 0,2,5-8`
- Logical numbering (node): `likwid-pin -c N:3-7`
- Logical numbering (socket): `likwid-pin -c S0:0,2@S2:0-3`
- Logical numbering (NUMA): `likwid-pin -c M0:1-3@M2:1-3`
- Logical numbering (cpuset): `likwid-pin -c L:3-7`

All logical numberings have physical cores first.

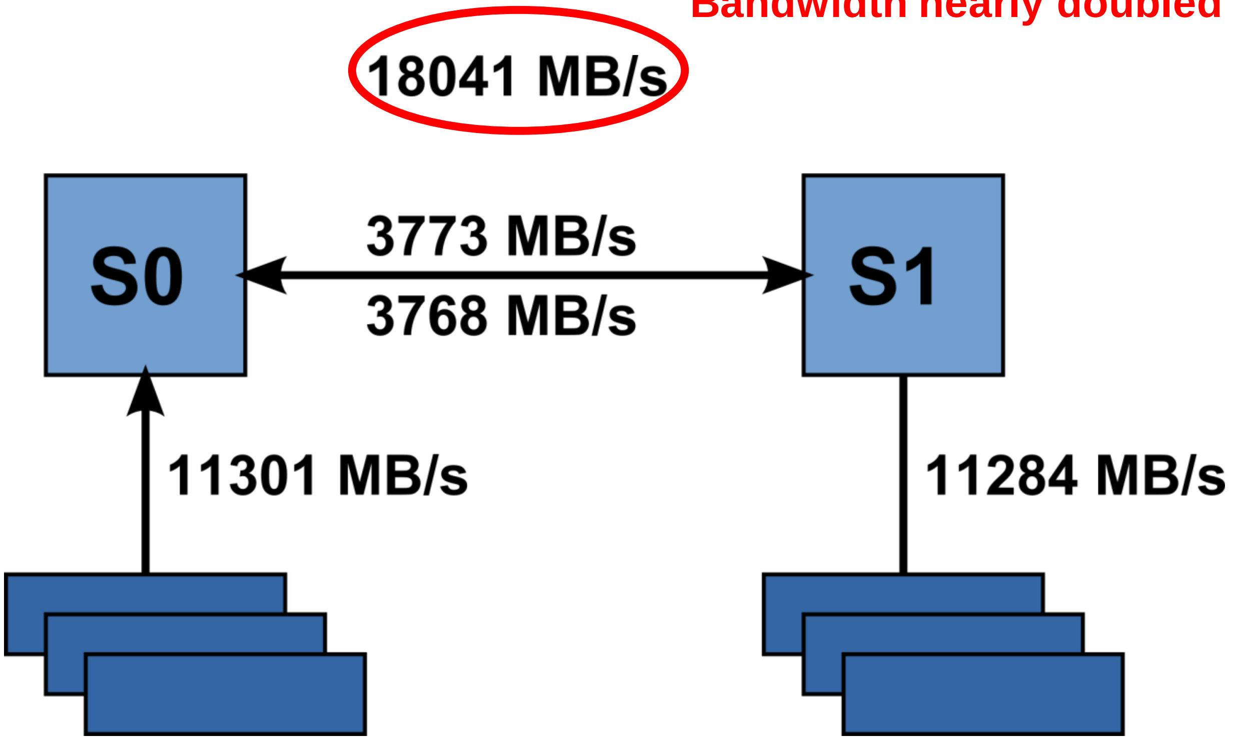
likwid-pin

Interleaving of memory pages

- **Effective improvement without any code change possible**
- **Memory policy is set to interleave with likwid-pin:**

```
likwid-pin -c N:0-7 -i likwid-bench -g 2 -i 1000 -t
copy -w S0:500MB:4 -w S1:500MB:4-0:S0,1:S0
```

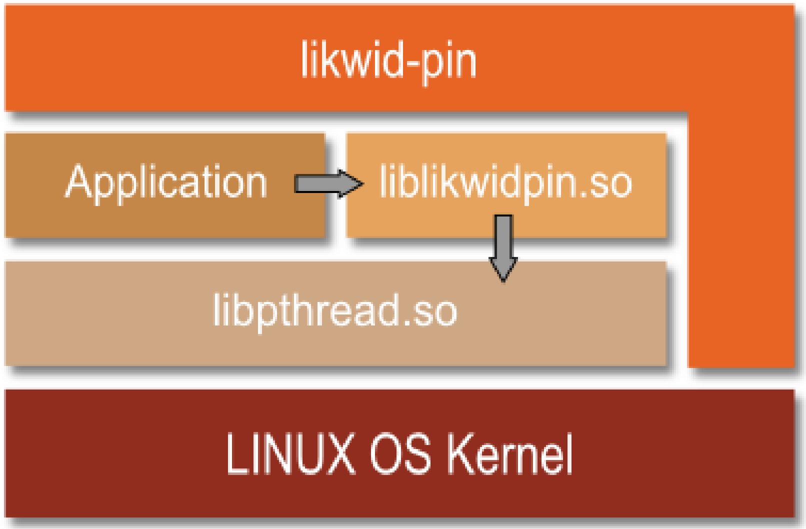
Bandwidth nearly doubled



likwid-pin

How it works

- Uses LD_PRELOAD for `pthread_create`
- Wrapper application controls library through environment variables
- Upon creation threads corresponding to bit position in a skip mask are not pinned



likwid-mpirun

MPI and Hybrid pinning

- On the long run a unified standard is needed
- Till then likwid provides a portable solution
- The examples here are for Intel MPI/OpenMP programs, but are also applicable to other threading models

At the moment the following issues arise

- Does the OpenMP implementation use a shepherd thread?
- Does the MPI implementation use a shepherd thread?
- Which threadIDs need to be skipped?

Pure MPI:

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

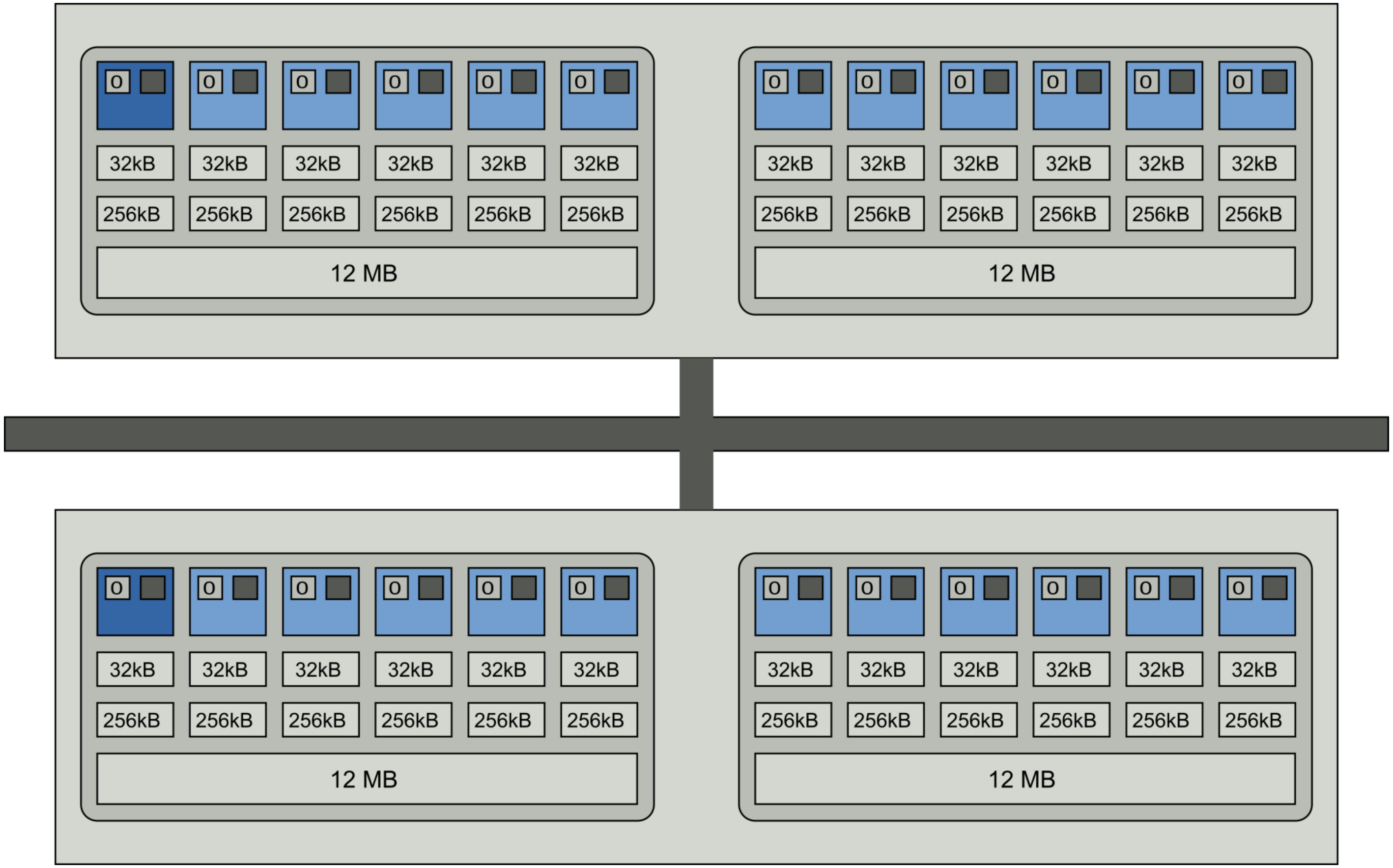
Hybrid:

```
likwid-mpirun -np 16 -pin S0:0,1_S1:0,1 ./a.out
```

likwid-mpirun

1 MPI process per node

```
OMP_NUM_THREADS=12 likwid-mpirun -np 2 -pin N:0-11  ./a.out
```



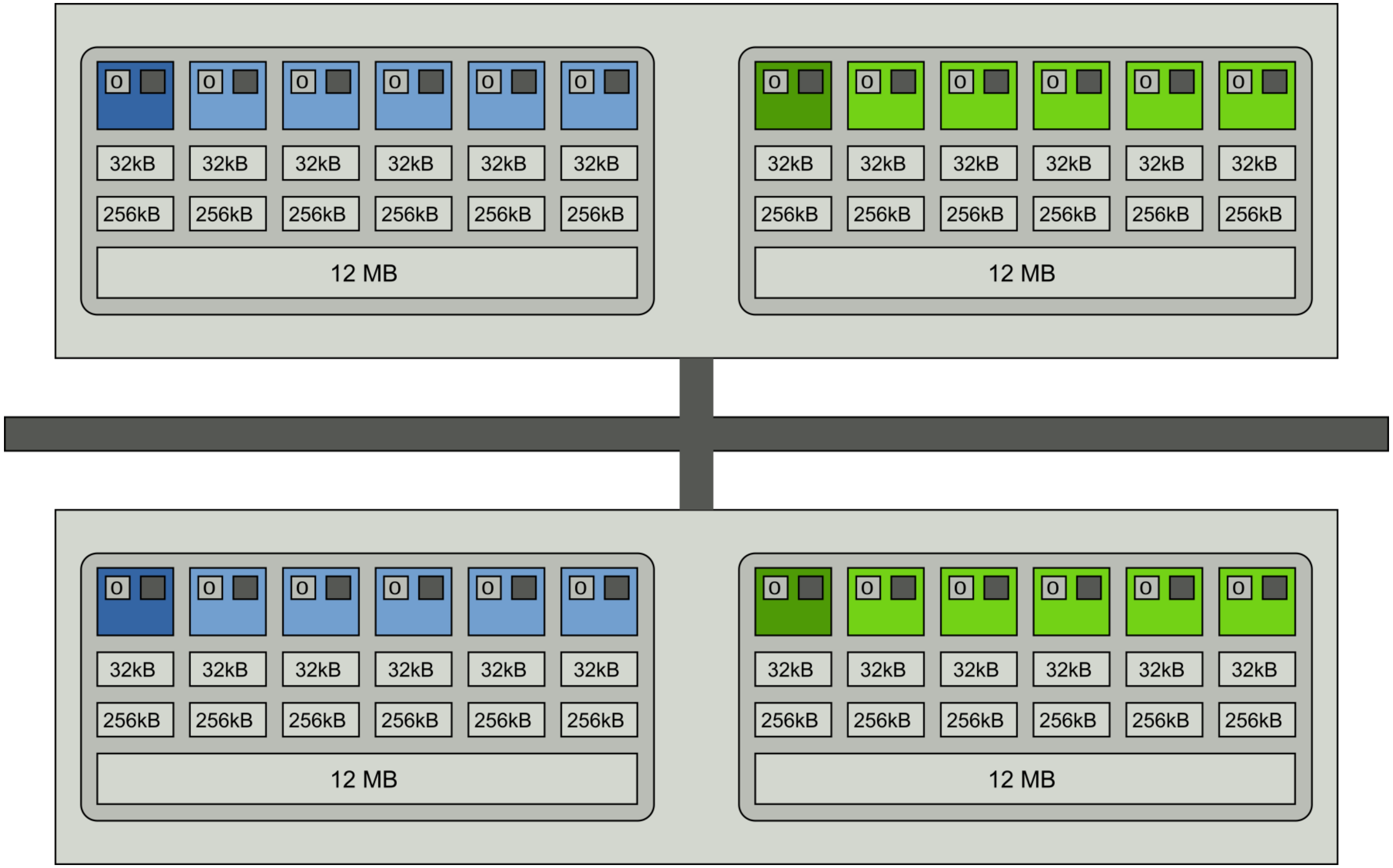
Intel MPI+compiler:

```
OMP_NUM_THREADS=12 mpirun -ppn 1 -np 2 -env KMP_AFFINITY scatter ./a.out
```

likwid-mpirun

1 MPI process per socket

```
OMP_NUM_THREADS=6  likwid-mpirun -np 4 -pin S0:0-5_S1:0-5 ./a.out
```



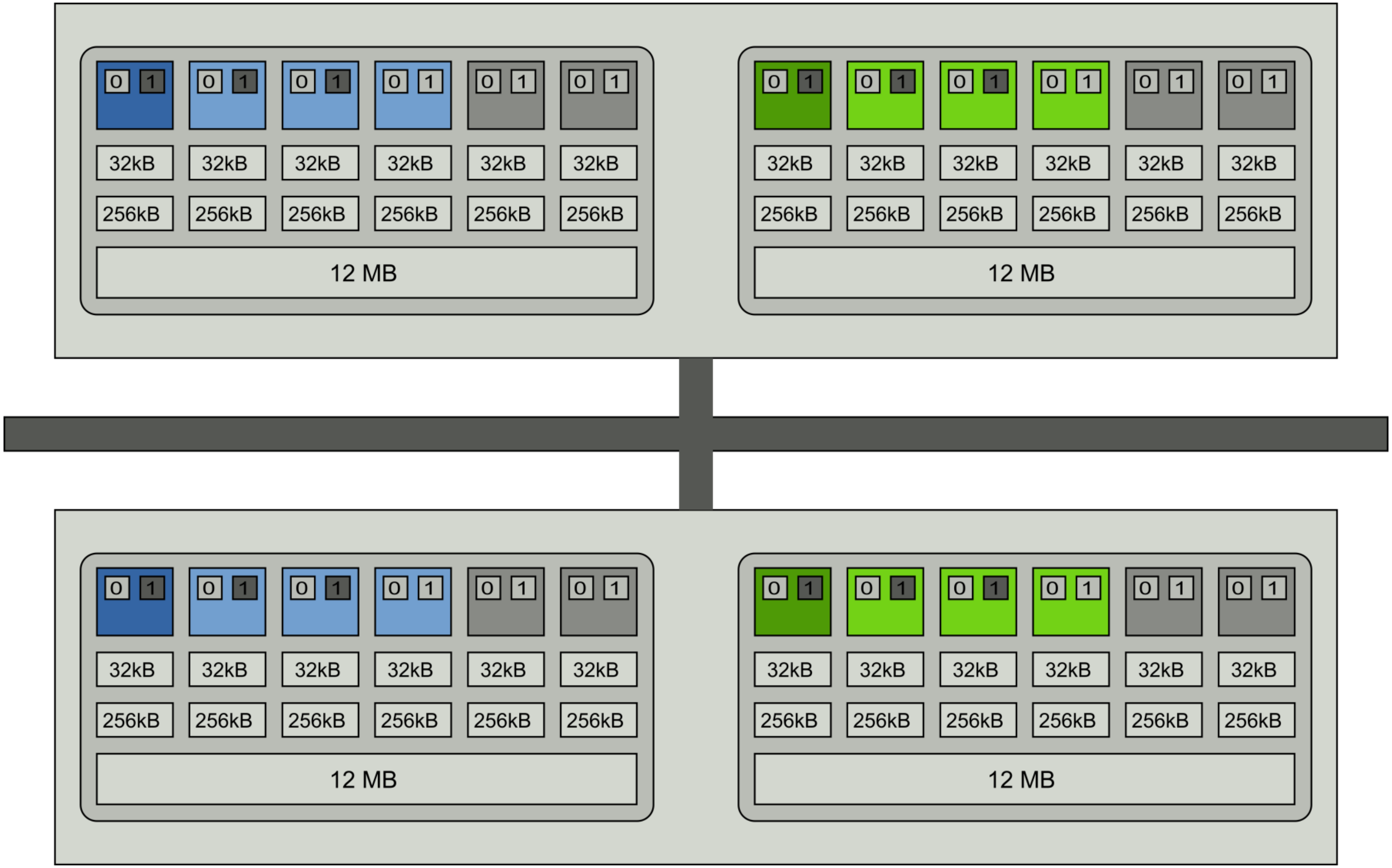
Intel MPI+compiler:

```
OMP_NUM_THREADS=6 mpirun -ppn 2 -np 4 \
     -env I_MPI_PIN_DOMAIN socket -env KMP_AFFINITY scatter ./a.out
```

likwid-mpirun

Communication thread on SMT

```
OMP_NUM_THREADS=5 likwid-mpirun -np 4 -pin S0:0-3,9_S1:0-3,9 ./a.out
```



Outlook for future release
likwid-powermeter

- **Implements Intel RAPL interface (Sandy Bridge)**
- **RAPL (Running average power limit)**

```
CPU name:       Intel Core SandyBridge processor
CPU clock:      3.49 GHz
Thermal Spec Power: 95 Watts
Minimum  Power: 20 Watts
Maximum  Power: 95 Watts
Maximum  Time Window: 0.15625 micro sec
Energy consumed: 126.597 Joules
Power consumed: 63.2983 Watts
```

Test case	Power
4 cores, plain C	45.25 Watt
4 cores, SSE	58.74 Watt
4 cores (SMT), SSE	65.71 Watt
4 cores (SMT), AVX	77.85 Watt

Outlook for future release

Quality improvements

- **Automated test suite for**
  - Feature tests on all supported architectures
  - Quantified overhead
  - Automatic validation and deviation tables for performance groups
- **Improved Documentation (WIKI pages)**
- **Adopt likwid for library use**
- **Document usage of likwid as library**
- **Test likwid-mpirun with all relevant MPI implementations**

Thats it!

Thank you for your attention!

Any Questions?