GHOST, Performance Engineering, SpMVM.

And 42.

Georg Hager
Erlangen Regional Computing Center (RRZE)
University of Erlangen-Nuremberg
Germany

Workshop “Sparse Solvers for Exascale: From Building Blocks to Applications”
March 23-25, 2015
Greifswald, Germany

Outline

Some words on the GHOST design

The SELL-C-σ matrix format

The Performance Engineering (PE) process

Analytically modeling spMVM performance

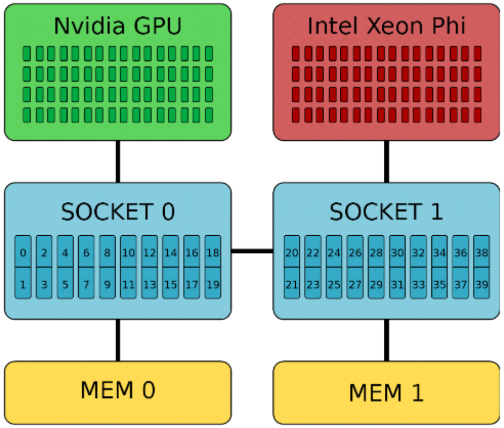
FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
FACULTY OF ENGINEERING

Software for sparse linear algebra

Requirements and possible solutions

Challenges for programming current & future systems

- **Heterogeneity**
  - CPU/GPU/Phi
  - Addressed by multi-target building blocks & functional parallelism & load balancing & optimized data formats & MPI+X



- **System topology**
  - Memory hierarchy, bottlenecks, affinity, ccNUMA, distributed memory
  - Addressed by bottleneck awareness & full control of affinity mechanisms & MPI+X

- **Communication**
  - Latency/bandwidth, network topology
  - Addressed by bottleneck awareness & functional parallelism

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
FACULTY OF ENGINEERING

GHOST design principles

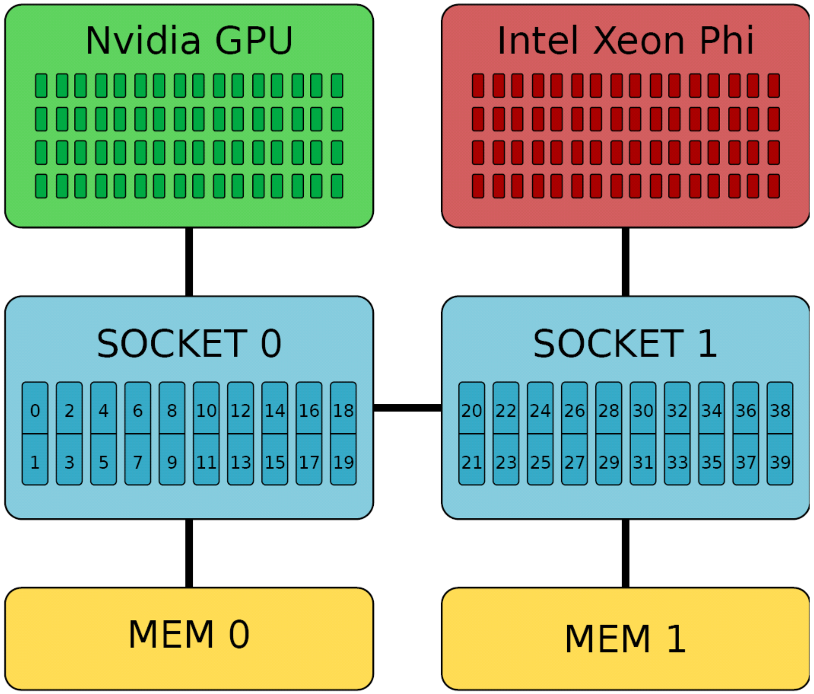
Not reinventing the wheel

GHOST design guidelines

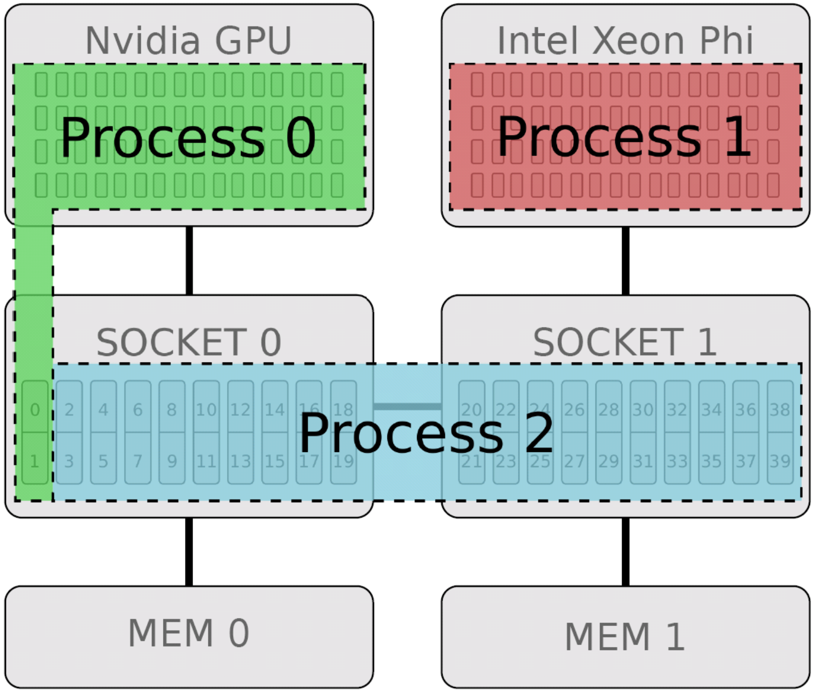
FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
FACULTY OF ENGINEERING

- **Enable fully heterogeneous operation**
  - CPU + GPU + Phi
- **Limit automation**
  - The user needs to know what is going on
- **Do not force dynamic tasking**
  - Allow access locality optimizations
- **Do not force C++ or an entirely new language**
  - Think “pragmatic”
  - We need to get a job done
- **Stick to the well-known “MPI+X” paradigm**
  - X = OpenMP, CUDA for now
- **Allow functional parallelism**
  - Spawn asynchronous tasks for almost anything
- **Allow for strict thread/process-core affinity**
  - Affinity matters!

Example: Hardware Affinity



Heterogeneous node



“Minimum” process distribution to address this architecture

Example: Work Distribution

1. Heterogeneity has to be considered for work distribution
   ➔ more power = more work

2. Work distribution for data-parallel approach: Divide the matrix row-wise between workers

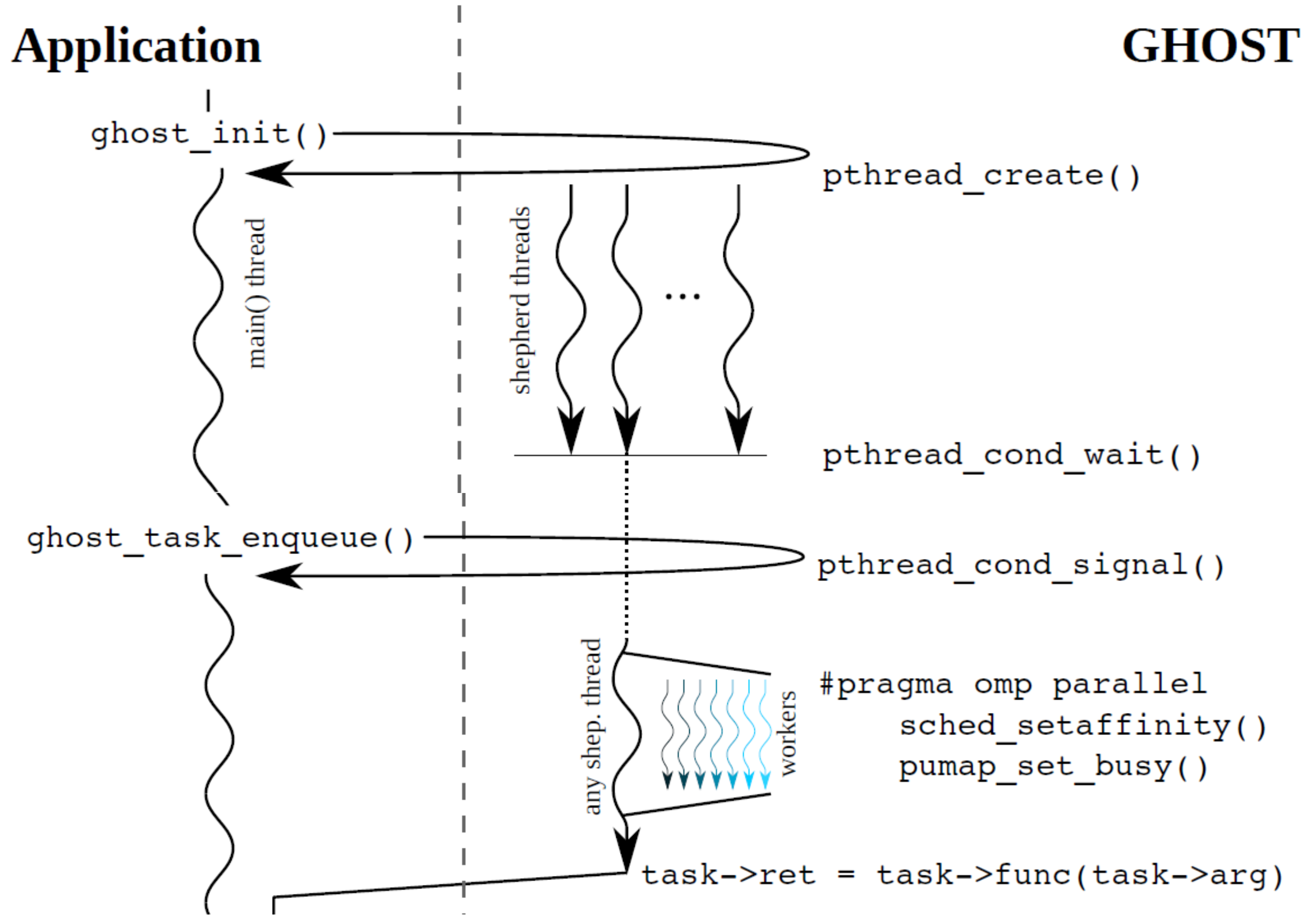
3. Example for memory-bound algorithm and a CPU-GPU node:
   1. GPU's memory bandwidth maybe 4x as large as CPU's
   2. Sparse matrix has, e.g., 10 million rows

   ➔ GPU gets assigned 8 million rows
   ➔ CPU gets assigned 2 million rows

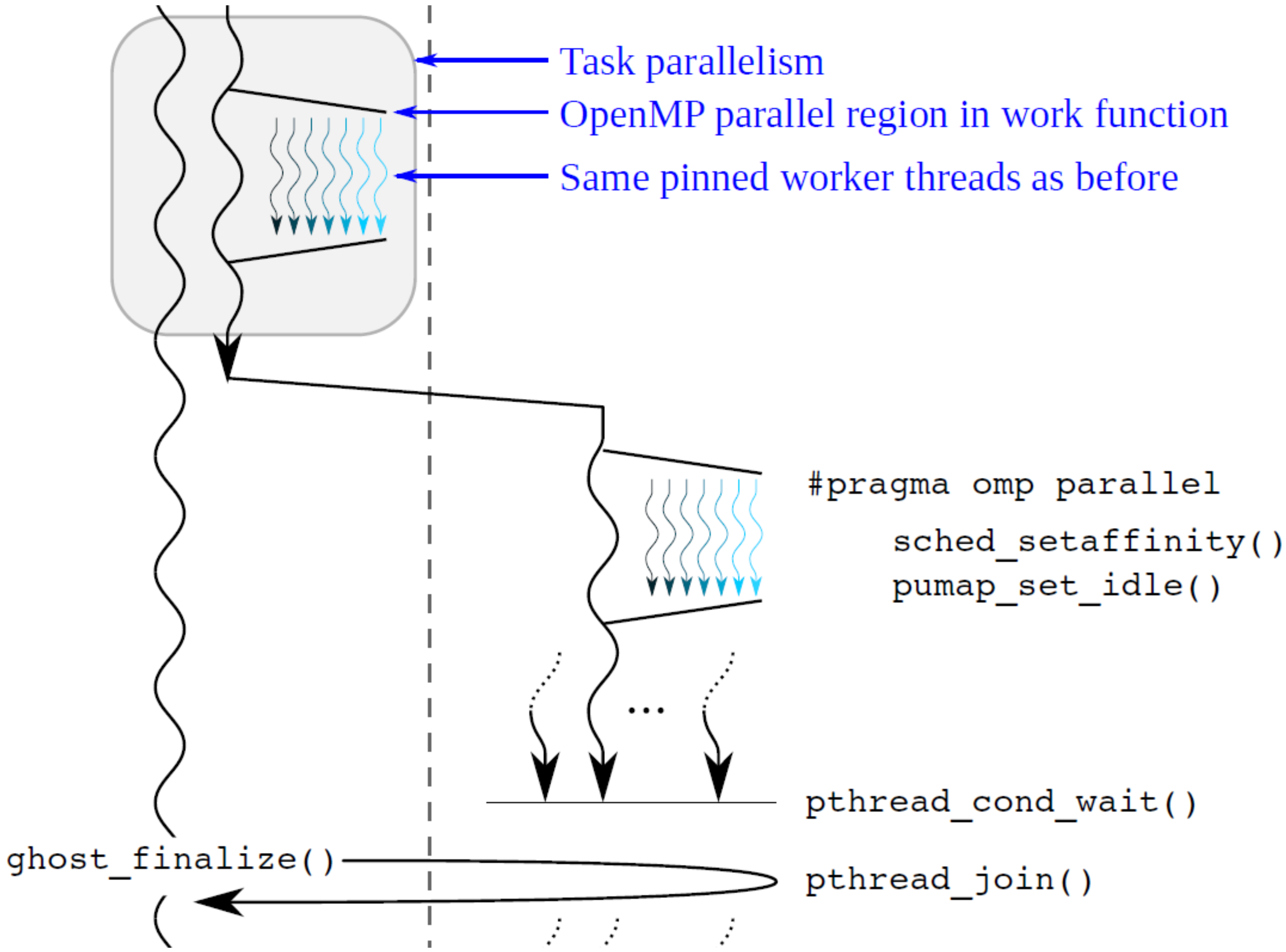
Example: Async task model

```
// define task: checkpointing with 1 thread
ghost_task_create(&chkpTask, 1, curTask->LD, &chkp_func, \
        (void *)&chkp_func_args, GHOST_TASK_DEFAULT, NULL, 0);

// define task: compute with N-1 threads
ghost_task_create(&compTask, curTask->nThreads-1, \
        curTask->LD, &comp_func, (void *)&comp_func_args, \
        GHOST_TASK_DEFAULT, NULL, 0);

// initiate tasks
ghost_task_enqueue(chkpTask); ghost_task_enqueue(compTask);
// wait for completion
ghost_task_wait(chkpTask); ghost_task_wait(compTask);
```

GHOST internals: Thread startup & task enqueue



GHOST internals: Task execute & finalize

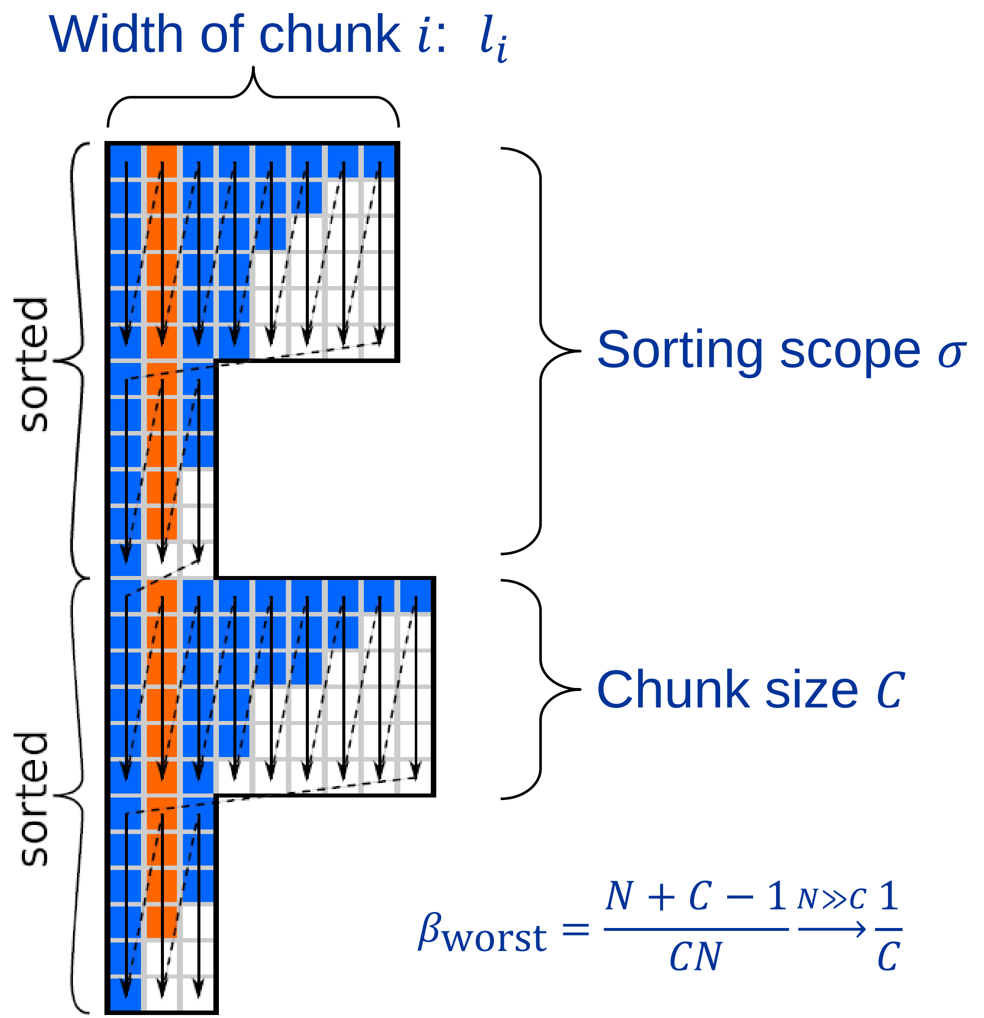


SELL-C-σ

Constructing SELL-C-σ

1. Pick chunk size C (guided by SIMD/T widths)
2. Pick sorting scope σ
3. Sort rows by length within each sorting scope
4. Pad chunks with zeros to make them rectangular
5. Store matrix data in “chunk column major order”
6. “Chunk occupancy”: fraction of “useful” matrix entries

$$\beta = \frac{N_{nz}}{\sum_{i=0}^{N_c} C \cdot l_i}$$



SELL-6-12
β=0.66

$$\beta_{\mathrm{worst}} = \frac{N + C - 1}{CN} \xrightarrow{N \gg C} \frac{1}{C}$$

Matrix characterization

"Corner case" matrices from "Williams Group":

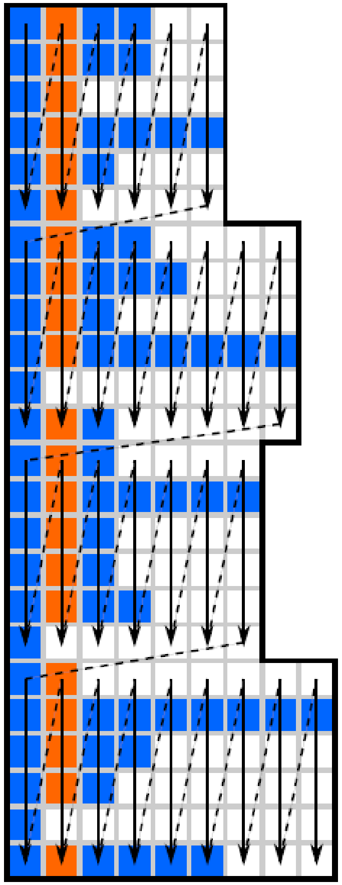
Test case	N	N_{nz}	N_{nzr}	density	$\beta_{\sigma=1}^{C=16}$	$\beta_{\sigma=256}^{C=16}$
RM07R	381,689	37,464,962	98.16	2.57e-04	0.63	0.93
kkt_power	2,063,494	14,612,663	7.08	3.43e-06	0.54	0.92
Hamrle3	1,447,360	5,514,242	3.81	2.63e-06	1.00	1.00
ML_Geer	1,504,002	110,879,972	73.72	4.90e-05	1.00	1.00

Remaining matrices:

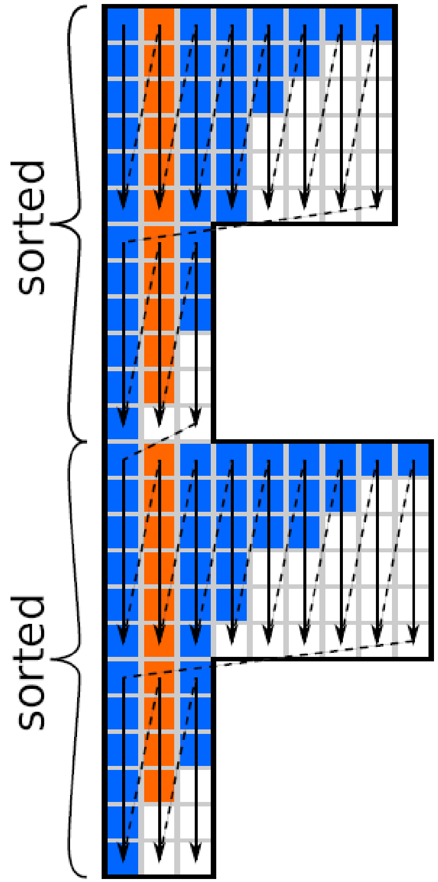
pwtk	217,918	11,634,424	53.39	2.45e-04	0.99	1.00
shipsec1	140,874	7,813,404	55.46	3.94e-04	0.89	0.98
consph	83,334	6,010,480	72.13	8.65e-04	0.94	0.97
pdb1HYS	36,417	4,344,765	119.31	3.28e-03	0.84	0.97
cant	62,451	4,007,383	64.17	1.03e-03	0.90	0.98

…

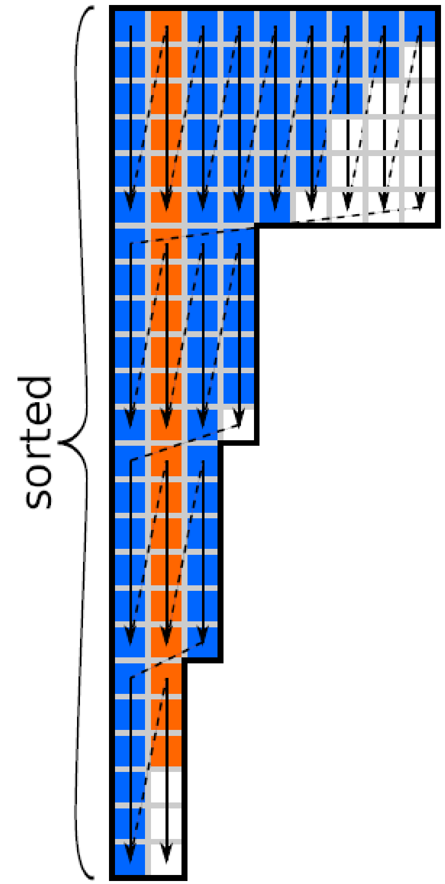
Variants of SELL-C-σ



SELL-6-1
β=0.51



SELL-6-12
β=0.66



SELL-6-24
β=0.84

The Performance Engineering (PE) process

Systematic performance analysis and pattern-guided optimization

Performance Engineering Process: Analysis

Algorithm/Code Analysis

Hardware/Instruction set architecture

Microbenchmarking

Application Benchmarking

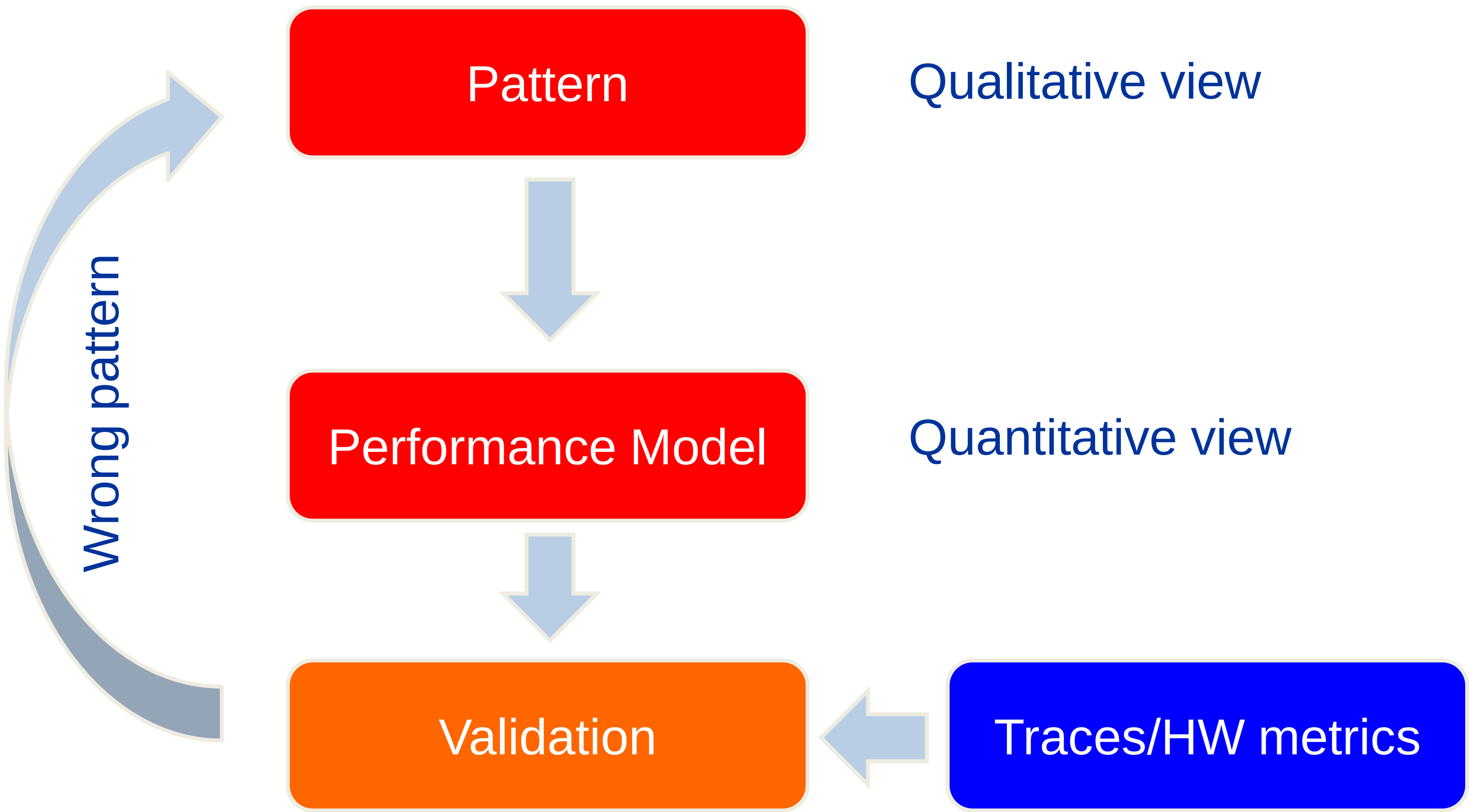
The set of input data indicating a pattern is its **signature**

Pattern

Performance patterns are typical performance limiting motifs

Step 1 **Analysis:** Understanding observed performance

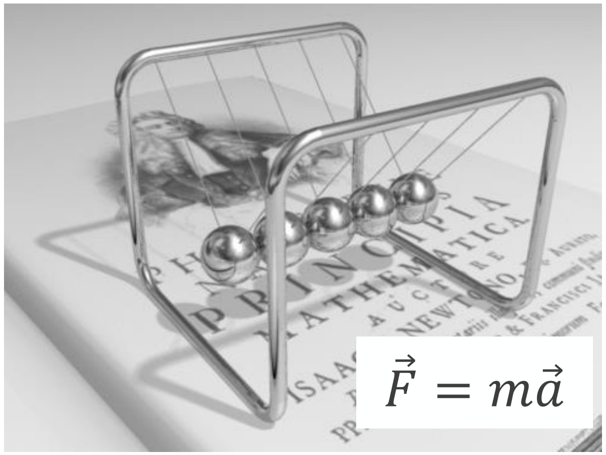
Performance Engineering Process: Modelling



Step 2 **Formulate Model:** Validate pattern and get quantitative insight.

Models in physics

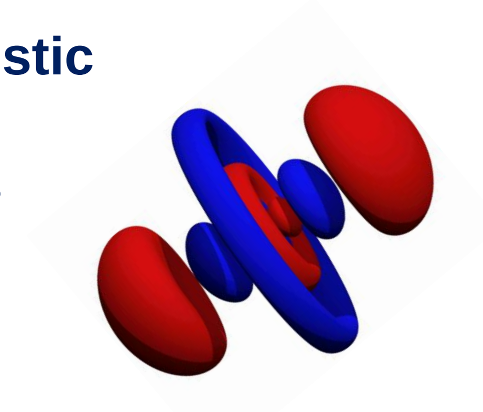
Newtonian mechanics



$$\vec{F} = m\vec{a}$$

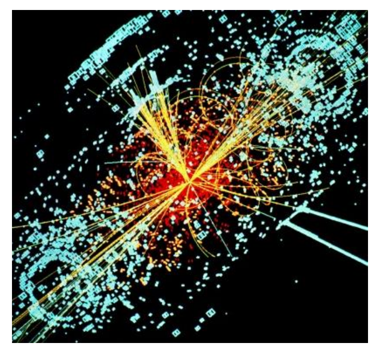
Fails @ small scales!

Nonrelativistic quantum mechanics



$$i\hbar \frac{\partial}{\partial t} \psi(\vec{r}, t) = H\psi(\vec{r}, t)$$

Fails @ even smaller scales!



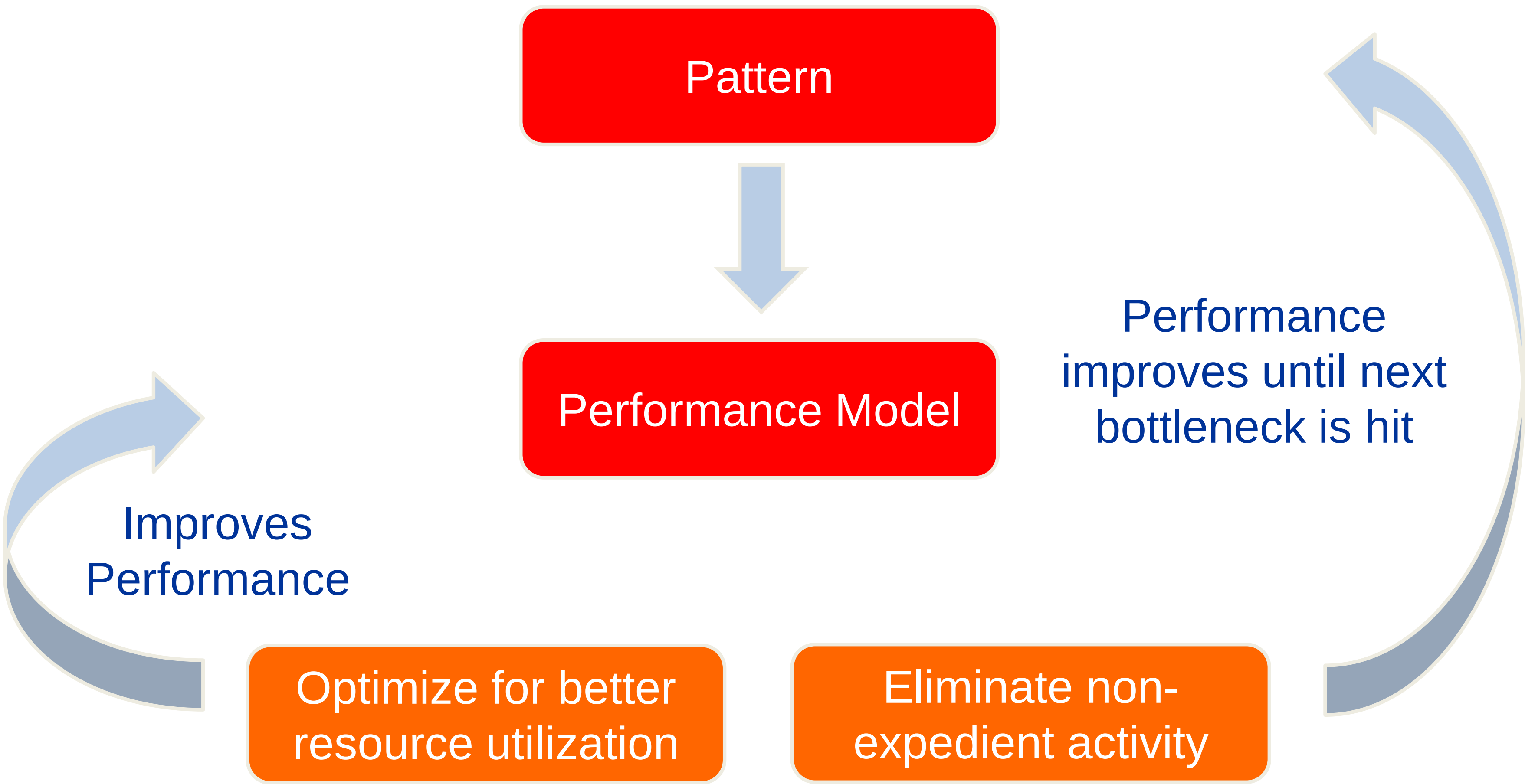
Relativistic quantum field theory

$$U(1)_Y \otimes SU(2)_L \otimes SU(3)_c$$

Consequences

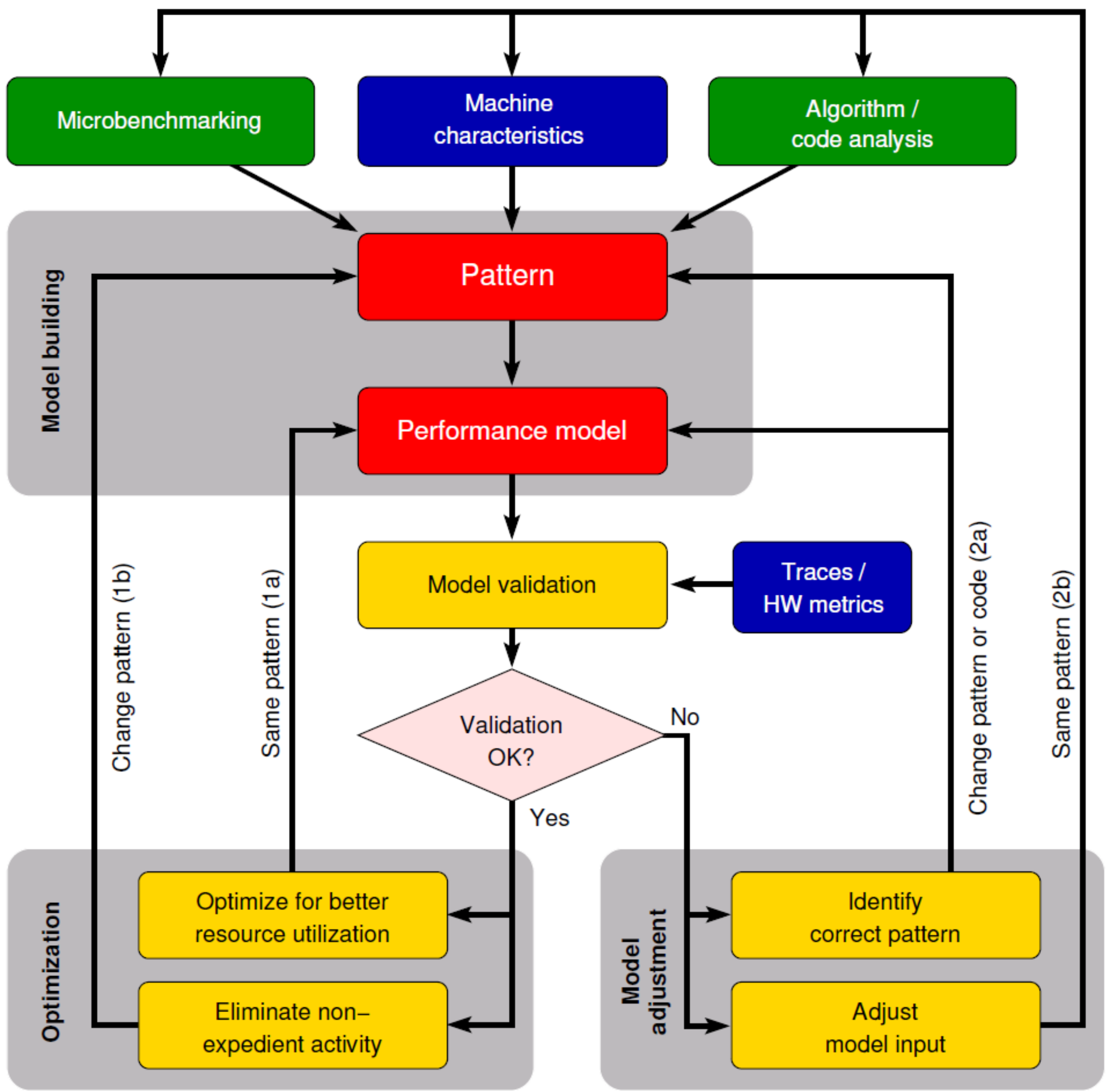
- If models fail, we learn more
- A simple model can get us very far before we need to refine

Performance Engineering Process: Optimization

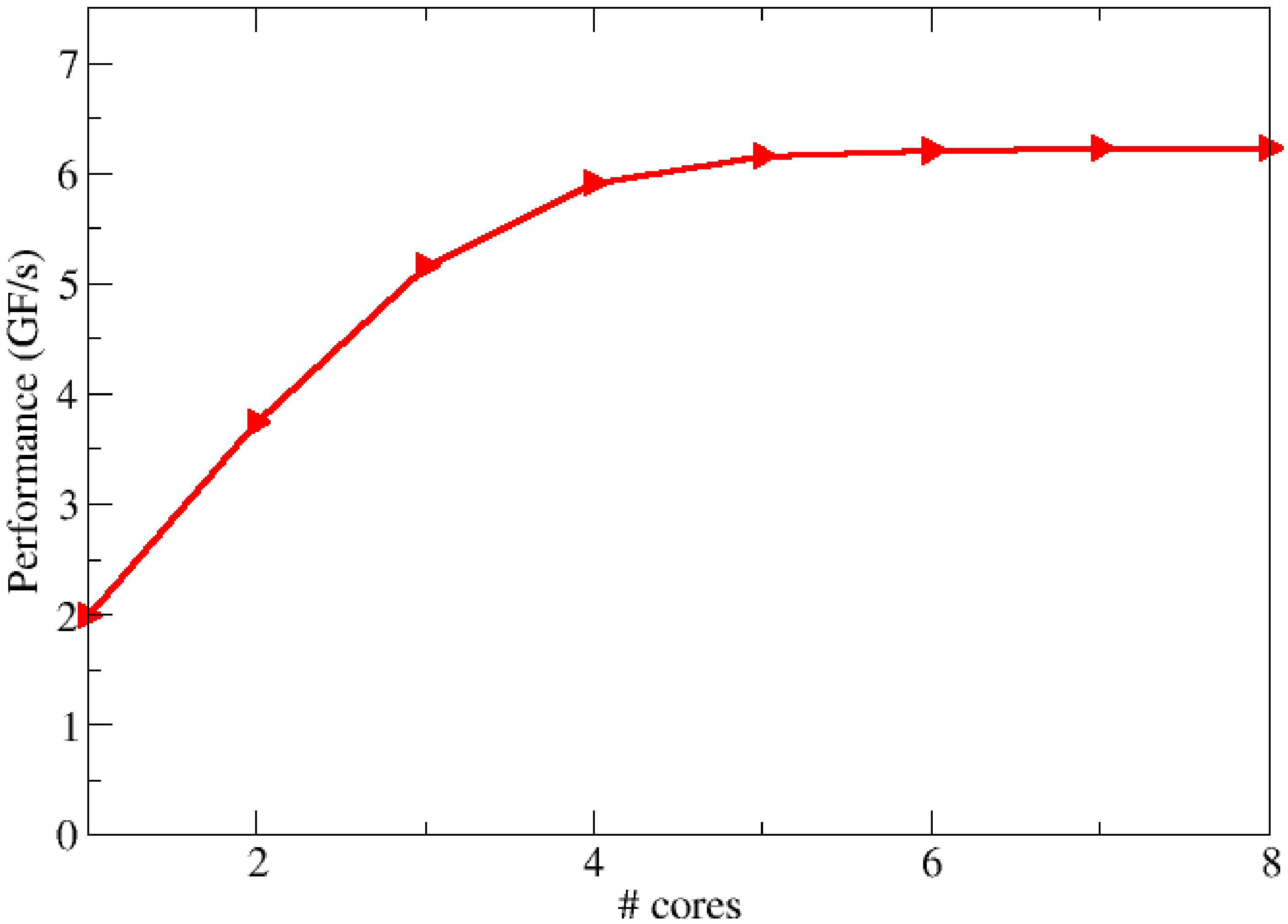


Step 3 **Optimization:** Improve utilization of offered resources.

The whole PE process at a glance



SpMVM Pattern: BW saturation



Roofline performance model for SELL-C-σ

Code balance (double precision FP, 4-byte index):

Matrix data & column index

RHS access

LHS update

$$B_{SELL}(\alpha,\beta,N_{nzr}) = \left(\boxed{\frac{1}{\beta}\left(\frac{8+4}{2}\right)} + \frac{\boxed{8\alpha} + \boxed{16/N_{nzr}}}{2}\right)\frac{\text{bytes}}{\text{flop}}$$

$$= \left(\frac{6}{\beta} + 4\alpha + \frac{8}{N_{nzr}}\right)\frac{\text{bytes}}{\text{flop}}$$

$$P(\alpha,\beta,N_{nzr},b) = \frac{b}{B_{SELL}(\alpha,\beta,N_{nzr})}$$

The α parameter

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
FACULTY OF ENGINEERING

Corner case scenarios:

1. $\alpha = 0$ → RHS in cache
2. $\alpha = \frac{1}{N_{nzc}}$ → Load RHS vector exactly once

   If $N_{nzc} \gg 1$, RHS traffic is insignificant: $\bar{P} = \frac{b\beta}{6\,\text{bytes/flop}}$

3. $\alpha \approx 1$ → Each RHS load goes to memory
4. $\alpha > 1$ → Houston, we've got a problem ☺

Determine α by measuring actual spMVM memory traffic (HPM)

SPPEXA

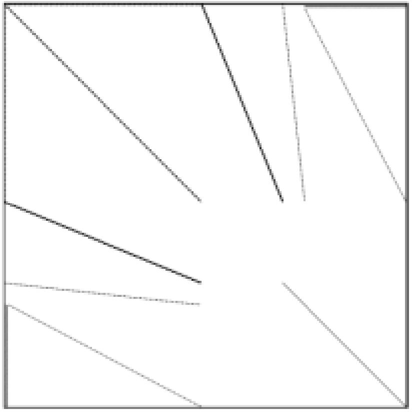
Determine RHS traffic

V_{meas} is the measured overall memory data traffic (using, e.g., likwid-perfctr)

Determine α:

$$\alpha = \frac{1}{4}\left(\frac{V_{meas}}{N_{nz} \cdot 2\ \text{bytes}} - 6 - \frac{8}{N_{nzr}}\right)$$

Example: kkt_power matrix on one Intel SNB socket



1. $N_{nz} = 14.6 \cdot 10^6, N_{nzr} = 7.1$
2. $V_{meas} \approx 258\ \text{MB}$
3. → $\alpha = 0.43$, $\alpha N_{nzr} = 3.1$
4. → RHS is loaded 3.1 times from memory
5. and:

$$\frac{B_{CRS}^{DP}(\alpha)}{B_{CRS}^{DP}(1/N_{nzc})} = 1.15$$

15% extra traffic
→ optimization potential!

Download our building block library and KPM application: http://tiny.cc/ghost

General, Hybrid, and Optimized Sparse Toolkit

- MPI + OpenMP + SIMD + CUDA
- Transparent data-parallel heterogeneous execution
- Task-parallelism (checkpointing, comm. hiding, etc.)
- Support for block vectors
  - Automatic code generation for common block vector sizes
  - Hand-implemented tall skinny dense matrix kernels
- Fused kernels (“augmented SpMV”)
- SELL-C-σ heterogeneous sparse matrix format

. . .

Further information

1. **About patterns in Performance Engineering**

   J. Treibig, G. Hager, and G. Wellein: Performance patterns and hardware metrics on modern multicore processors: Best practices for performance engineering. PROPER 2012, DOI: 10.1007/978-3-642-36949-0_50

2. **About Performance Modeling in general**

   ISC15 Workshop "Performance Modeling: Methods and Applications", July 16, 2015, Frankfurt

3. **About our holistic performance engineering approach**

   PRACE tutorials (July 6-7 @HLRS Stuttgart & December 10-11 @LRZ Garching)
   SWSC workshop (April 9/10 @U Leuven, Belgium)
   SC15 tutorial?

Thank you.