ERLANGEN REGIONAL COMPUTING CENTER

Some Obstacles on the Way to Reproducibility of Performance Measurements – Tales from the Trenches

Georg Hager, Jan Treibig, Gerhard Wellein
Erlangen Regional Computing Center & Department of Computer Science
Friedrich-Alexander-University of Erlangen-Nuremberg

Repar 2015 Workshop / Euro-Par 2015

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

…outperforms standard implementations, running 2–8 times faster…

…our implementation improves CPI by 1.78 …

…we achieve a parallel efficiency of 98.97 % on 100,000 cores …

Averaging over 200 test cases our data layout outperforms standard layouts by 13.43% …

… our manually tuned code outperforms compiler generated code by 7x …

… in average our stencil compiler increases performance of a wide range of stencils by up to 73.45% …

Often impossible to reproduce (even) qualitatively the findings in many “research” papers with a focus on performance evaluation

One example from our daily business

We performed similar work – our work has been cited and put in context ☺

Nguyen, A.; et al., "3.5-D Blocking Optimization for Stencil Computations on Modern CPUs and GPUs," SC 2010, doi: 10.1109/SC.2010.2

Abstract:

Stencil computation…. **We present a novel 3.5D-blocking algorithm** that performs 2.5D-spatial and temporal blocking of the input grid into on-chip memory for both CPUs and GPUs…. **Our performance numbers are faster or comparable to state-of-the-art-stencil implementations** on CPUs and GPUs…. **For Lattice Boltzmann methods, the corresponding speedup number on CPUs is 2.1X**.

We never achieved speedups higher than 1.4X….

- Purchased the same desktop PC → Result (with our) code not reproducible
- Ask author for code or suggestions → "it is not possible to disclose both.." (authors are from Intel)

What is the novelty / insight from such contributions????

Performance Reproducibility in our daily work

- **Performance measurement** and evaluation of new implementations, optimizations or parallelization approaches

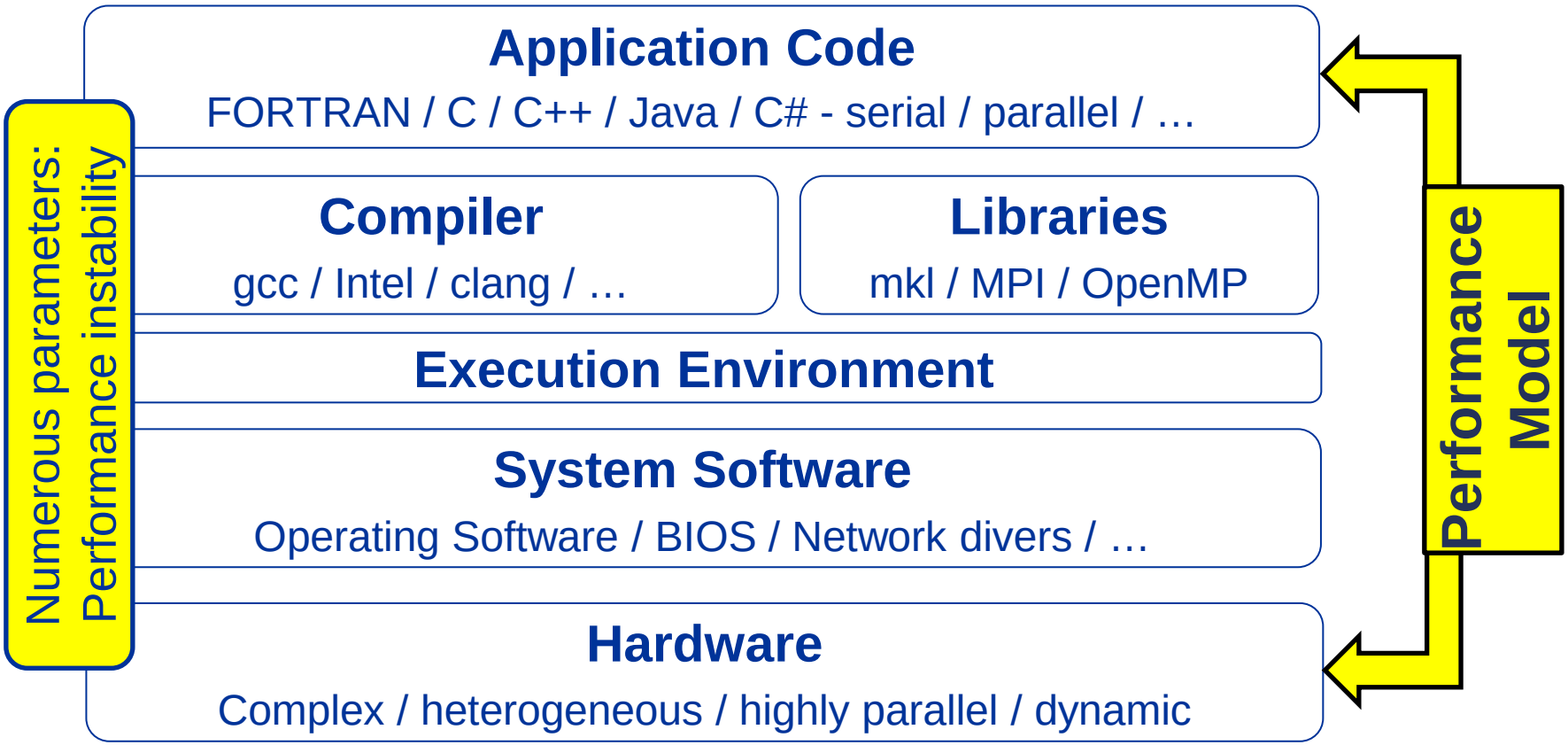
- **Teaching** (I): Results demonstrated in the lectures should be reproducible on identical or similar (next gen.) hardware.
- **Teaching** (II): Reproduce results from students work….

- **Procurements** of new HPC systems: Performance commitments of vendors must be reproducible (over lifetime of the system…)

Observations

- **Performance may change over time**
- **Reproducing performance on similar systems often fails (even qualitatively)**

Software/Hardware stack: Instable performance



Performance models: Roofline[1,2]

HW feature:
Max. Performance

HW feature:
Max. Bandwidth

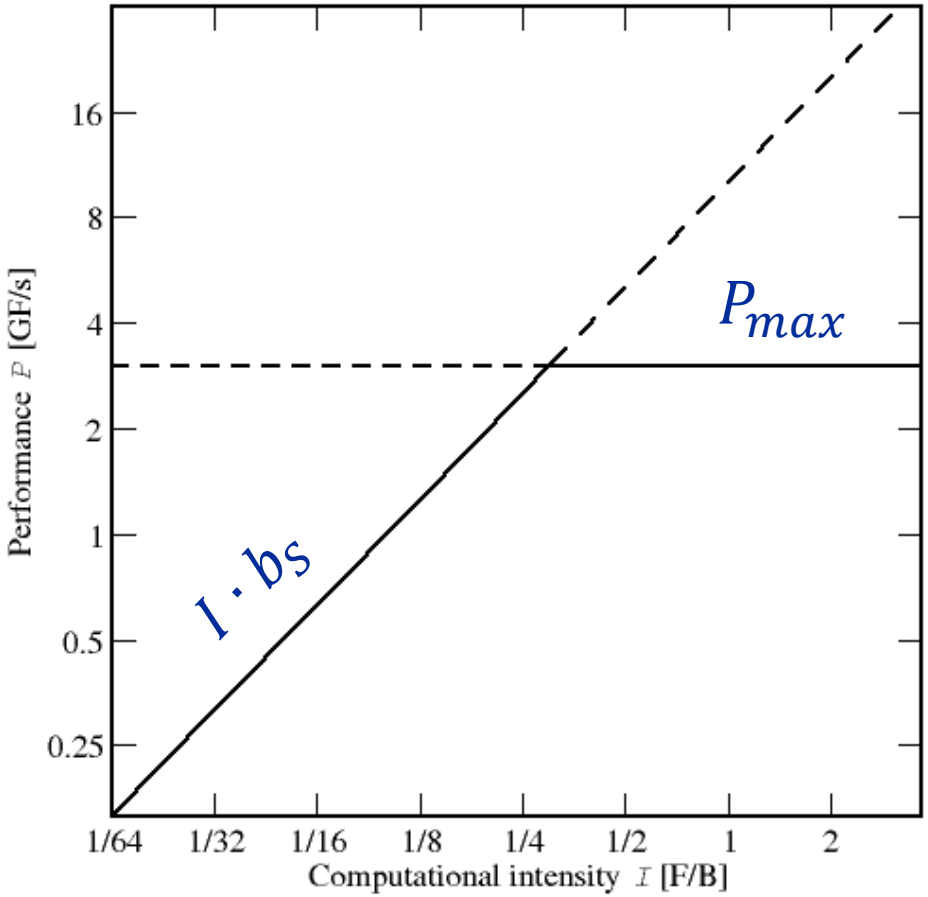
$$P = min(P_{max}, I \cdot b_S)$$

Application feature:
Comp. Intensity [F/B]

Performance bottleneck

- Computation OR
- Data transfer over slowest data path

Apply to full socket / node performance measurements



[1] W. Schönauer: Scientific Supercomputing: Architecture and Use of Shared and Distributed Memory Parallel Computers. (2000)
[2] S. Williams: Auto-tuning Performance on Multicore Computers. UCB Technical Report No. UCB/EECS-2008-164. PhD thesis (2008)

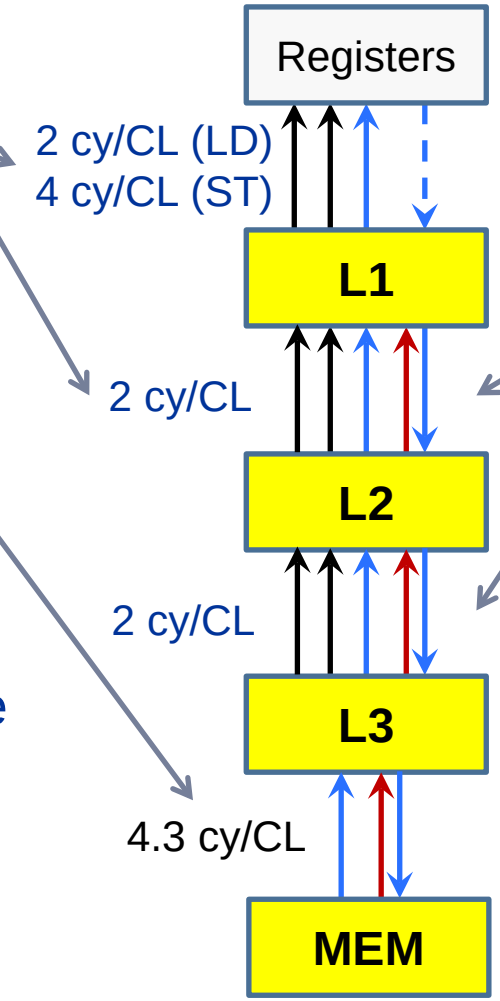
Performance Models: Execution Cache Memory

Machine model:

- Cache transfer times
  Full socket memory bandwidth (b_S)
- Data transfer and core execution overlap
- Data streaming

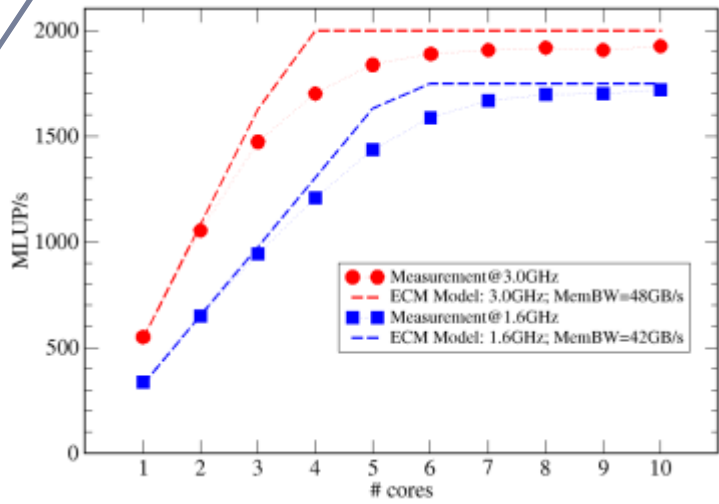
→ Single core performance

→ Linear scaling until bandwidth saturation (if applicable)



Application model:

- Core execution
- All data transfers



G. Hager et al: *Exploring performance and power properties of modern multicore chips via simple machine models*. CCPE (2013), DOI: 10.1002/cpe.3180 (2013).

H. Stengel et al.: *Quantifying performance bottlenecks of stencil computations using the Execution-Cache-Memory model*. Proc. ACM ICS'15, DOI: 10.1145/2751205.2751240

Agenda

Motivation (done)

Hardware – never trust a stranger

Software – the instable universe of parameters and setting

Lesson to be learned – Understand what you are doing

THE HARDWARE

There are well defined hardware specifications for reproducible performance!

Never trust a stranger…

Performance (Intel Xeon E5 2660v2)	
# of Cores	10
# of Threads	20
Processor Base Frequency	2.2 GHz
Max Turbo Frequency	3 GHz
TDP	95 W

Memory Specifications	
Max Memory Size	768 GB
Memory Types	DDR3 800/1066/1333/ 1600/1866

http://ark.intel.com/products/75272/Intel-Xeon-Processor-E5-2660-v2-25M-Cache-2_20-GHz

The hardware –
You often get less/more than ordered…

Main memory (1): The hidden setting

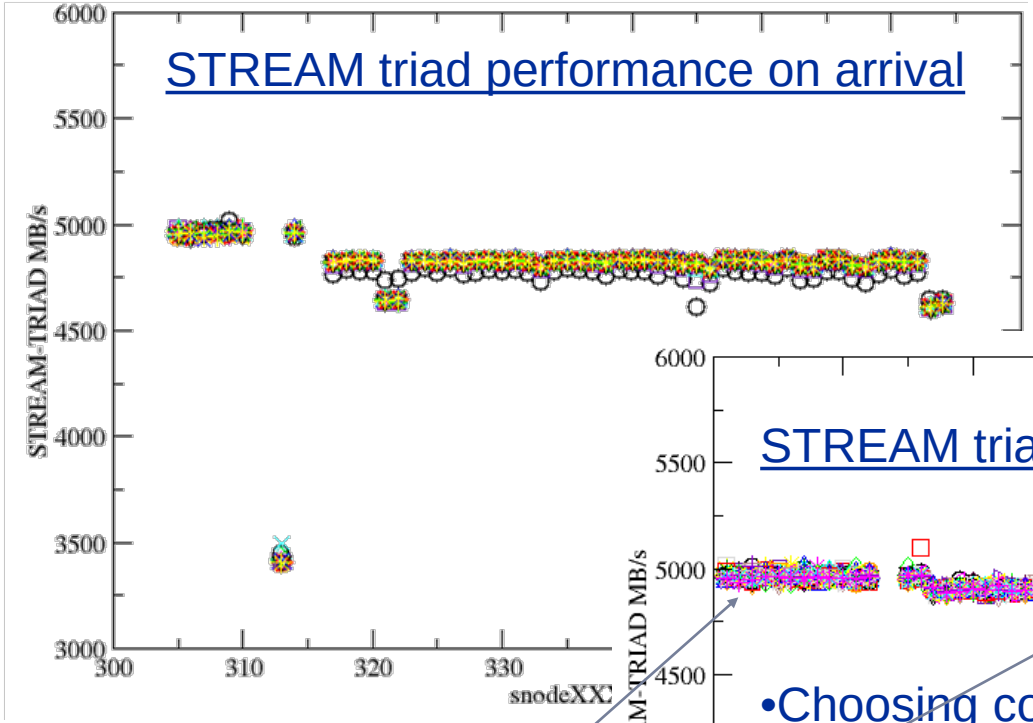
2006: RRZE procurement for a new cluster

- Winner: Intel Xeon 5150 (“Woodcrest”) based IB cluster
- Main memory bandwidth:
  - Fact Sheet: 21.3 GB/s (4* FB-DIMM@667MHz)
  - STREAM: 6.4,…,6.6 GB/s (literature)

  - Vendor commitment: 6.3 GB/s
  - New cluster: 4.8 GB/s
  - Reason: BIOS setting (high DIMM refresh rate)

- Finally we got 2x main memory size to achieve 6.4 GB/s

Main memory (2): New cluster – new problems

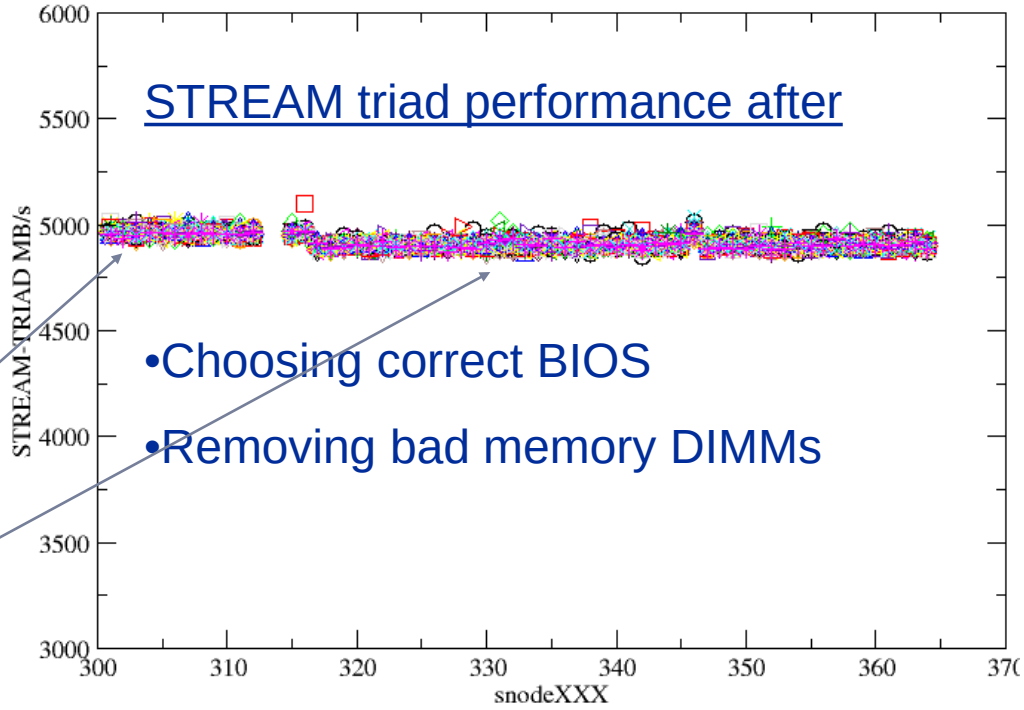


66 single socket nodes:

Intel S3000PT; FSB1066

2.66 GHz Xeon3070 DC

2 deliveries: (16 + 50) nodes



DIMMs:

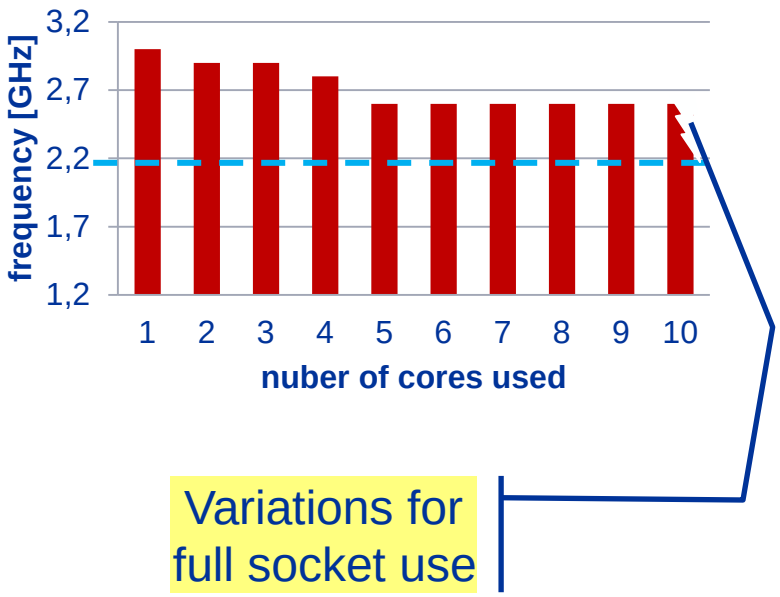
Samsung

Kingston

CPU speed: Turbo boost
2013: New cluster – new story

- 560 nodes with 2 Intel Xeon E5-2660v2 (“Ivy Bridge”) each

- Clock speed if “Turbo Mode” is enabled



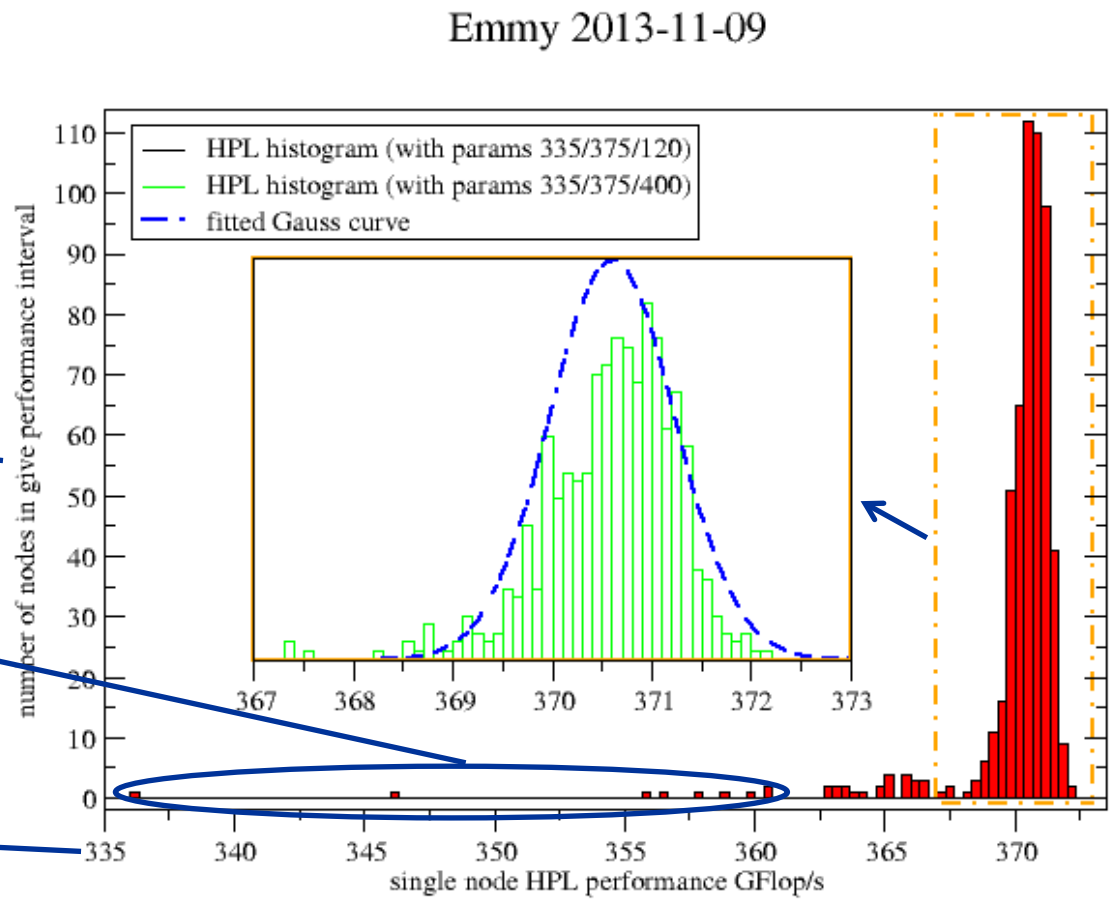
Variations for full socket use

Performance (Intel Xeon E5 2660v2)	
# of Cores	10
# of Threads	20
Processor Base Frequency	2.2 GHz
Max Turbo Frequency	3 GHz
TDP	95 W
…	…
Intel® Turbo Boost Technology ‡	2.0

http://ark.intel.com/products/75272/Intel-Xeon-Processor-E5-2660-v2-25M-Cache-2_20-GHz

CPU speed: Turbo boost
2013: New cluster – new story

- Strong correlation between clock speed & LINPACK
- Run 560 independent copies of LINPACK



#nodes in performance interval

Omitting those nodes – overall LINPACK increases

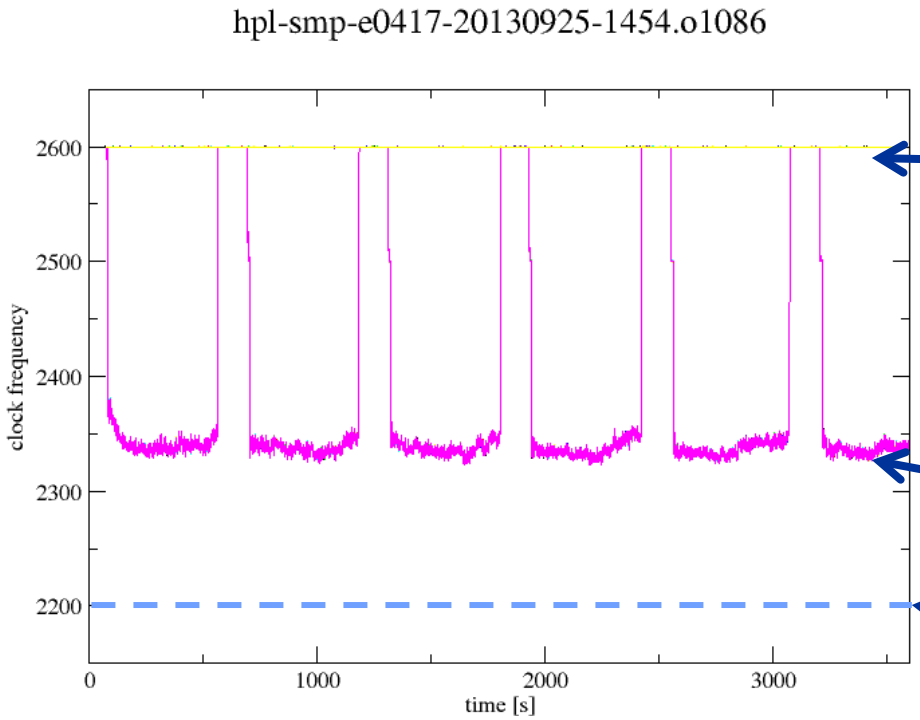
Single node LINPACK performance

CPU speed: Turbo boost
2013: New cluster – new story

For our cluster: Clock speed variation ←→ chip manufacturing

You may even have good and bad processor chips within a node:

Clock speed within one node during LINPACK run



Socket 0

Socket 1

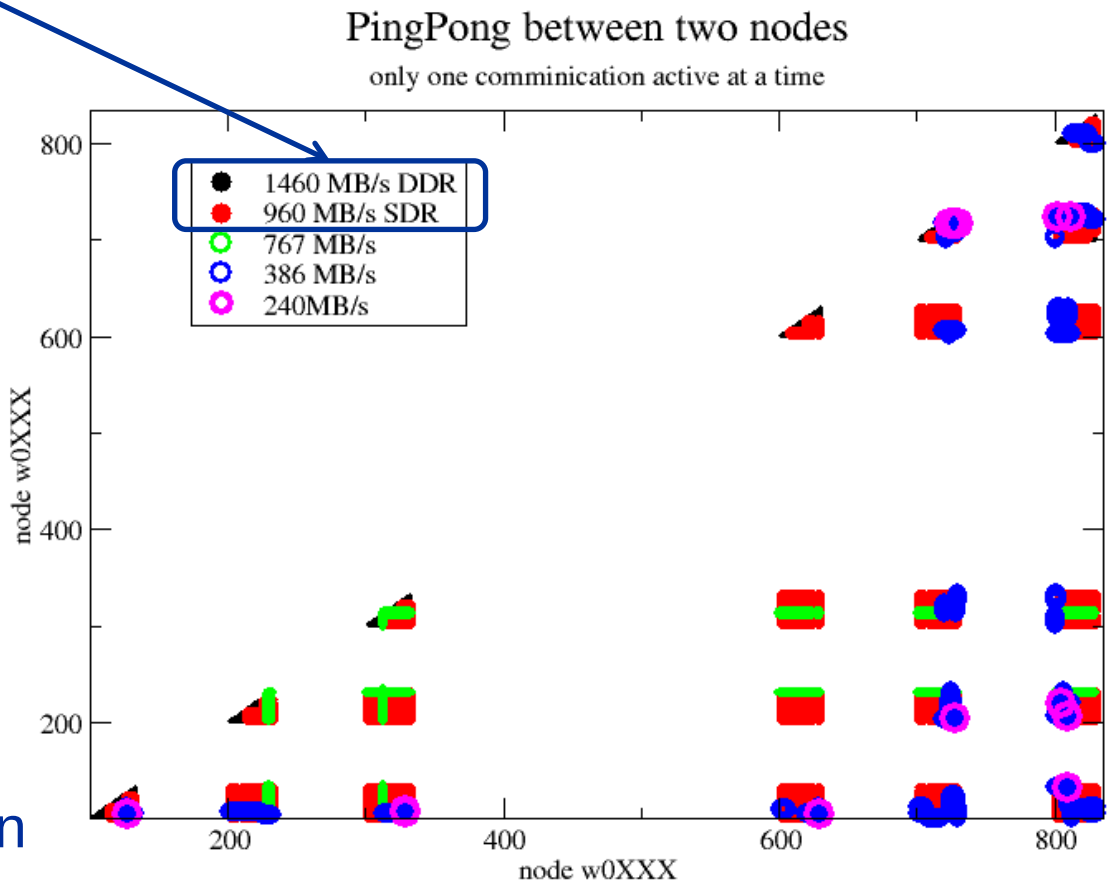
That's the number from the type plate!

Interconnect – did you ever check your hardware?

Back to our 2006 cluster – first DDR switch with SDR backplane

→ Expect ~1000 MB/s or 1500 MB/s for PingPong MPI tests!

→ Several reboots
& firmware upgrades
solved the problem

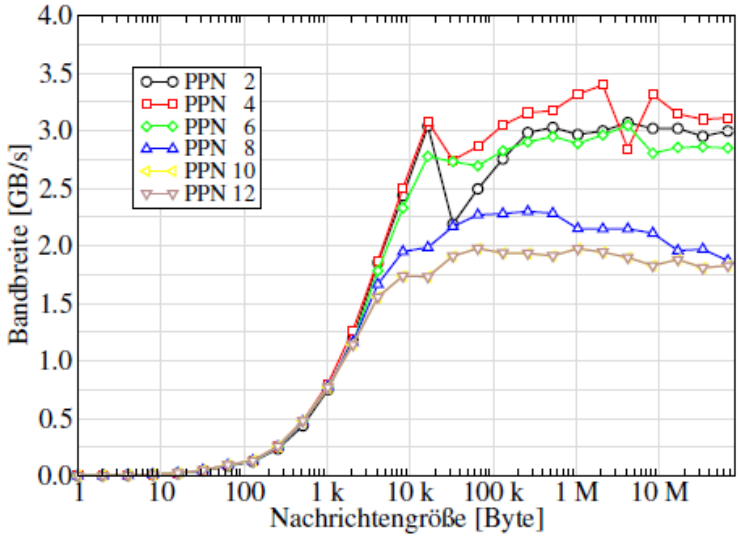


SuperMUC@LRZ:
3000 optical IB cables
with “bad silicon” - after
some months of operation

Interconnect – what is your network speed?

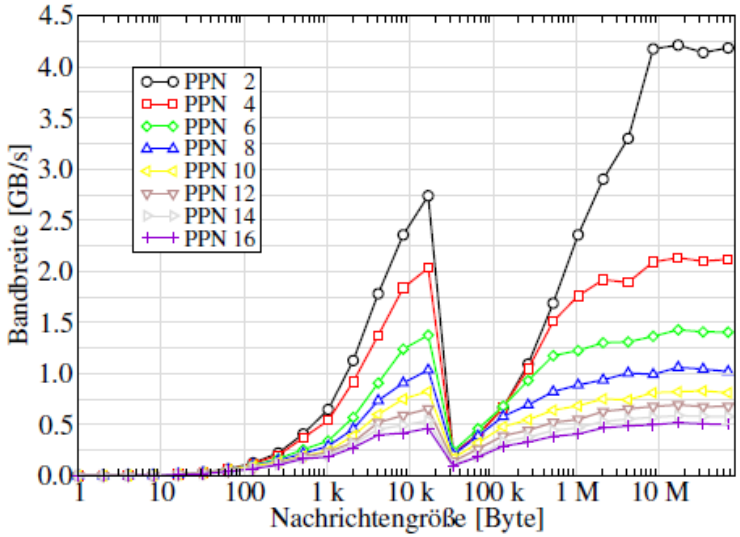
Exchange benchmark between three nodes on

QDR Infiniband cluster (RRZE)



(a) Lima, 2,66 GHz

FDR Infiniband cluster (LRZ Munich)



(b) SuperMUC, 2,7 GHz

If message sizes of 20k,…,500 k are relevant → SuperMUC we have a problem (probably a software problem)

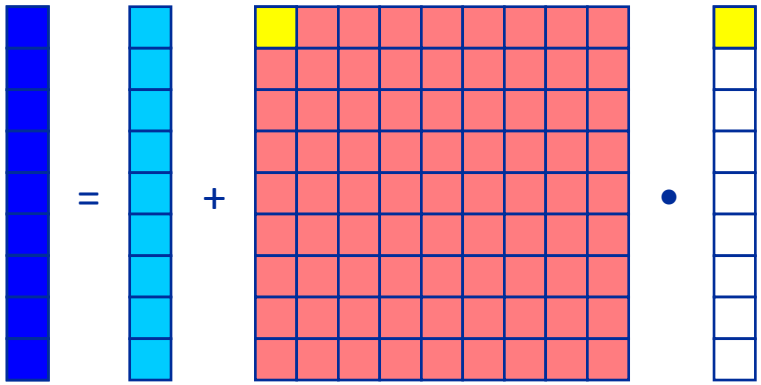
Intel Corp.: Intel MPI benchmarks, 2014. http://software.intel.com/en-us/articles/intel-mpi-benchmarks

THE SOFTWARE

The instable universe of parameters and settings

A simple case study with some surprises…

A simple case study: Dense matrix vector multiplication aka DGEMV



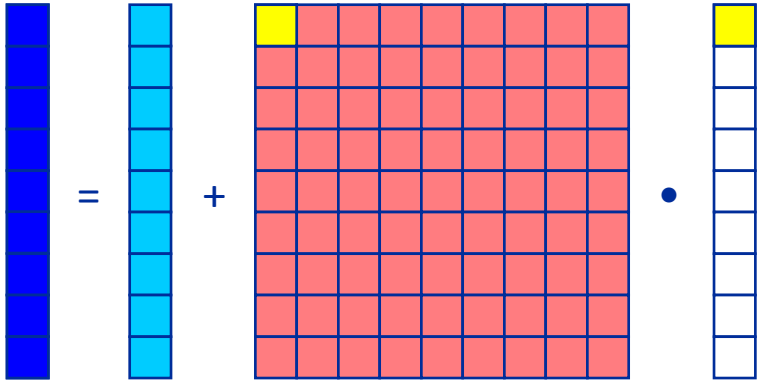

```
void dmvm(int n, int m, double *lhs, double *rhs, double *mat){

  for(r=0; r<n; ++r)
    offset=m*r;

    for(c=0; c<m; ++c)

      lhs[r] += mat[c + offset]*rhs[c];
}
```

Inner loop invariant

Linear Access (Cache)

Linear Access (Memory)

A simple case study: Dense matrix vector multiplication aka DGEMV



- One test:
  - Call **dmvm** `iter` times
  - Measure wallclocktime (s)
  - Report performance (MF/s)
- Run multiple tests to check for fluctuations / runtime instabilities
- Two test cases:
  - Cache only: m=n=10^3 – Main Memory: m=n=10^4

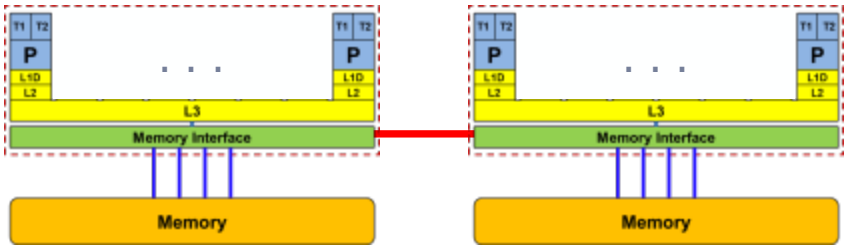
```
void dmvm(int n, int m, double *lhs, double *rhs, double *mat){

  for(r=0; r<n; ++r)
    offset=m*r;
    for(c=0; c<m; ++c)
        lhs[r] += mat[c + offset] * rhs[c]; }
```

  - Computational intensity (on modern CPUs) for m=n=10^4 ?

$$I = \frac{2\ Flop}{8\ Byte}$$

Testsytem

- **Haswell compute-node: 2-socket server**
  - Intel Xeon E5-2695 v3 – **2.30 GHz** (fixed for all measurements) – AVX2 (FMA)
  - **14 cores** per chip; 32 kB L1; 256 kB; (17 + 17) MB L3
  - **C**luster **o**n **D**ie: **Enabled** (4 NUMA domains with 7 cores (17 MB L3) each)
  - **D**ynamic **F**requency **S**caling: **Disabled**
  - 8 * 8 GB DDR4-2133 / 4 channels → Peak BW= 68.3 GB/s
  - Ubuntu 14.04.2 LTS (GNU/Linux 3.13.0-57-generic x86_64)
  - Intel Compiler: **`icc`** 15.0.1 20141023
  - gcc Compiler: **`gcc`** 4.8.4
  - Thread pinning via: **`likwip-pin`**
    (see https://github.com/rrze-likwid/likwid)
  - CPU frequency is fixed to baseline frequency



Single core performance – compiler

```
void dmvm(int n, int m, double *lhs, double *rhs, double *mat){

  for(r=0; r<n; ++r)
    offset=m*r;
    for(c=0; c<m; ++c)
      lhs[r] += mat[c + offset]*rhs[c]; }
```

- **Let's optimize for AVX2:**
  - gcc –Ofast –march=core-avx2 –c
  - icc –Ofast –xHost –c

Compile on Haswell

- **Single core peak performance:** 2.3 * 2 * 2 * 4 GF/s = **36.8 GF/s**
- **Measurements:**

All data in cache

Matrix from main memory

	gcc	icc	ECM Model
$m=n=10^3$	427 MF/s	6930 MF/s	7360 MF/s
$m=n=10^4$	418 MF/s	3190 MF/s	3530 MF/s

Single core performance – from high-level to low-level language

```
void dmvm(int n, int m, double *lhs, double *rhs, double *mat){

  for(r=0; r<n; ++r)
    offset=m*r;
    for(c=0; c<m; ++c)
      lhs[r] += mat[c + offset]*rhs[c]; }
```

gcc –Ofast –march=core-avx2 –S

```
.L6:
        vmovsd  (%rcx,%rax,8), %xmm0
        vmovsd  (%rdx), %xmm1
        vfmadd132sd     (%r8,%rax,8),
        addq    $1, %rax
        vmovsd  %xmm0, (%rdx)
        cmpl    %eax, %esi
        jg      .L6
```

icc –Ofast –xHost –S

```
                                                rbp rsi r8 r11 r13 r
..B1.24:                        # Preds ..B1.24 ..B1.23
        vmovupd   (%r14,%rsi,8), %ymm10                         #21.17
        vmovupd   32(%r14,%rsi,8), %ymm11                       #21.17
        vmovupd   64(%r14,%rsi,8), %ymm12                       #21.17
        vmovupd   96(%r14,%rsi,8), %ymm13                       #21.17
        vmovupd   128(%r14,%rsi,8), %ymm14                      #21.17
        vfmadd231pd (%r8,%rsi,8), %ymm10, %ymm9                 #21.7
        vfmadd231pd 32(%r8,%rsi,8), %ymm11, %ymm5               #21.7
        vmovupd   160(%r14,%rsi,8), %ymm15                      #21.17
        vmovupd   192(%r14,%rsi,8), %ymm10                      #21.17
        vmovupd   224(%r14,%rsi,8), %ymm11                      #21.17
        vfmadd231pd 64(%r8,%rsi,8), %ymm12, %ymm4               #21.7
        vfmadd231pd 96(%r8,%rsi,8), %ymm13, %ymm3               #21.7
        vfmadd231pd 128(%r8,%rsi,8), %ymm14, %ymm2              #21.7
        vfmadd231pd 160(%r8,%rsi,8), %ymm15, %ymm0              #21.7
        vfmadd231pd 192(%r8,%rsi,8), %ymm10, %ymm1              #21.7
        vfmadd231pd 224(%r8,%rsi,8), %ymm11, %ymm6              #21.7
        addq      $32, %rsi                                     #18.5
        cmpq      %rcx, %rsi                                    #18.5
        jb        ..B1.24       # Prob 82%                      #18.5
```

Always remember:
THIS is what the processor finally executes!

Single core performance – Pointer aliasing

```
void dmvm(int n, int m, double *restrict lhs, double *restrict
rhs, double *restrict mat){
… }
```

C99 only
-std=c99

	gcc -Ofast –march=core2-avx	gcc + restrict keyword	icc -Ofast -xHost
$m=n=10^3$	427 MF/s	3620 MF/s	6930 MF/s
$m=n=10^4$	418 MF/s	3220 MF/s	3190 MF/s

- **Intel compiler generates multiple versions!**
  (-fno-alias frequently helps for more complex C/C++ code)
- **There is a huge search space for gcc, e.g.**

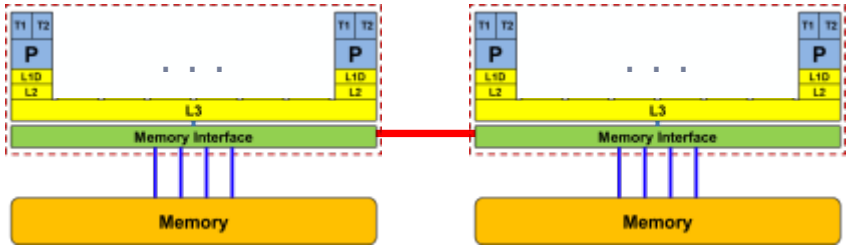
	gcc -Ofast	gcc -Ofast +restrict
$m=n=10^3$	1520 MF/s	3070 MF/s
$m=n=10^4$	1470 MF/s	2600 MF/s

Always remember:
Compilers are not stable with respect to performance!

Node performance – just link with multithreaded libraries

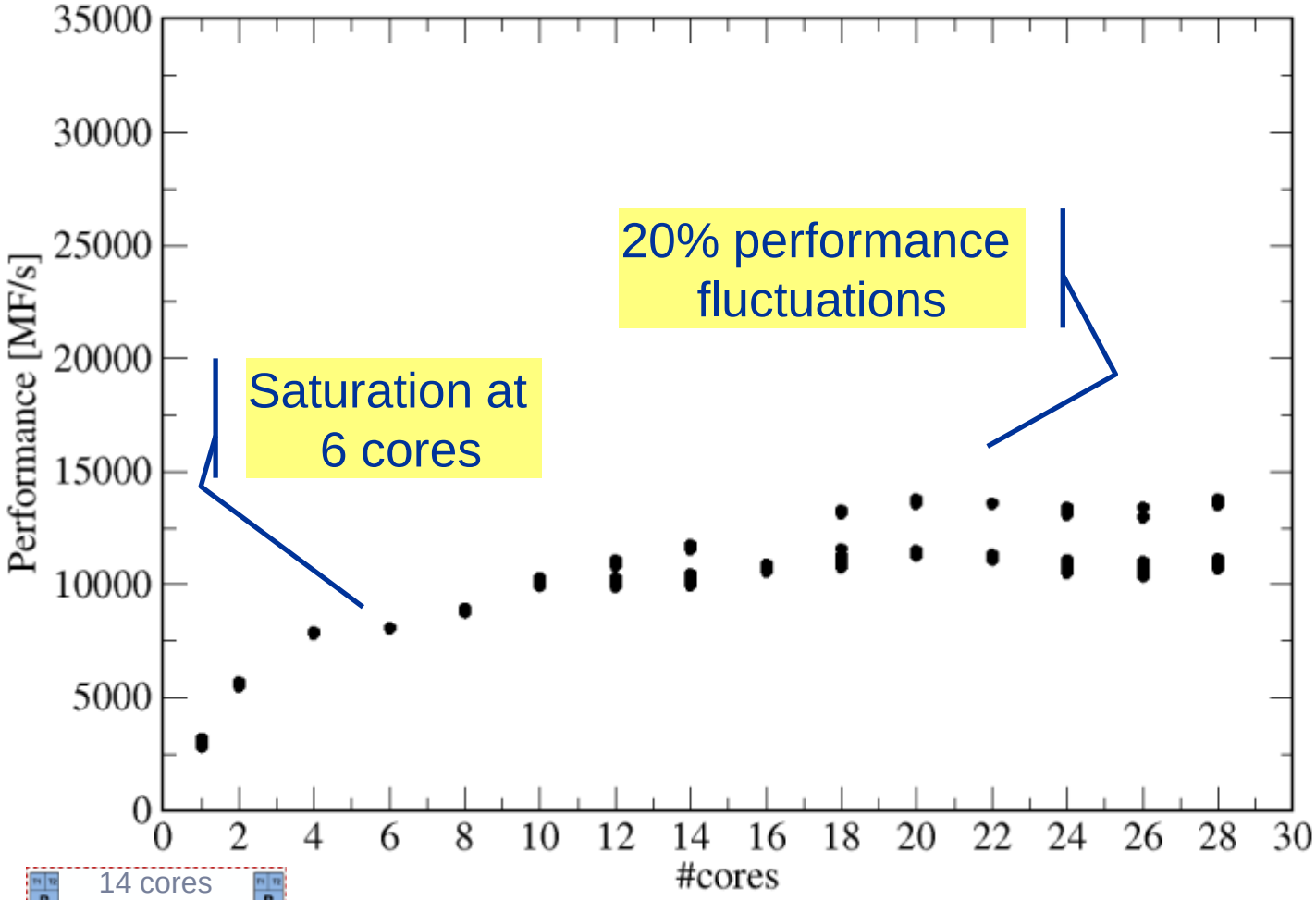
```
void dmvm(int n, int m, double *lhs, double *rhs, double *mat){
#pragma omp for private(offset,c) schedule(static) {
  for(r=0; r<n; ++r)
    offset=m*r;
    for(c=0; c<m; ++c)
      lhs[r] += mat[c + offset]*rhs[c];
} }
```

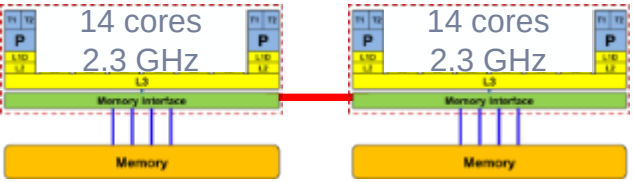
- Run scaling experiment with OMP_NUM_THREADS = 1 (,2,4,6,8,…,28)
- Fill socket first / No SMT
- Run 10 tests
  (with `iter` dmvm calls each)



Node performance – just link with multithreaded libraries

`iter = 200`

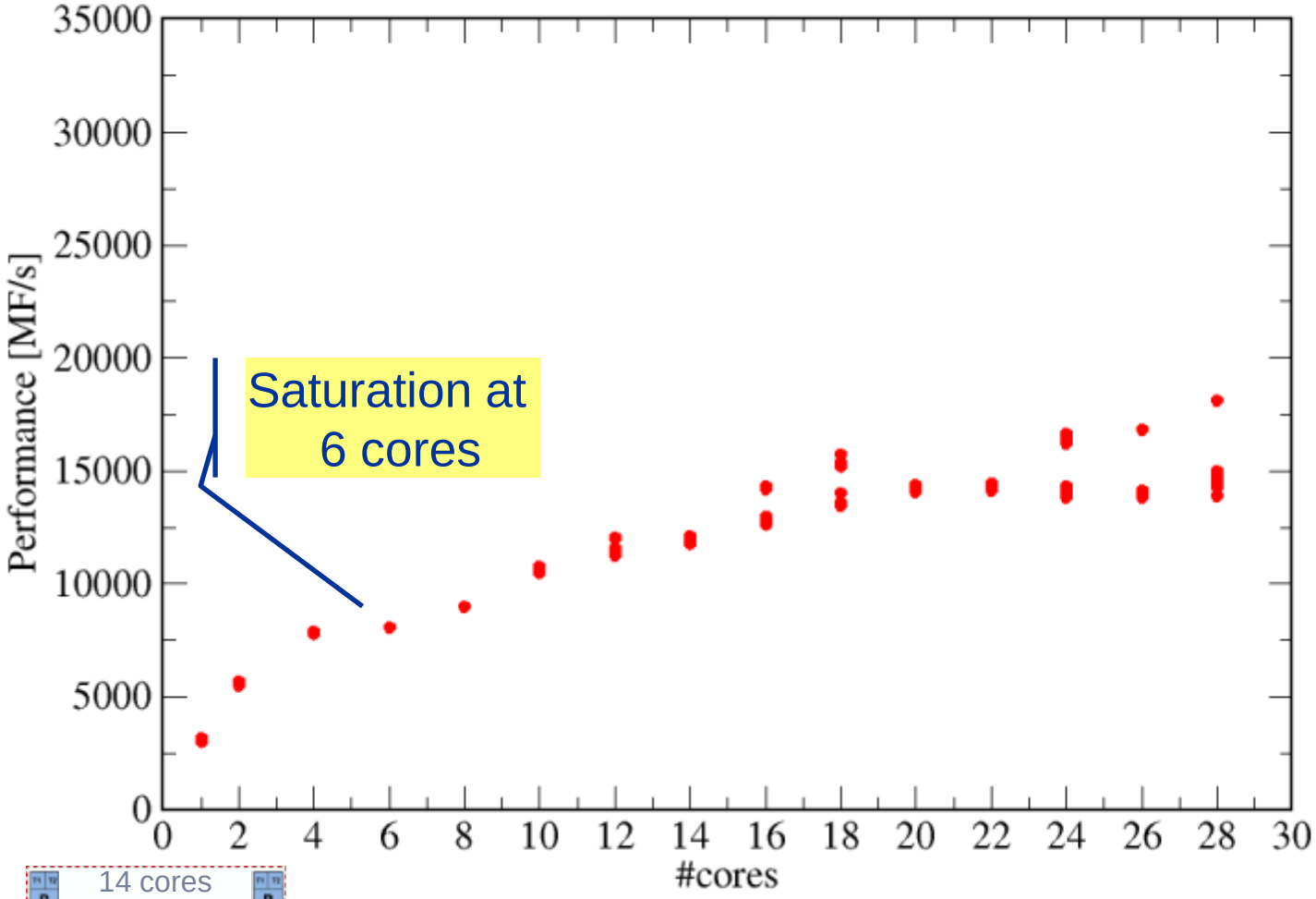


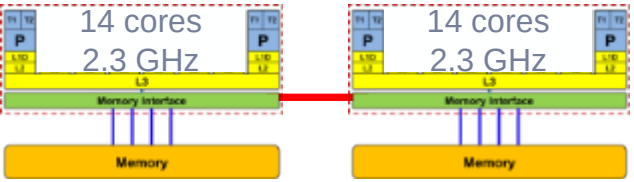


OMP_NUM_THREADS

Node performance – just link with multithreaded libraries

`iter = 500`

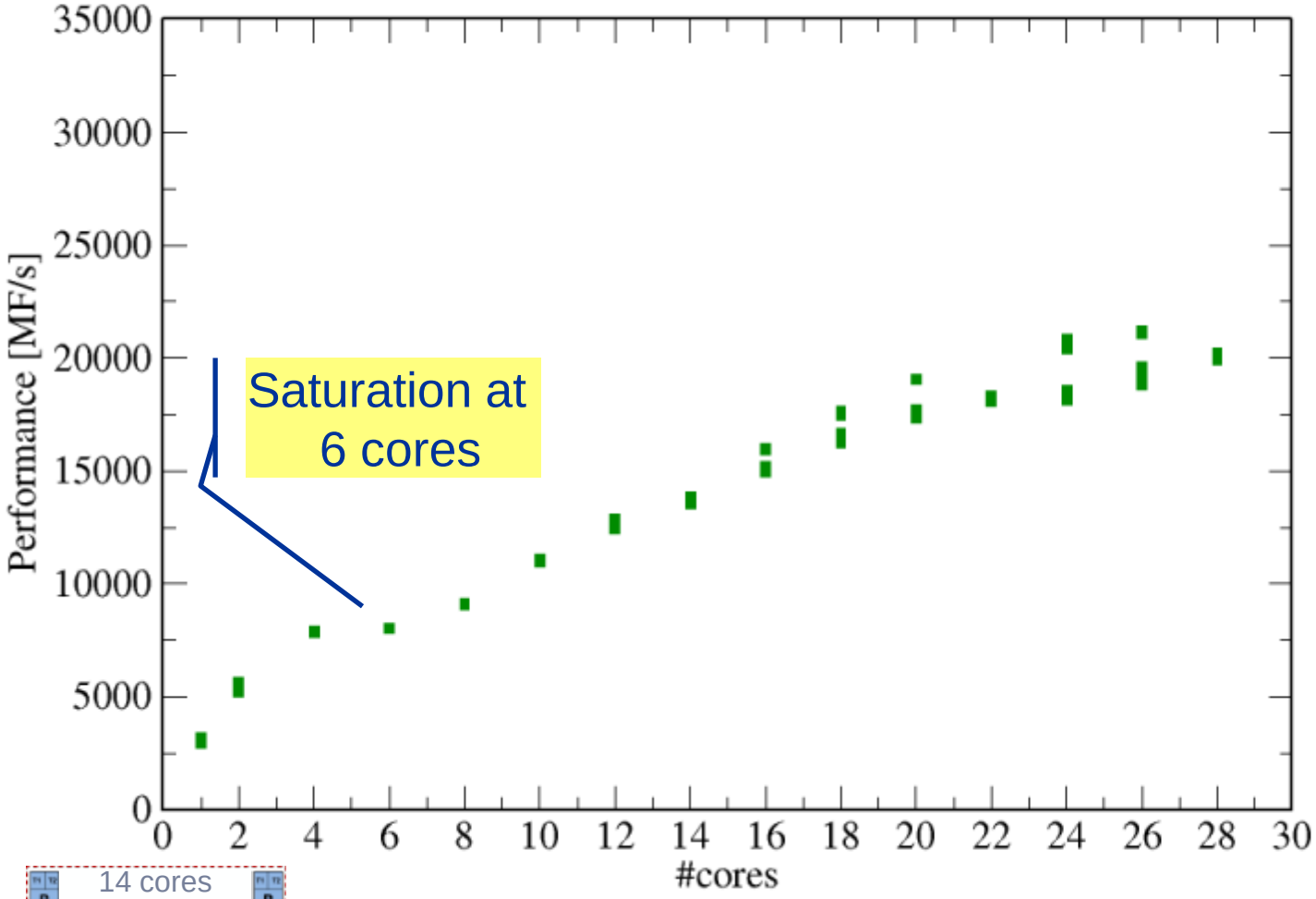


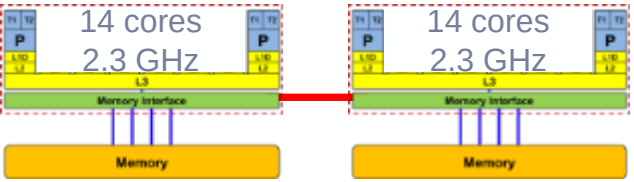


OMP_NUM_THREADS

Node performance – just link with multithreaded libraries

`iter = 1000`

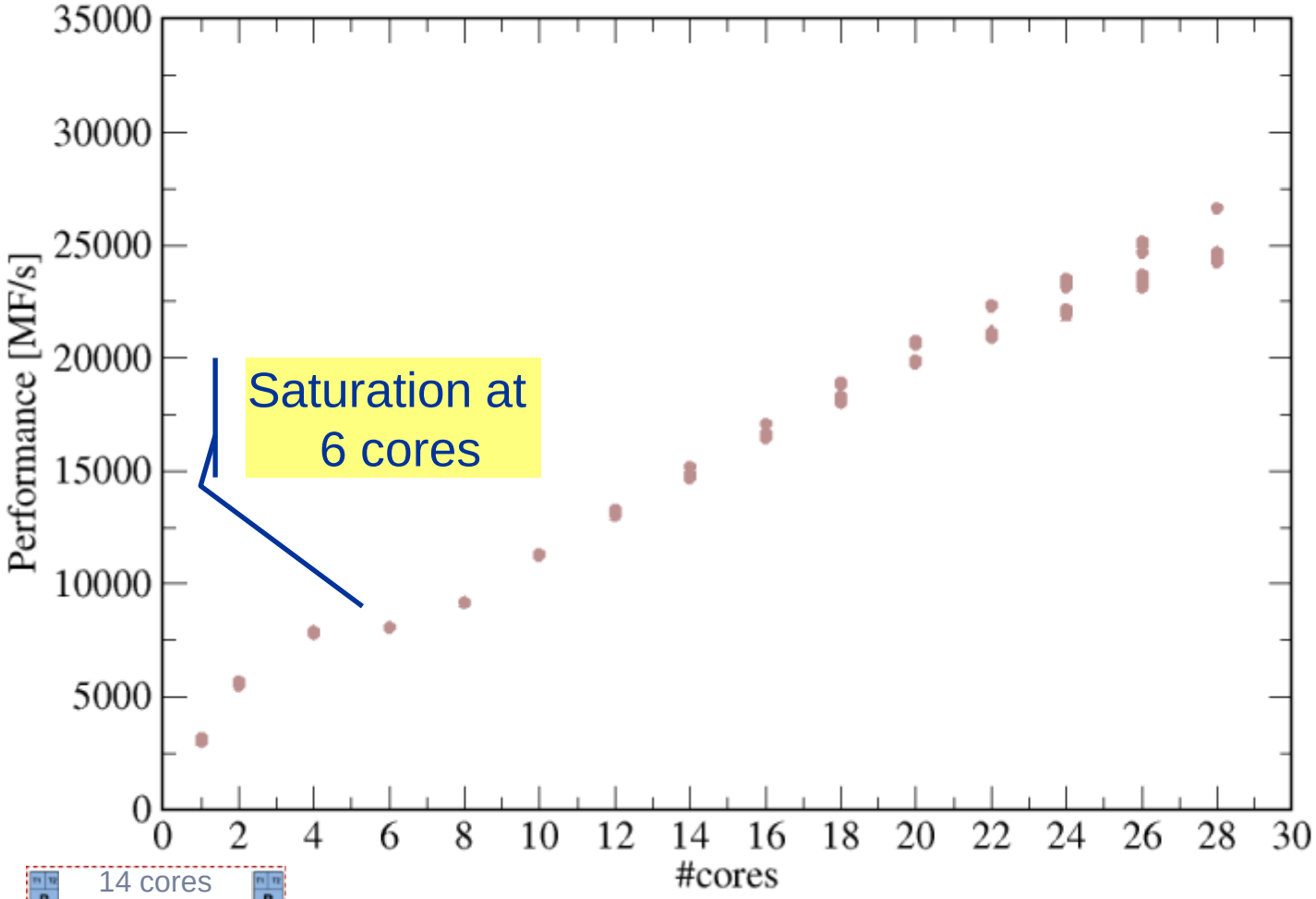


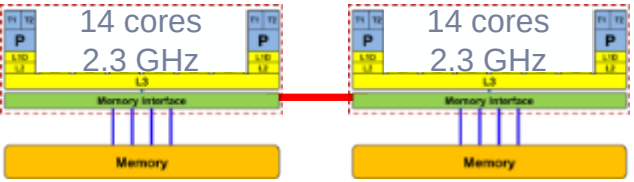


OMP_NUM_THREADS

Node performance – just link with multithreaded libraries

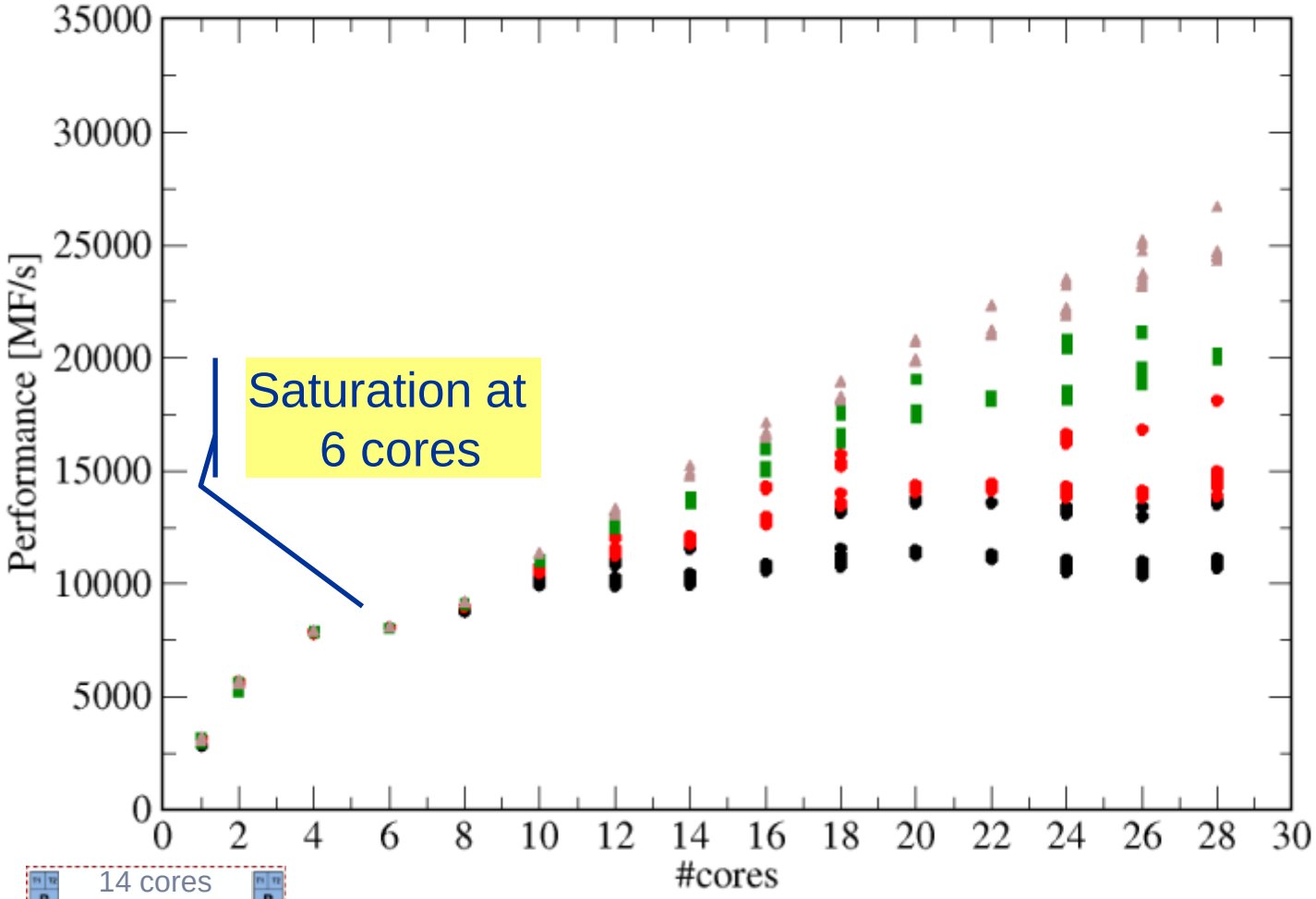
`iter = 2000`

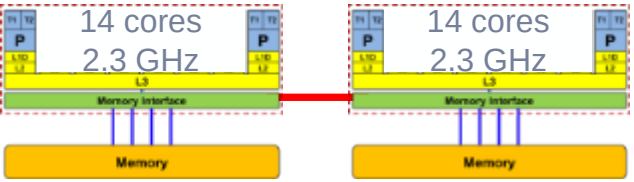




OMP_NUM_THREADS

Node performance – just link with multithreaded libraries
All at once – Any suggestions?

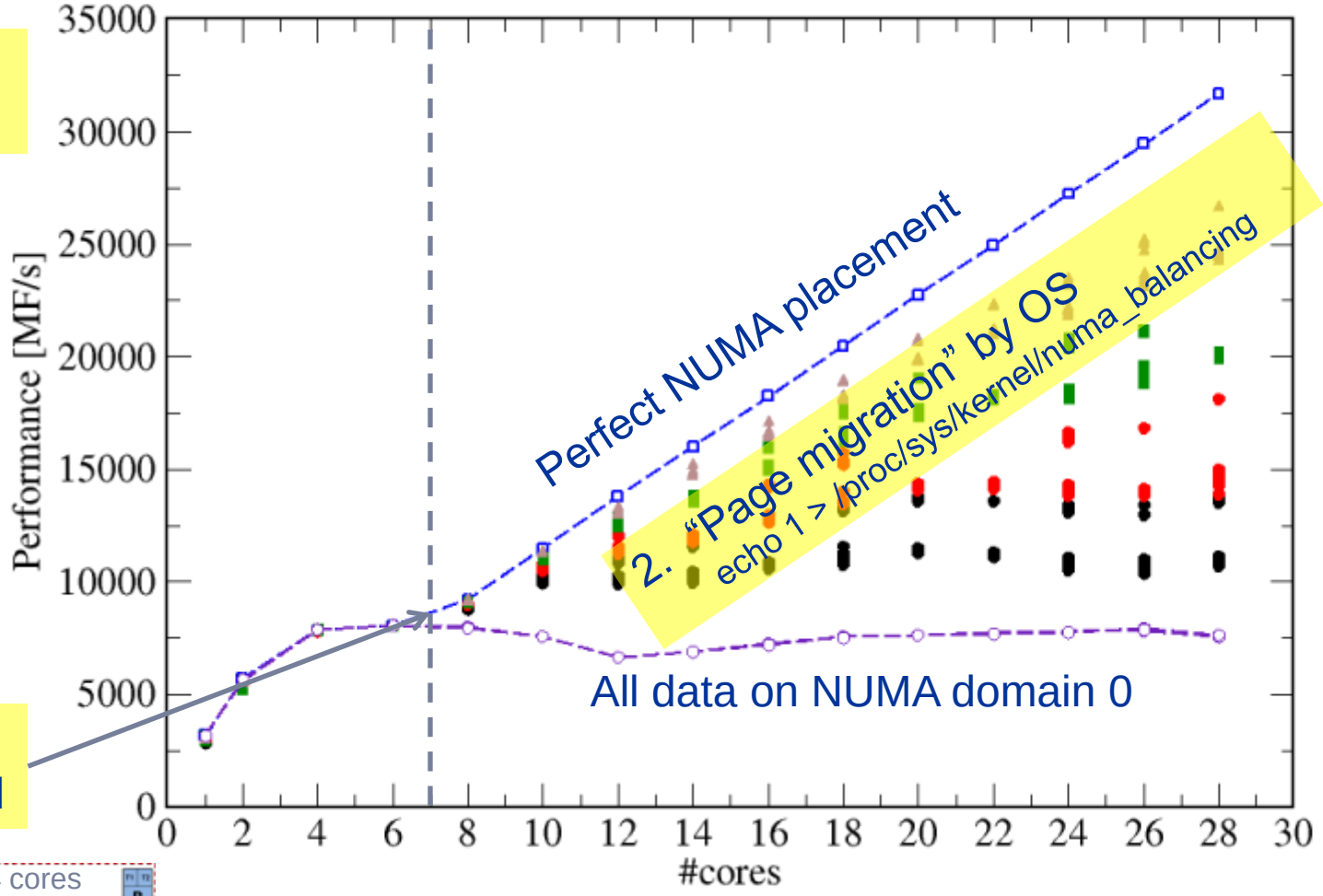




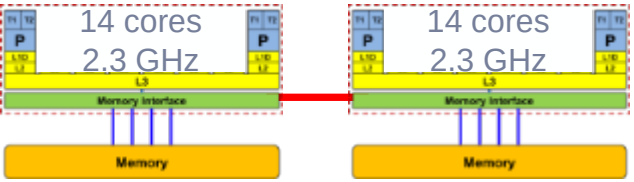
OMP_NUM_THREADS

Node performance – just link with multithreaded libraries
The problems explained

1. Serial matrix initialization



3. Cluster on Die Mode: enabled



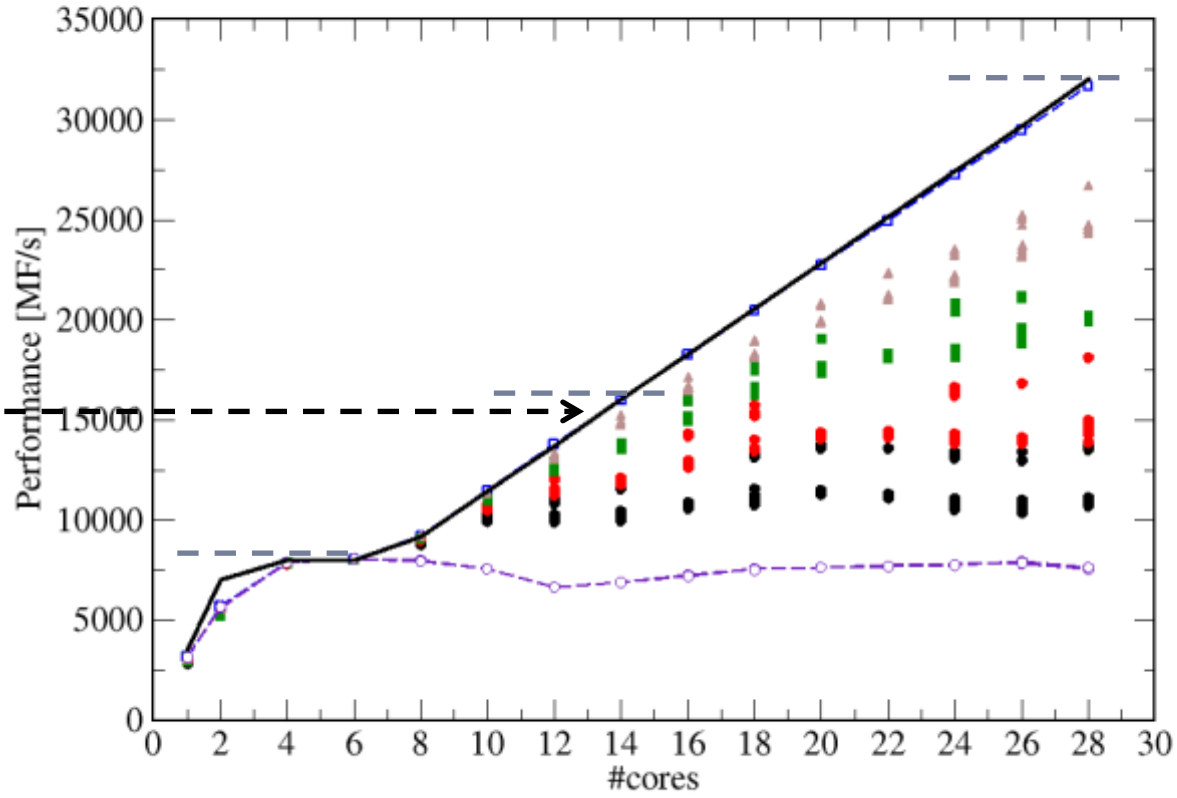
Node performance – just link with multithreaded libraries
Performance models

Roofline model:

- Assume $b_S = 64\ GB/s$ (socket)

ECM model:

- Assume $b_S = 64\ GB/s$ (socket)
- Cluster on Die: Enabled
- NUMA initialization



Typically we start with the model and use performance measurement for model validation or modification (of code or model)

Yet another setting - a bit more challenging data access pattern

```
void dmvm(int n, int m, double *lhs, double *rhs, double *mat){
  for(r=0; r<n; ++r)
    for(c=0; c<m; ++c)
      lhs[r] += mat[r + c * n] *rhs[c];
}
```

High stride access to matrix data

$m=n=10^3$; iter=10

10 – core IVB@3GHz

	Xeon E5-2690v2	Xeon E5-2695v3
Performance	113 MF/s	600 MF/s
Mem – L3 (MEM)	0.84 GB	0.81 GB
L3 – L2 (L3)	19.3 GB	6.85 GB
L2 – L1 (L2)	13.7 GB	7.25 GB
DTLB	100 Mevents	7 Mevents
OS	Ubuntu 12.04.5	Ubuntu 14.04.2
Huge Pages[2]	**madvise**	**always**

Processor counter analysis[1] per `dmvm` call

[1]Using `likwid-perfctr` (see https://github.com/rrze-likwid/likwid)

[2]See `cat /sys/kernel/mm/transparent_hugepage/enabled`

Performance Experiments Summary

- Node calibration for full transparency and control
  (Fix clock speeds, SMT – off, pin threads/processes,…)
- Testbed characterization
  (STREAM / LINPACK values, Compilers, options, OS, BIOS,…)
- Controlled interference “injection”
  (use different compiler /-switches, OS,…)
- Experimentation “Methodology”
  (use scripts/automation, cvs,…)
- Software tools for experiment control
  (hwloc, likwid, PAPI,…)

This is the end

Lessons learned for performance experiments

- Hardware/software layers: Turn off any “clever” dynamic behavior (remember Walid’s talk?!)
- Clearly understand what you are doing and what is happening – only then you are able to tell others (remember Walid’s talk?!)
- Performance models are helpful for (qualitative) reproducibility
- Our vision: Performance model is key part – performance measurements are for model validation only

Insights not Numbers!

Acknowledgement

We gratefully thank these funding agencies to support our work on Performance Modelling / Performance Engineering

- German Research Foundation (DFG) through SPPEXA projects ESSEX & EXASTEEL

- Federal Ministry of Education and Research (BMBF) through projects hpCADD, FETOL, SKALB

- Competence Network for HPC in Bavaria through project OMI4papps