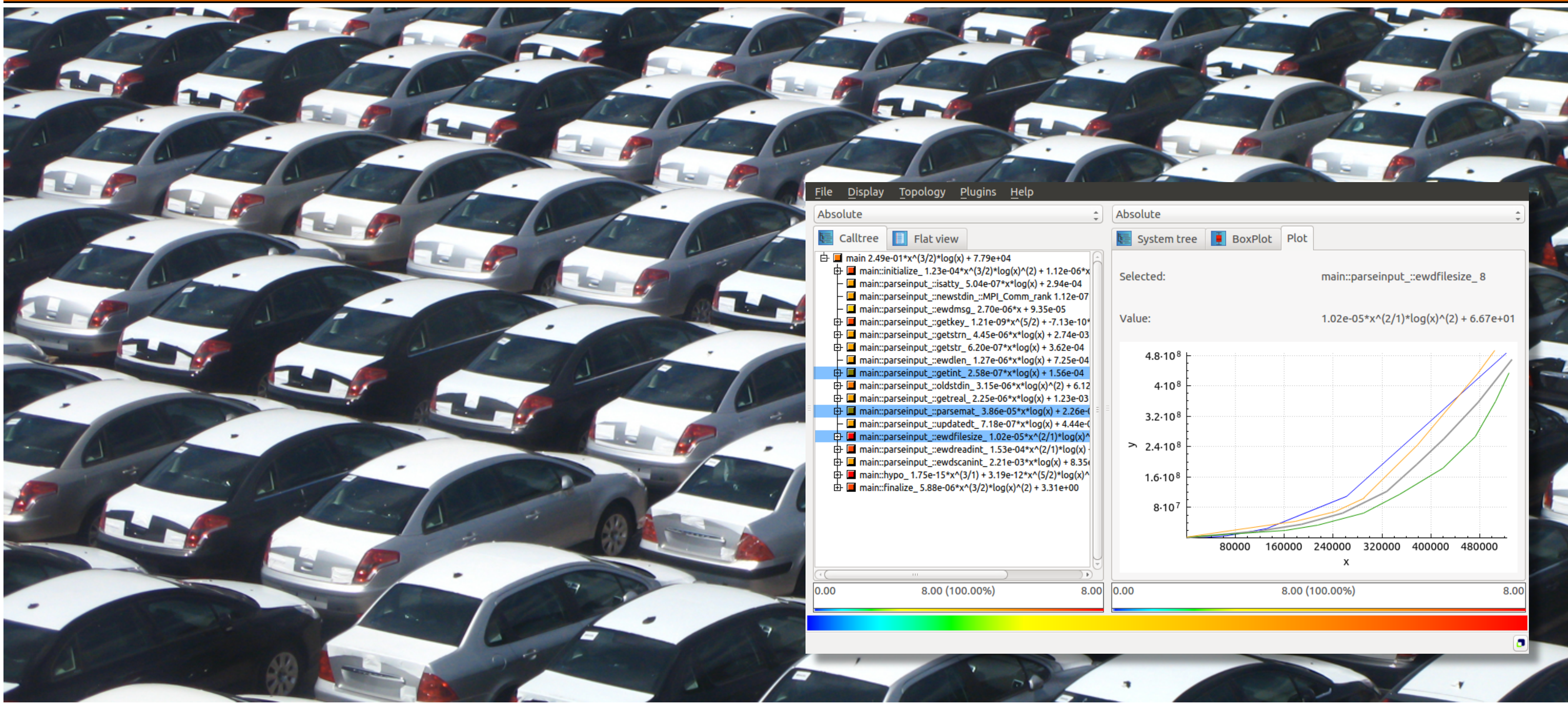
Is your software ready for exascale? – How the next generation of performance tools can give you the answer

TECHNISCHE UNIVERSITÄT DARMSTADT

Felix Wolf, TU Darmstadt

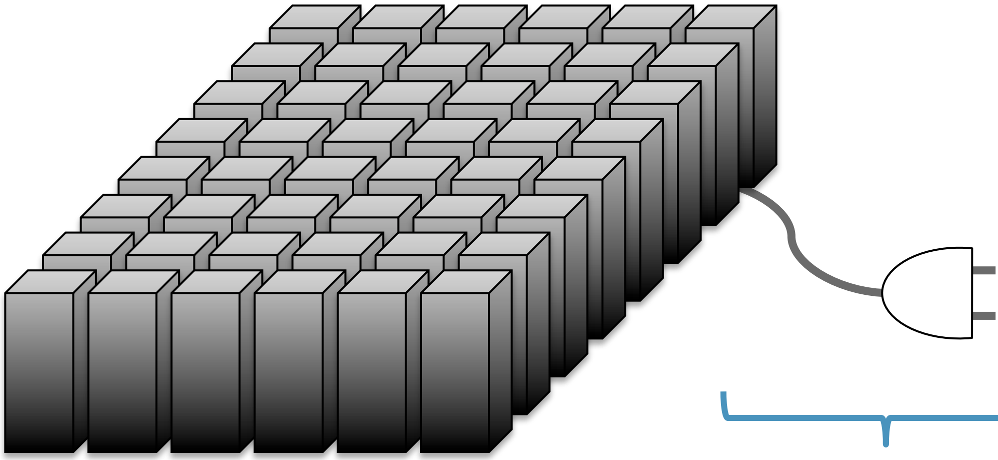


Acknowledgement

- Alexandru Calotoiu (TU Darmstadt)
- Torsten Höfler (ETH Zurich)
- Sergei Shudler (TU Darmstadt)
- Alexandre Strube (Jülich Supercomputing Centre)
- Andreas Vogel (GU Frankfurt)

- Marius Poke (RWTH Aachen)
- Paul Wiedeking (RWTH Aachen)

Power envelope of hardware



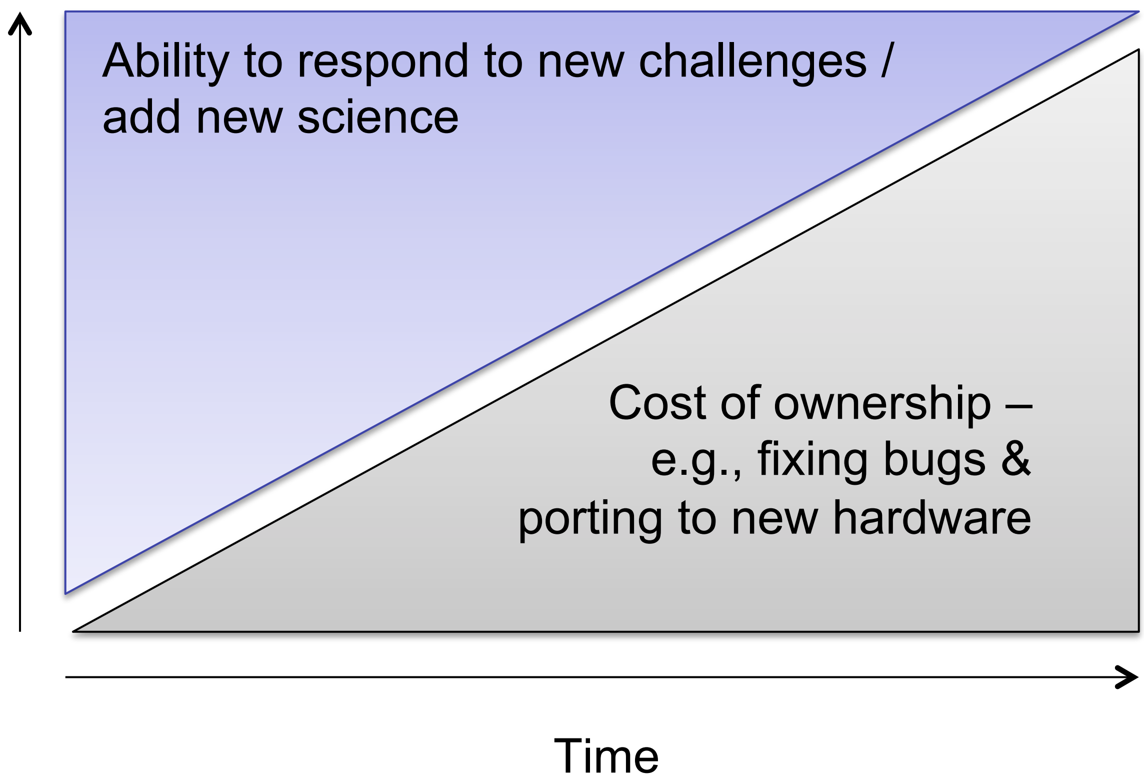
Exascale system

≤ 20 MW << 142 MW

Green 500 # 1	
GFLOPS/W	7.03
Site	RIKEN
Computer	Shoubu - ExaScaler
Total Power (kW)	50.32

Manpower envelope of software

Available
manpower



Electrical power vs. manpower

Energy costs	
Power	20 MW
Price per kWh	0.1 €
Hours per year	5,000 h
Energy costs per year	10 M€

200 FTEs

(50k€ per FTE)

To amortize the investment, one FTE needs to tune the workload by **0.5%**

Electrical power vs. manpower

Energy costs	
Power (DARPA estimate)	60 MW
Price per kWh	0.1 €
Hours per year	5,000 h
Energy costs per year	30 M€

600 FTEs

To amortize the investment, one FTE needs to tune the workload by **0.15%**

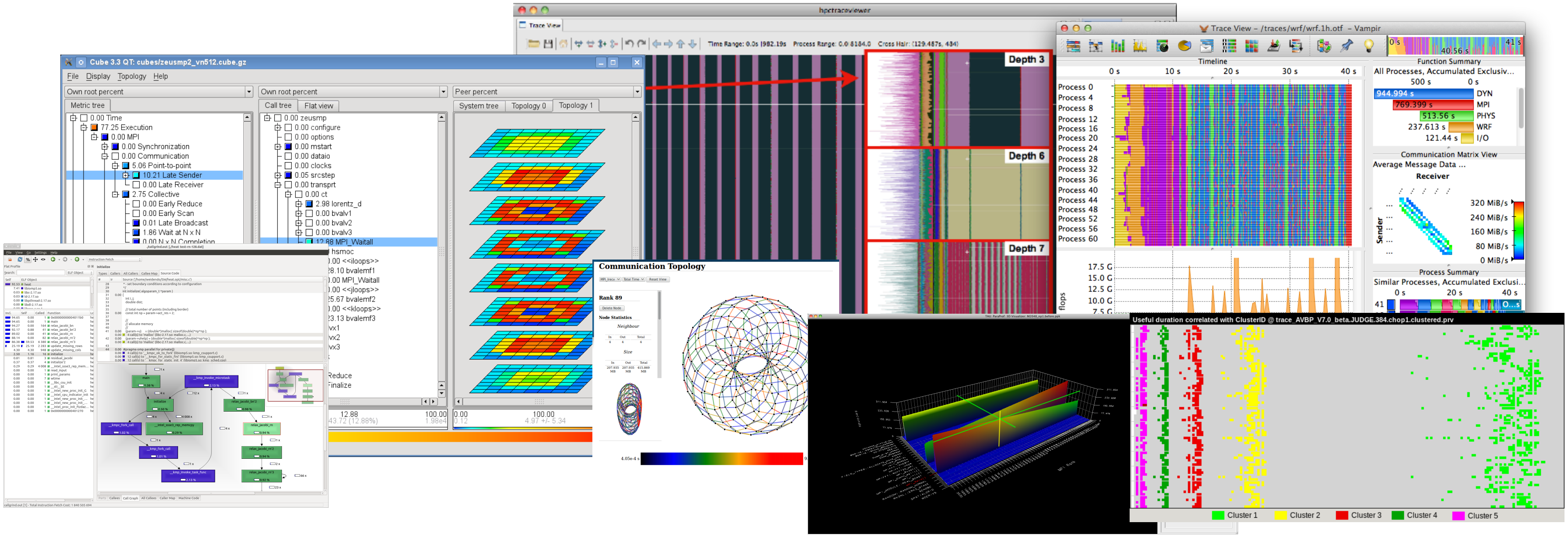
- Potential in trading hardware for brainware*
- Productivity of staff can be further increased through performance tools
- Early resolution of performance issues maximizes benefit

* C. Bischof et al.: Brainware for green HPC, Computer Science-Research and Development, Springer

TECHNISCHE UNIVERSITÄT DARMSTADT

Traditional performance tools

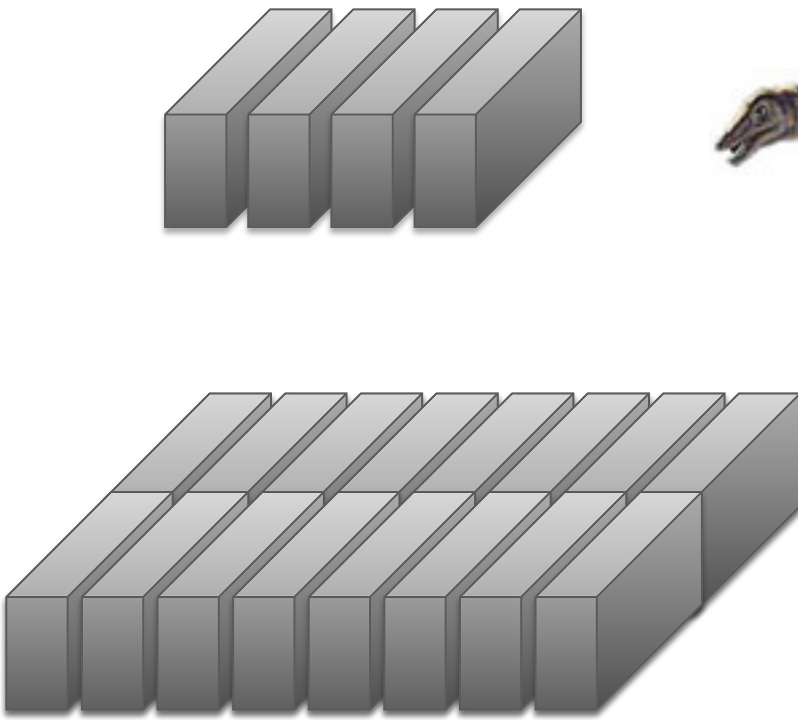
Provide insight into measured performance behavior



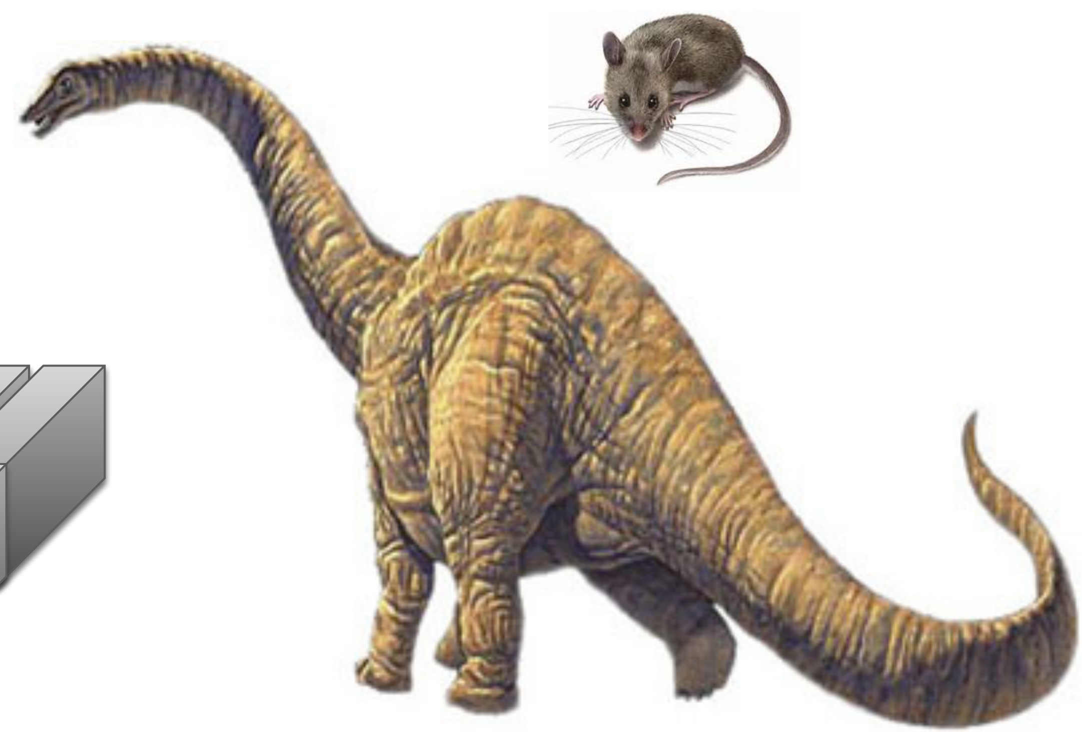
Scale of insight = scale of experiment

Latent scalability bugs

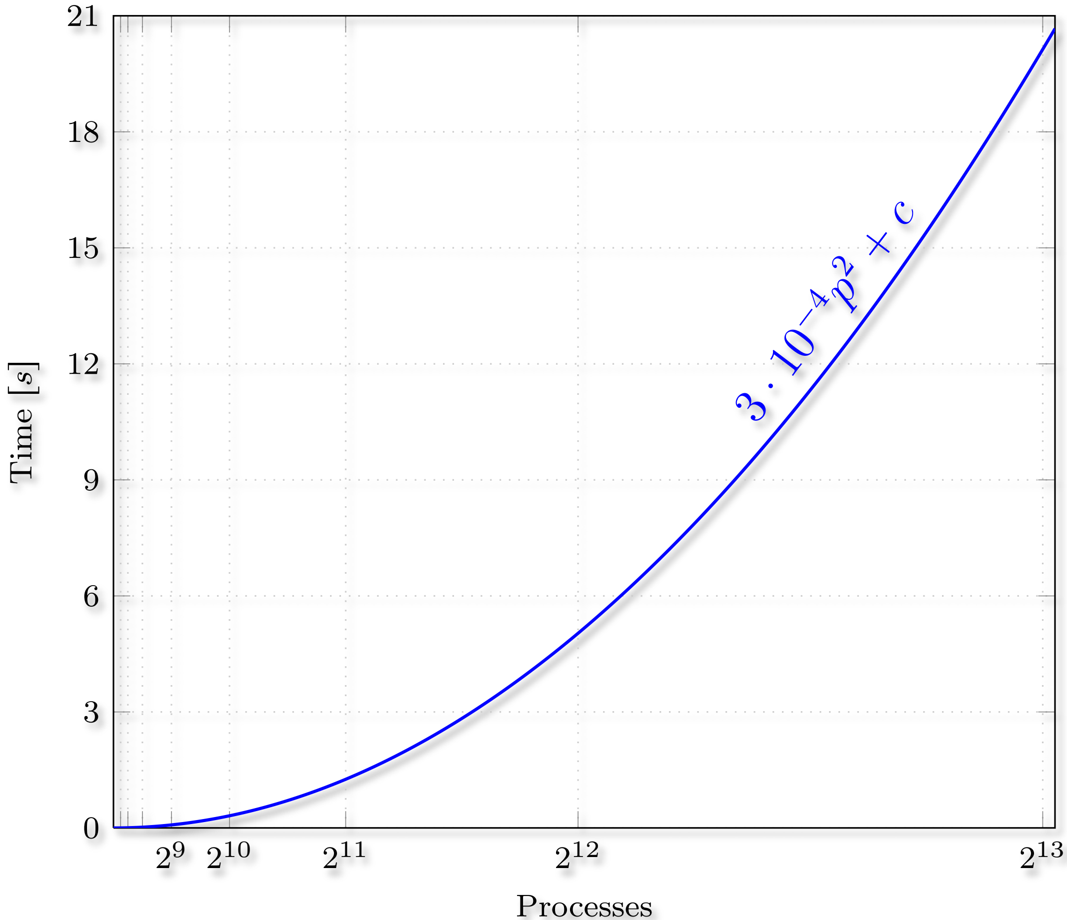
System size



Execution time



Scalability model



Analytical scalability modeling

TECHNISCHE UNIVERSITÄT DARMSTADT

Identify kernels

- Parts of the program that dominate its performance at larger scales
- Identified via small-scale tests and intuition

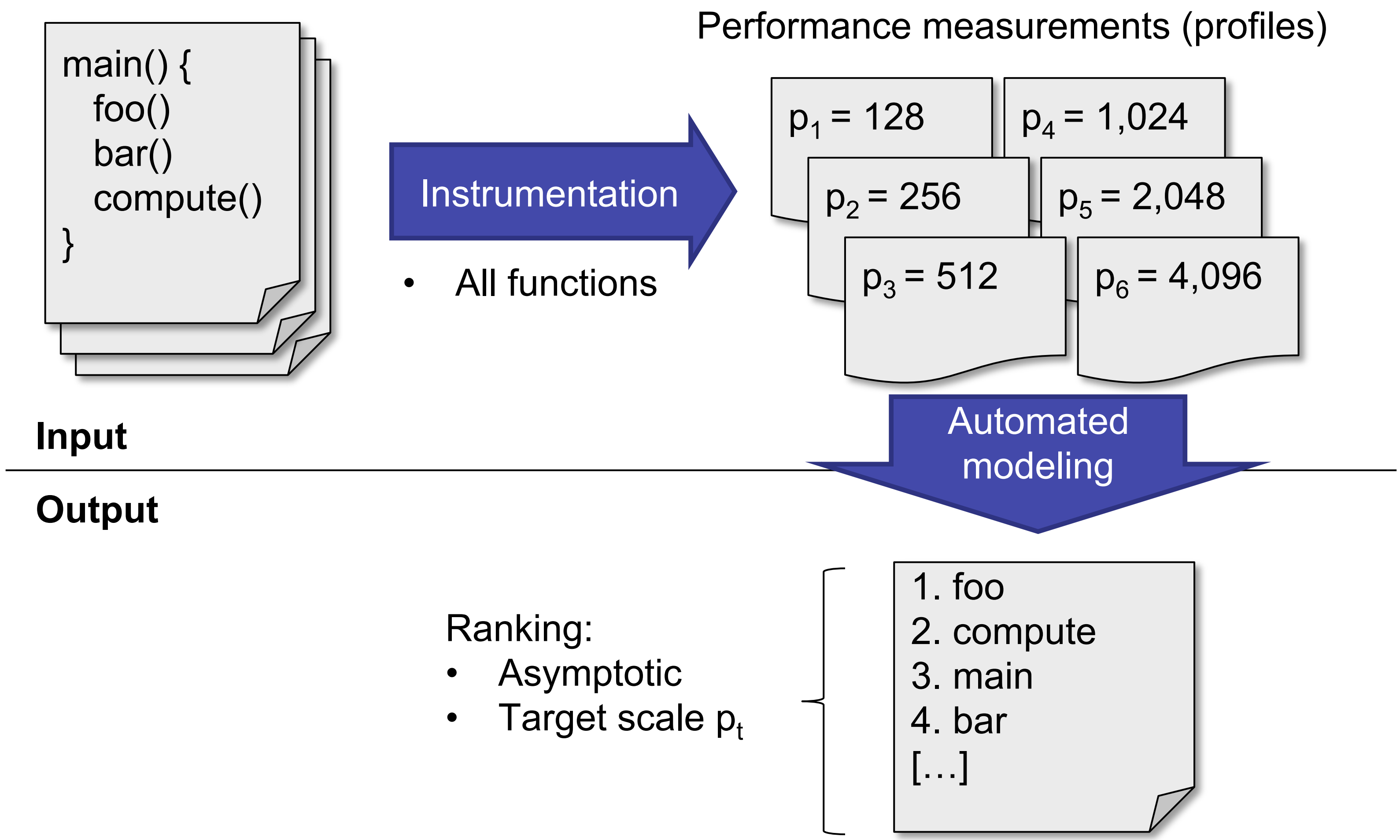
Create models

- Laborious process
- Still confined to a small community of skilled experts

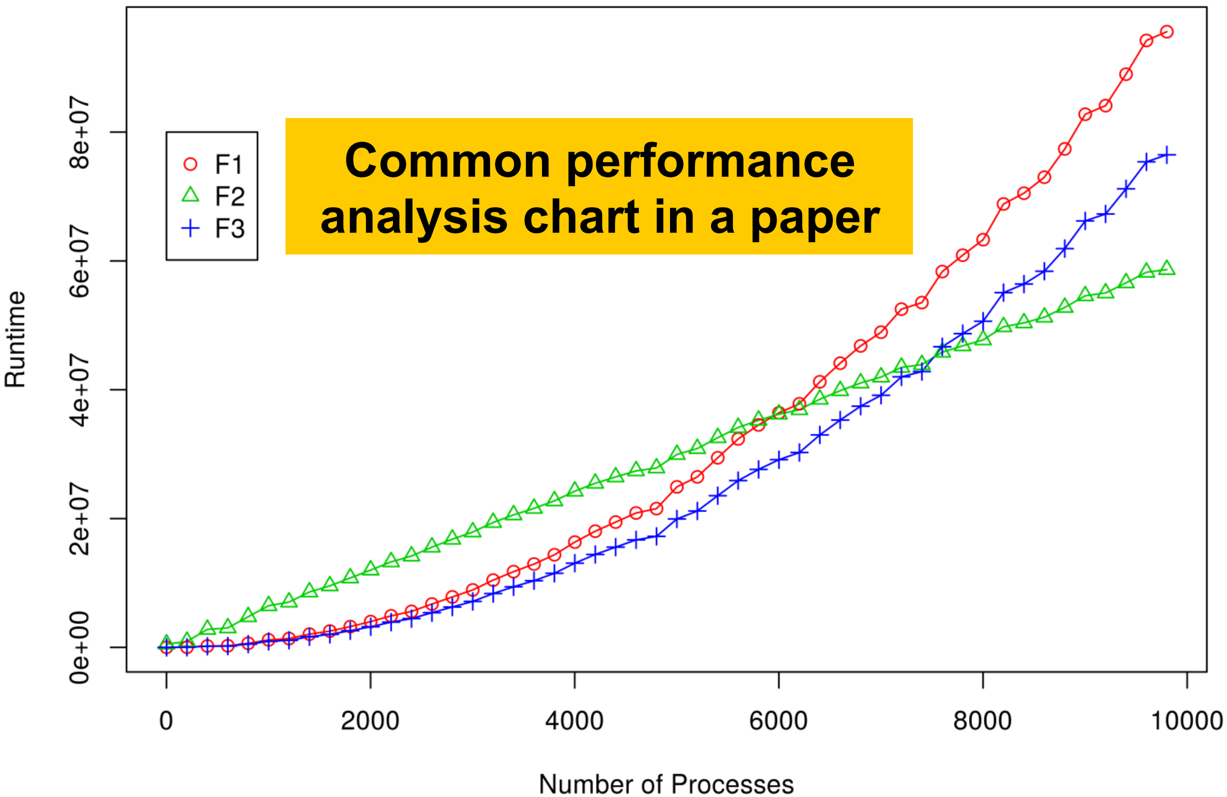
Disadvantages

- Time consuming
- Danger of overlooking unscalable code

Automated empirical modeling (2)



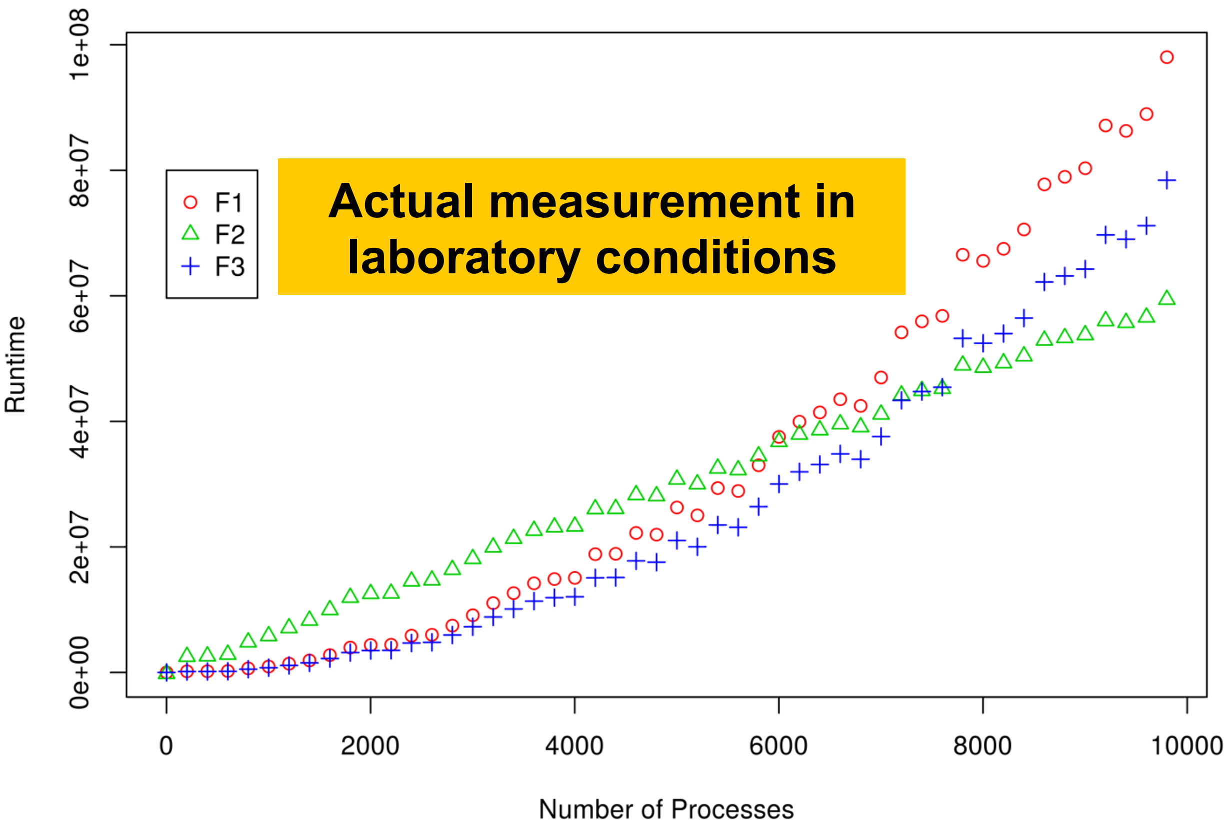
Primary focus on scaling trend



Our ranking

1. F_1
2. F_3
3. F_2

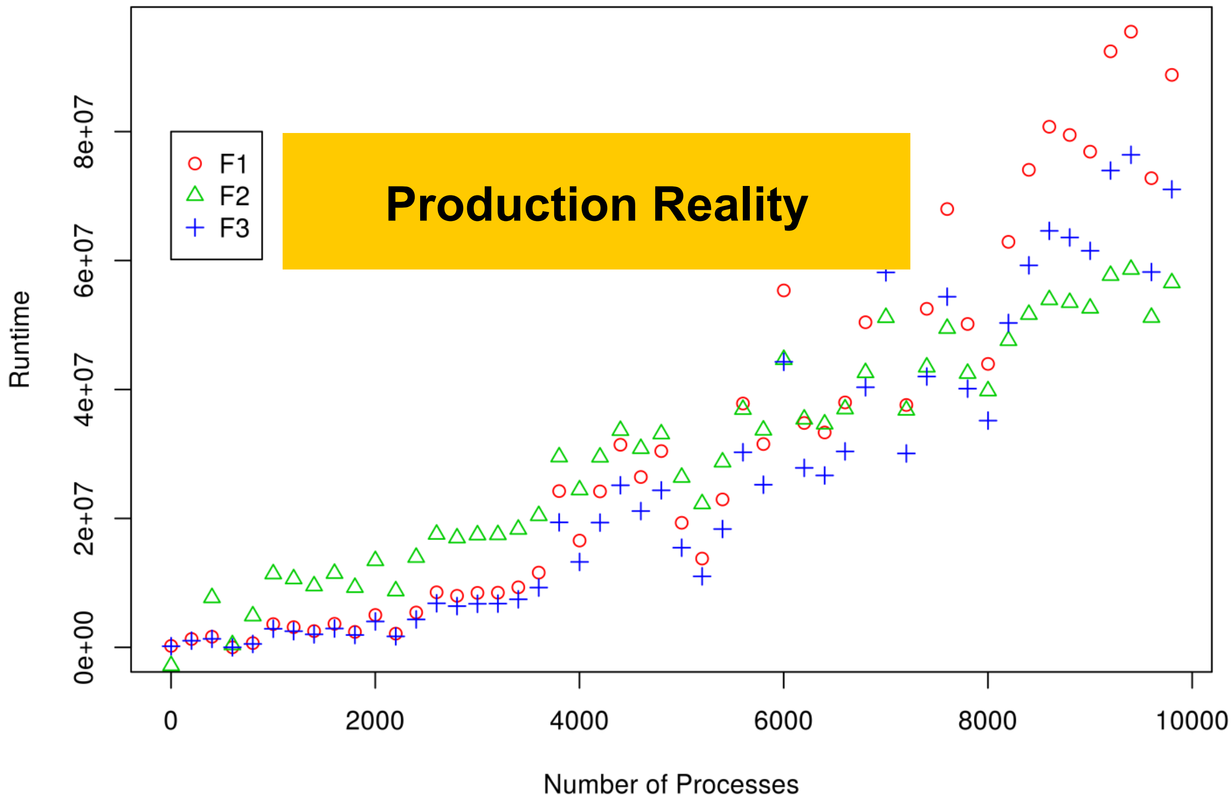
Primary focus on scaling trend



Our ranking

1. F_1
2. F_3
3. F_2

Primary focus on scaling trend



Our ranking

1. F_1
2. F_3
3. F_2

Model building blocks

Computation	Communication
LU $t(p) \sim c$	LU $t(p) \sim c$
FFT $t(p) \sim \log_2(p)$	FFT $t(p) \sim c$
Naïve N-body $t(p) \sim p$	Naïve N-body $t(p) \sim p$
…	…
Samplesort $t(p) \sim p^2 \log_2^2(p)$	Samplesort $t(p) \sim p^2$

Performance model normal form

$$f(p) = \sum_{k=1}^{n} c_k \cdot p^{i_k} \cdot \log_2^{j_k}(p)$$

$n \in \mathbb{N}$
$i_k \in I$
$j_k \in J$
$I, J \subset \mathbb{Q}$

$n = 1$
$I = \{0,1,2\}$
$J = \{0,1\}$

c_1 $\quad$ $c_1 \cdot \log(p)$
$c_1 \cdot p$ $\quad$ $c_1 \cdot p \cdot \log(p)$
$c_1 \cdot p^2$ $\quad$ $c_1 \cdot p^2 \cdot \log(p)$

Performance model normal form

TECHNISCHE UNIVERSITÄT DARMSTADT

$$f(p)=\sum_{k=1}^{n} c_k \cdot p^{i_k} \cdot \mathrm{lo}\ldots$$

$n \in \mathbb{N}$

$n = 2$

$I = \{0,1,2\}$

$J = \{0,1\}$

$c_1 + c_2 \cdot p$

$c_1 + c_2 \cdot p^2$

$c_1 + c_2 \cdot \log(p)$

$c_1 + c_2 \cdot p \cdot \log(p)$

$c_1 + c_2 \cdot p^2 \cdot \log(p)$

$c_1 \cdot \log(p) + c_2 \cdot p$

$c_1 \cdot \log(p) + c_2 \cdot p \cdot \log(p)$

$c_1 \cdot \log(p) + c_2 \cdot p^2$

$c_1 \cdot \log(p) + c_2 \cdot p^2 \cdot \log(p)$

$c_1 \cdot p + c_2 \cdot p \cdot \log(p)$

$c_1 \cdot p + c_2 \cdot p^2$

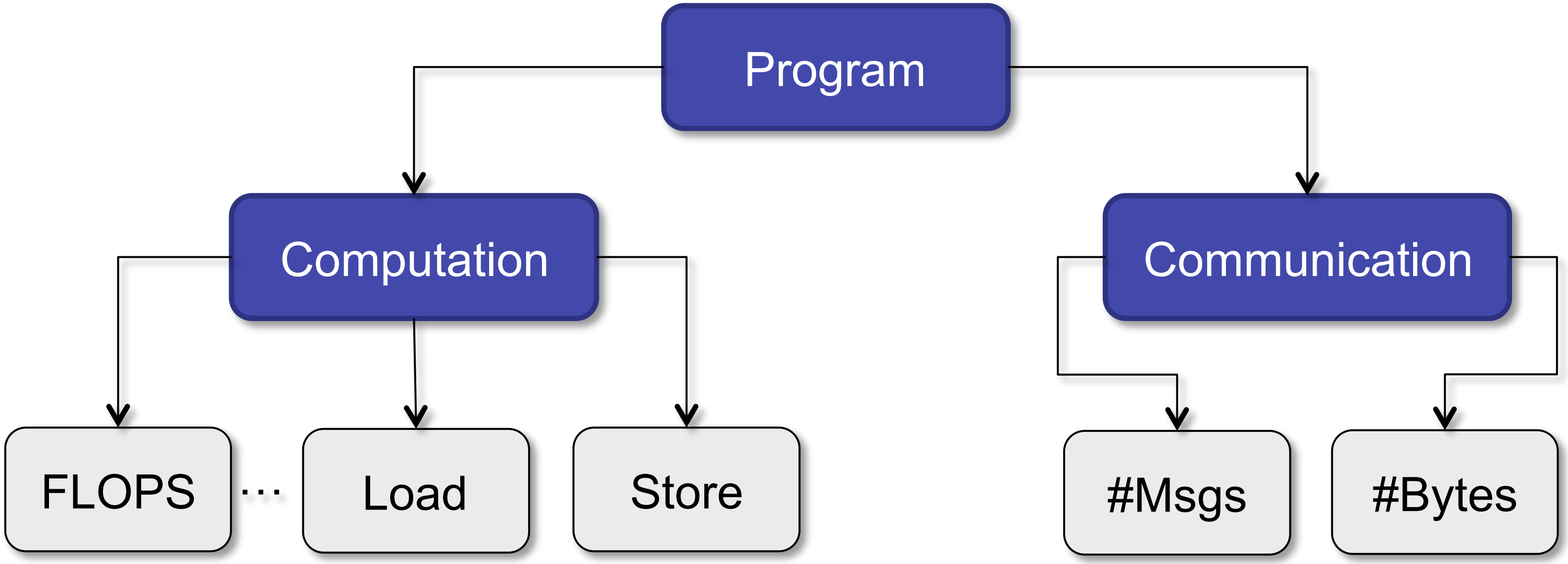
$c_1 \cdot p + c_2 \cdot p^2 \cdot \log(p)$

$c_1 \cdot p \cdot \log(p) + c_2 \cdot p^2$

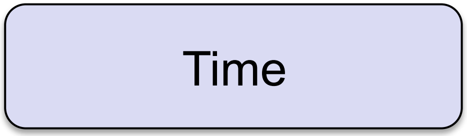
$c_1 \cdot p \cdot \log(p) + c_2 \cdot p^2 \cdot \log(p)$

$c_1 \cdot p^2 + c_2 \cdot p^2 \cdot \log(p)$

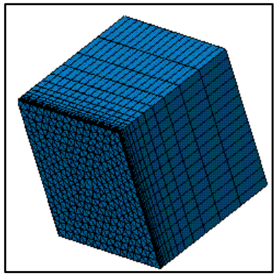
Modeling operations vs. time

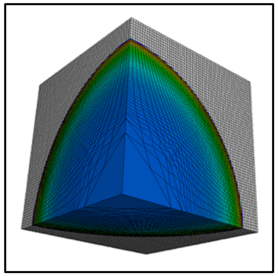


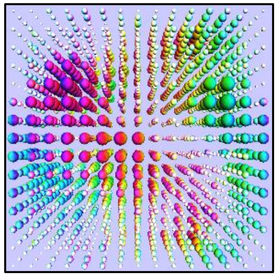
Disagreement may be indicative of wait states

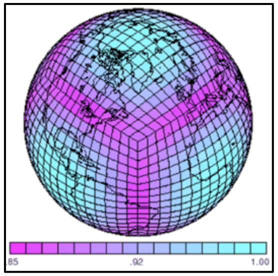


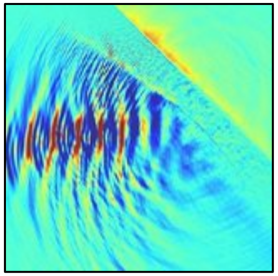
Case studies

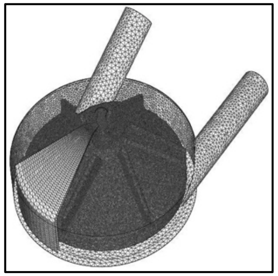

Sweep3d


Lulesh

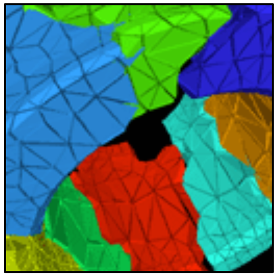

Milc

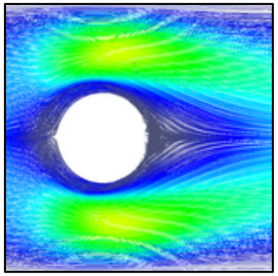

HOMME

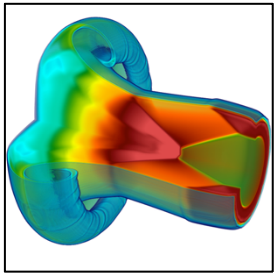

JUSPIC


XNS


NEST


UG4


MP2C


BLAST

MPI

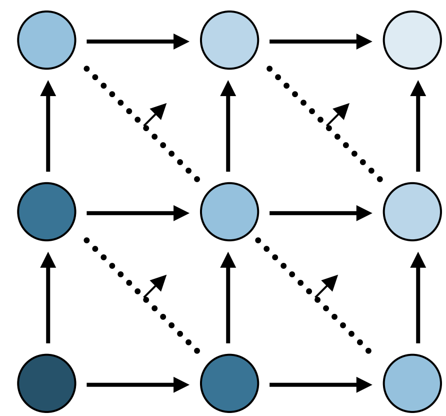
OpenMP

Sweep3D - Neutron transport simulation

- LogGP model for communication developed by Hoisie et al.

$$t^{comm} = [2(p_x + p_y - 2) + 4(n_{sweep} - 1)] \cdot t_{msg}$$

$$t^{comm} = c \cdot \sqrt{p}$$



Kernel [2 of 40]	Model [s] t = f(p)	Predictive error [%] p_t=262k
sweep → MPI_Recv	$4.03\sqrt{p}$	5.10
sweep	582.19	.01

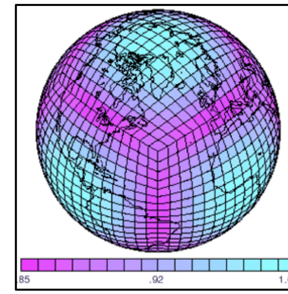
#bytes ≈ const.
#msg ≈ const.

$p_i \leq 8\text{k}$

HOMME – Climate

Core of the Community Atmospheric Model (CAM)

- Spectral element dynamical core on a cubed sphere grid



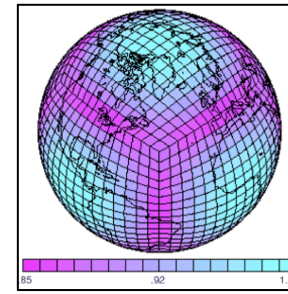
Kernel [3 of 194]	Model [s] t = f(p)	Predictive error [%] p_t = 130k
box_rearrange → MPI_Reduce	$0.026 + 2.53 \cdot 10^{-6} p \cdot \sqrt{p} + 1.24 \cdot 10^{-12} p^3$	57.02
vlaplace_sphere_vk	49.53	99.32
compute_and_apply_rhs	48.68	1.65

$$p_i \le 15\text{k}$$

HOMME – Climate

Core of the Community Atmospheric Model (CAM)

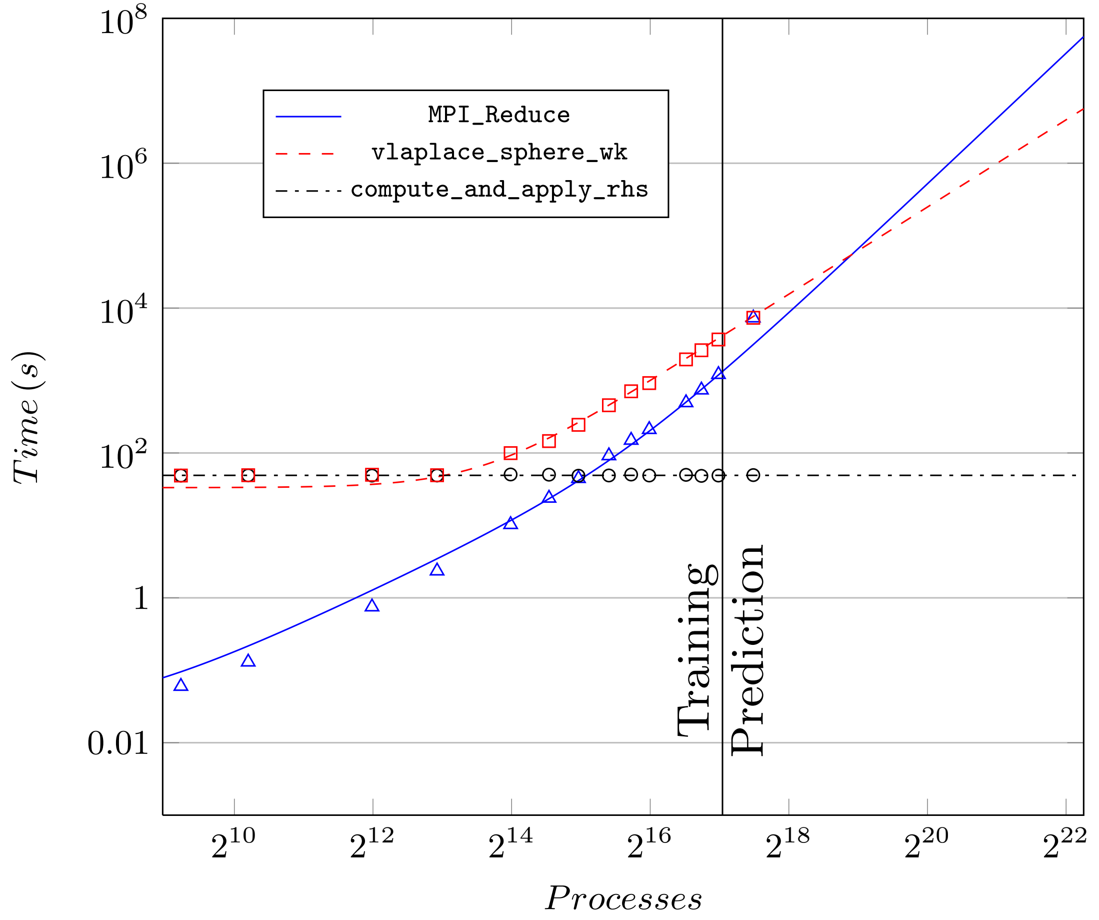
- Spectral element dynamical core on a cubed sphere grid



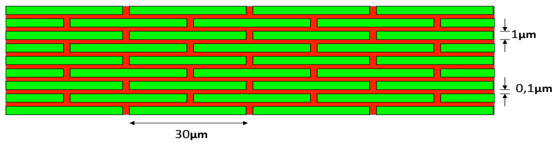
Kernel [3 of 194]	Model [s] $t = f(p)$	Predictive error [%] p_t = 130k
box_rearrange → MPI_Reduce	$3.63 \cdot 10^{-6} p \cdot \sqrt{p} + 7.21 \cdot 10^{-13} p^3$	30.34
vlaplace_sphere_vk	$24.44 + 2.26 \cdot 10^{-7} p^2$	4.28
compute_and_apply_rhs	49.09	0.83

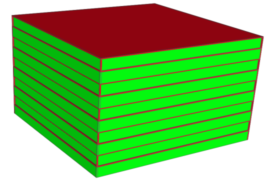
$\boxed{p_i \leq 43\text{k}}$

HOMME – Climate (2)



UG4





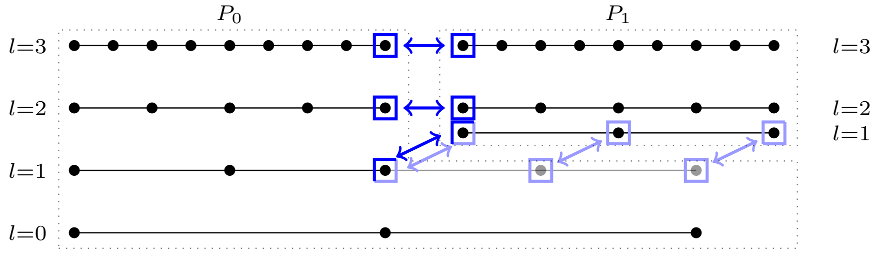
TECHNISCHE
UNIVERSITÄT
DARMSTADT

- Numerical framework for grid-based solution of partial differential equations (~500,000 lines of C++ code, 2,000 kernels)
  - Application: drug diffusion through the human skin
- In general, all kernels scale well
  - Multigrid solver kernel (MGM) scales logarithmically
  - Number of iterations needed by the unpreconditioned conjugate gradient (CG) method depends on the mesh size
    - Increases by factor of two with each refinement
    - Will therefore suffer from iteration count increase in weak scaling

Kernel	Model (time [s])
CG	$0.227 + 0.31 * p^{0.5}$
MGM	$0.219 + 0.0006 * \log^2(p)$

Issue with MPI communicator group creation

- Create MPI communicator groups for each level of multigrid hierarchy
- Exclude processes that do not own a grid part on that level



- *Before*: Membership info communicated using MPI_Allreduce with array of length p - non-scalable *p* * *O*(*MPI_Allreduce*) complexity
- *Now*: MPI_Allreduce replaced by MPI_Comm_split - enhanced algorithms of which are known to have $O(\log^2 p)$ complexity

(C. Siebert, F. Wolf: Parallel sorting with minimal data. Recent Advances in the Message Passing Interface, 2011)

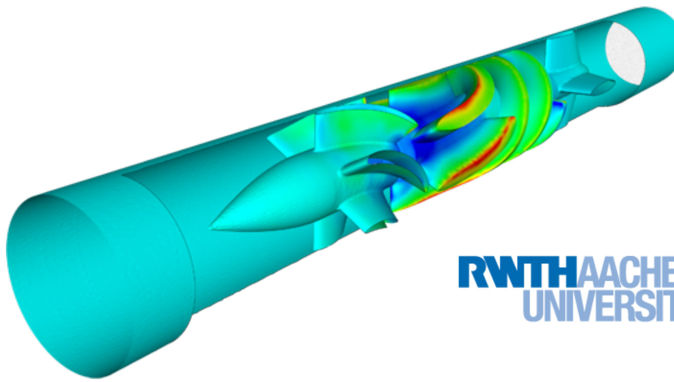
Weak vs. strong scaling

- Wall-clock time not necessarily monotonically increasing under strong scaling
  - Harder to capture model automatically
  - Different invariants require different reductions across processes

	Weak scaling	Strong scaling
Invariant	Problem size per process	Overall problem size
Model target	Wall-clock time	Accumulated time
Reduction	Maximum / average	Sum

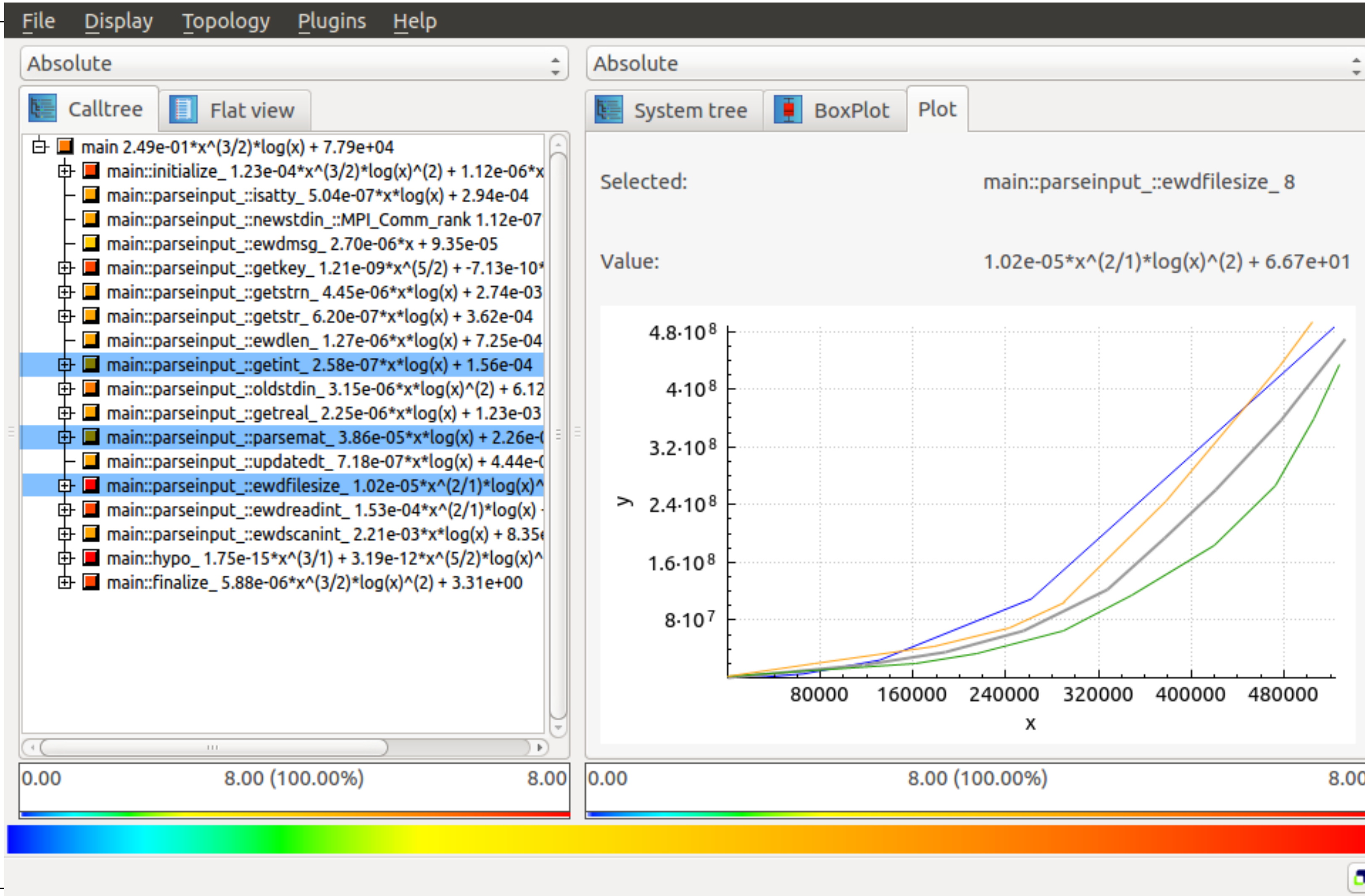
XNS

- Finite element flow simulation
- Strong scaling analysis using accumulated time across processes as metric

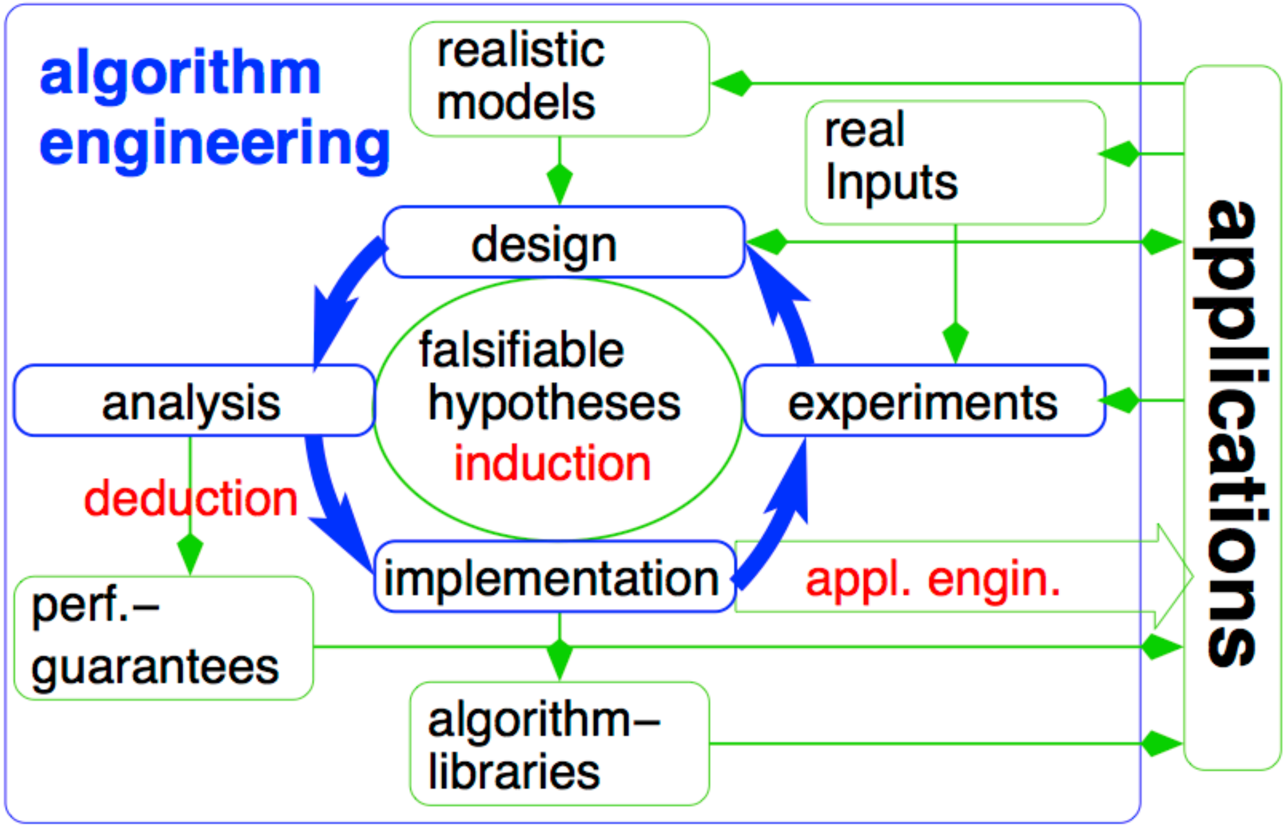


Kernel	Runtime[%] p=128	Runtime[%] p=4096	Model [s] t = f(p)
ewdgennprm->MPI_Recv	0.46	51.46	$0.029 \cdot p^2$
ewddot	44.78	5.04	$p \cdot \log(p)$

#bytes = ~p
#msg = ~p


File Display Topology Plugins Help
Absolute
Absolute
Calltree
Flat view
System tree
BoxPlot
Plot
main 2.49e-01*x^(3/2)*log(x) + 7.79e+04
main::initialize_ 1.23e-04*x^(3/2)*log(x)^(2) + 1.12e-06*x
main::parseinput_::isatty_ 5.04e-07*x*log(x) + 2.94e-04
main::parseinput_::newstdin_::MPI_Comm_rank 1.12e-07
main::parseinput_::ewdmsg_ 2.70e-06*x + 9.35e-05
main::parseinput_::getkey_ 1.21e-09*x^(5/2) + -7.13e-10*
main::parseinput_::getstrn_ 4.45e-06*x*log(x) + 2.74e-03
main::parseinput_::getstr_ 6.20e-07*x*log(x) + 3.62e-04
main::parseinput_::ewdlen_ 1.27e-06*x*log(x) + 7.25e-04
main::parseinput_::getint_ 2.58e-07*x*log(x) + 1.56e-04
main::parseinput_::oldstdin_ 3.15e-06*x*log(x)^(2) + 6.12
main::parseinput_::getreal_ 2.25e-06*x*log(x) + 1.23e-03
main::parseinput_::parsemat_ 3.86e-05*x*log(x) + 2.26e-
main::parseinput_::updatedt_ 7.18e-07*x*log(x) + 4.44e-
main::parseinput_::ewdfilesize_ 1.02e-05*x^(2/1)*log(x)^
main::parseinput_::ewdreadint_ 1.53e-04*x^(2/1)*log(x)
main::parseinput_::ewdscanint_ 2.21e-03*x*log(x) + 8.35
main::hypo_ 1.75e-15*x^(3/1) + 3.19e-12*x^(5/2)*log(x)^
main::finalize_ 5.88e-06*x^(3/2)*log(x)^(2) + 3.31e+00
Selected: main::parseinput_::ewdfilesize_ 8
Value: 1.02e-05*x^(2/1)*log(x)^(2) + 6.67e+01
4.8·10^8
4·10^8
3.2·10^8
2.4·10^8
1.6·10^8
8·10^7
y
80000 160000 240000 320000 400000 480000
x
0.00 8.00 (100.00%) 8.00
0.00 8.00 (100.00%) 8.00

Algorithm engineering



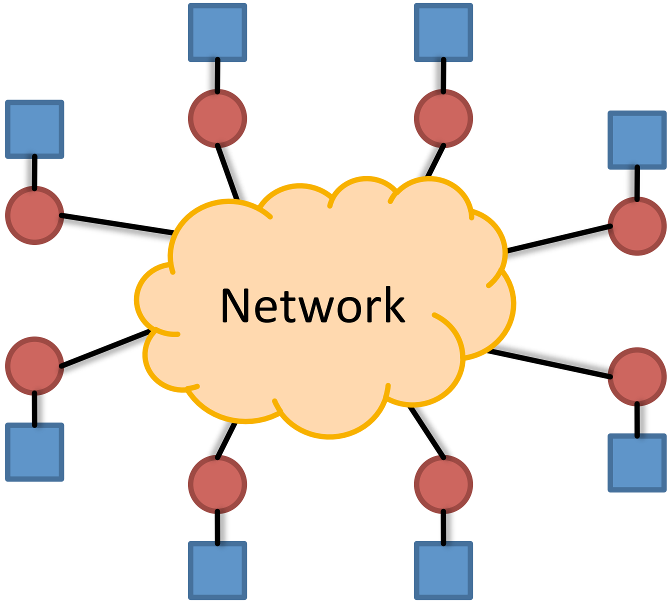
Courtesy of Peter Sanders, KIT

How to validate scalability in practice?

Program	Small text book example	Real application
Expectation	Verifiable analytical expression	Asymptotic complexity
	$\#FLOPS = n^2(2n - 1)$	$\#FLOPS = O(n^{2.8074})$

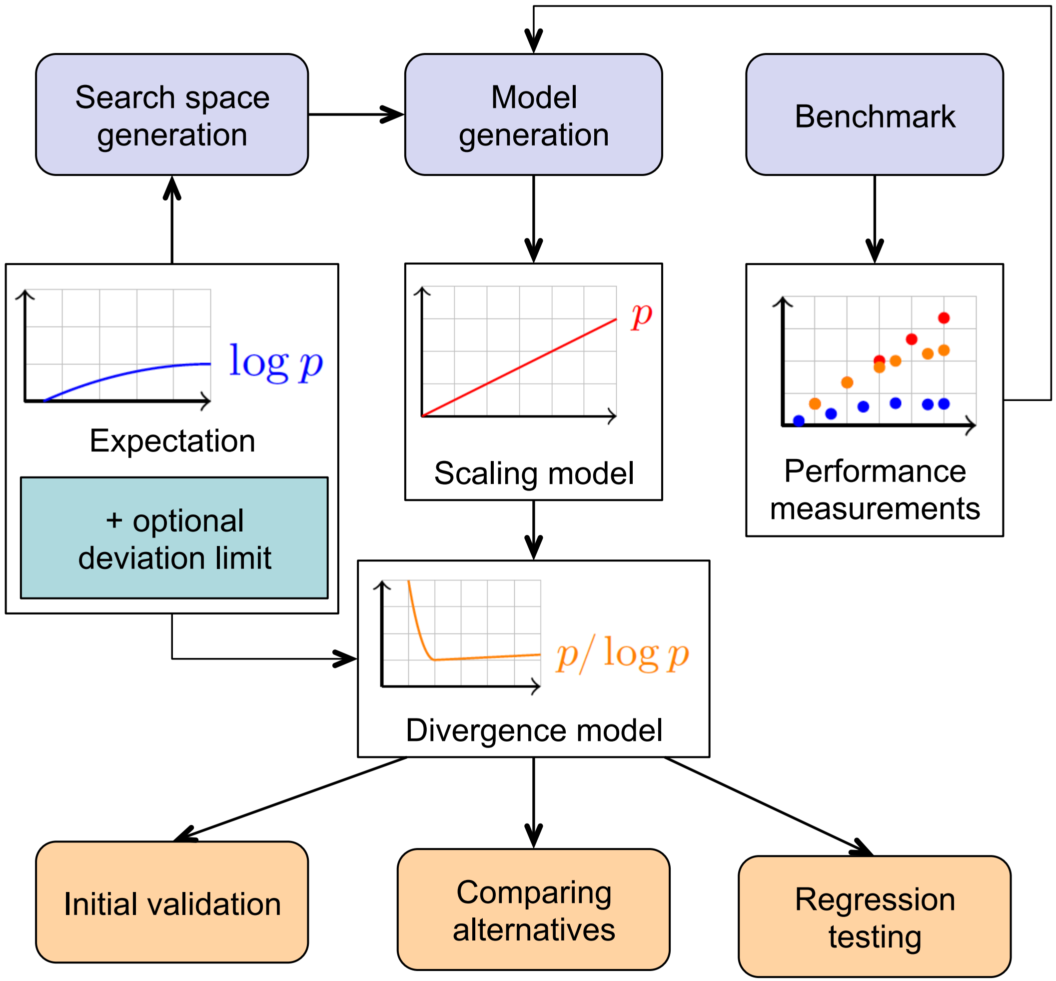
HPC libraries

- Focus on algorithms rather than applications
- Theoretical expectations more common
- Reuse factor makes scalability even more important



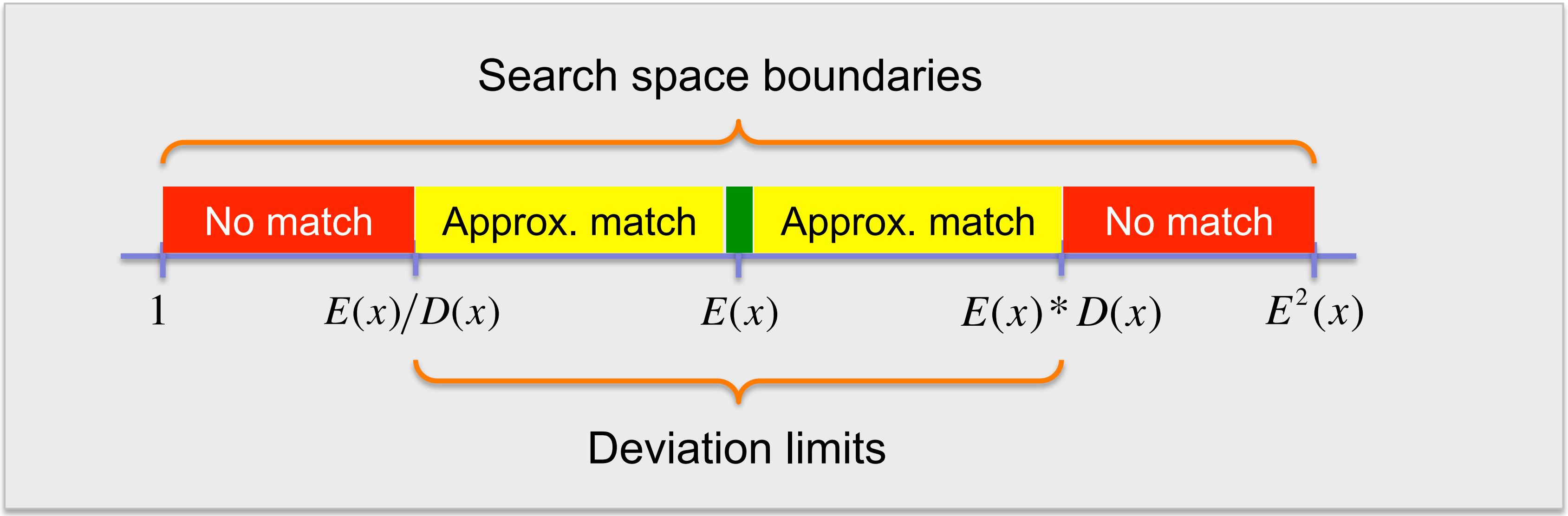
Example:
MPI communication library

Scalability evaluation framework



Customized search space

- Constructed around expectation
- Supports wider range of model functions than original PMNF



Test systems

Platform	Topology	Nodes	CPU	Cores*	Memory*	MPI
Juqueen (BG/Q)	5D torus	28,672	PPC A2	16	16 GB	PAMI
Juropa (Intel)	Fat tree w/ IB	3,288	X5570	8	24 GB	ParTec
Piz Daint (Cray-XC30)	Dragonfly	5,727	E5-2670	8	32 GB	Cray

* Per node

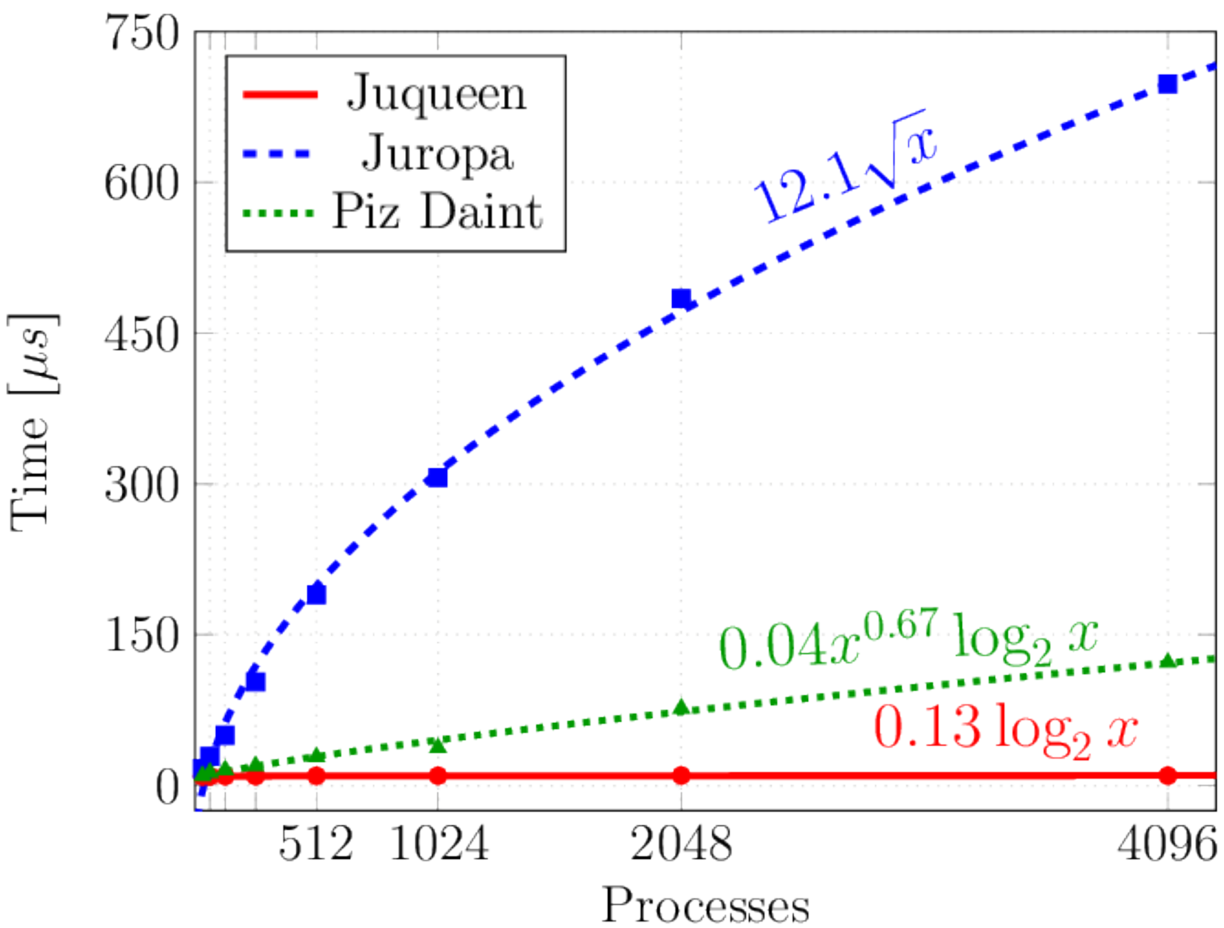

MPI

TECHNISCHE UNIVERSITÄT DARMSTADT

Platform	Juqueen	Juropa	Piz Daint
Allreduce [s]		**Expectation: *O* (log *p*)**	
Model	$O(\log p)$	$O(p^{0.5})$	$O(p^{0.67} \log p)$
R^2	0.87	0.99	0.99
Divergence	0	$O(p^{0.5}/\log p)$	$O(p^{0.67})$
Match	✓	~	✗!
Comm_dup [B]		**Expectation: *O* (1)**	
Model	2.2e5	256	$3770 + 18p$
R^2	1	1	0.99
Divergence	$O(1)$	$O(1)$	$O(p)$
Match	✓	✓	✗

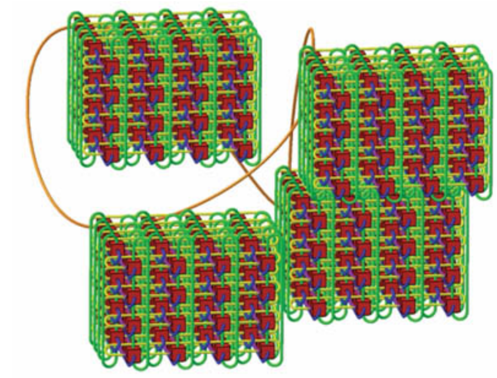
Platform	Daint
Barrier [s]	**O (log *p*)**
Model	og *p*)
R^2	
Divergence	)
Match	
Bcast [s]	**on: *O* (*p*)**
Model	)
R^2	)
Divergence	)
Match	
Reduce [s]	**on: *O* (*p*)**
Model	)
R^2	)
Divergence	)
Match	

Allreduce results



Divergence on Piz Daint is $O(p^{0.67})$, the highest of all three

Potential reasons for discrepancies



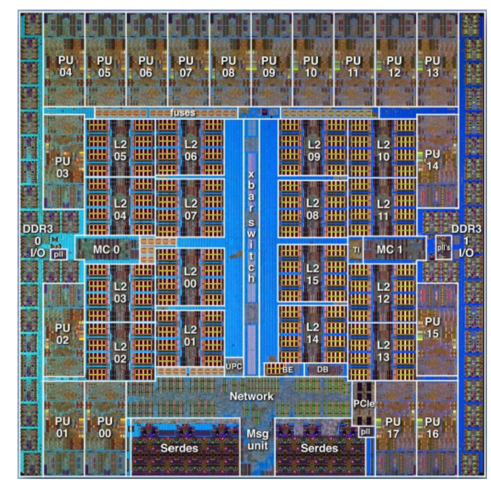
1. Expectations based on simplified network models (in reality: IB, N-D torus)

2. Nodes allocation, node's neighborhood

   A. Bhatele, K. Mohror, S. H. Langer, and K. E. Isaacs.
   There Goes the Neighborhood: Performance Degradation Due to Nearby Jobs, SC' 13

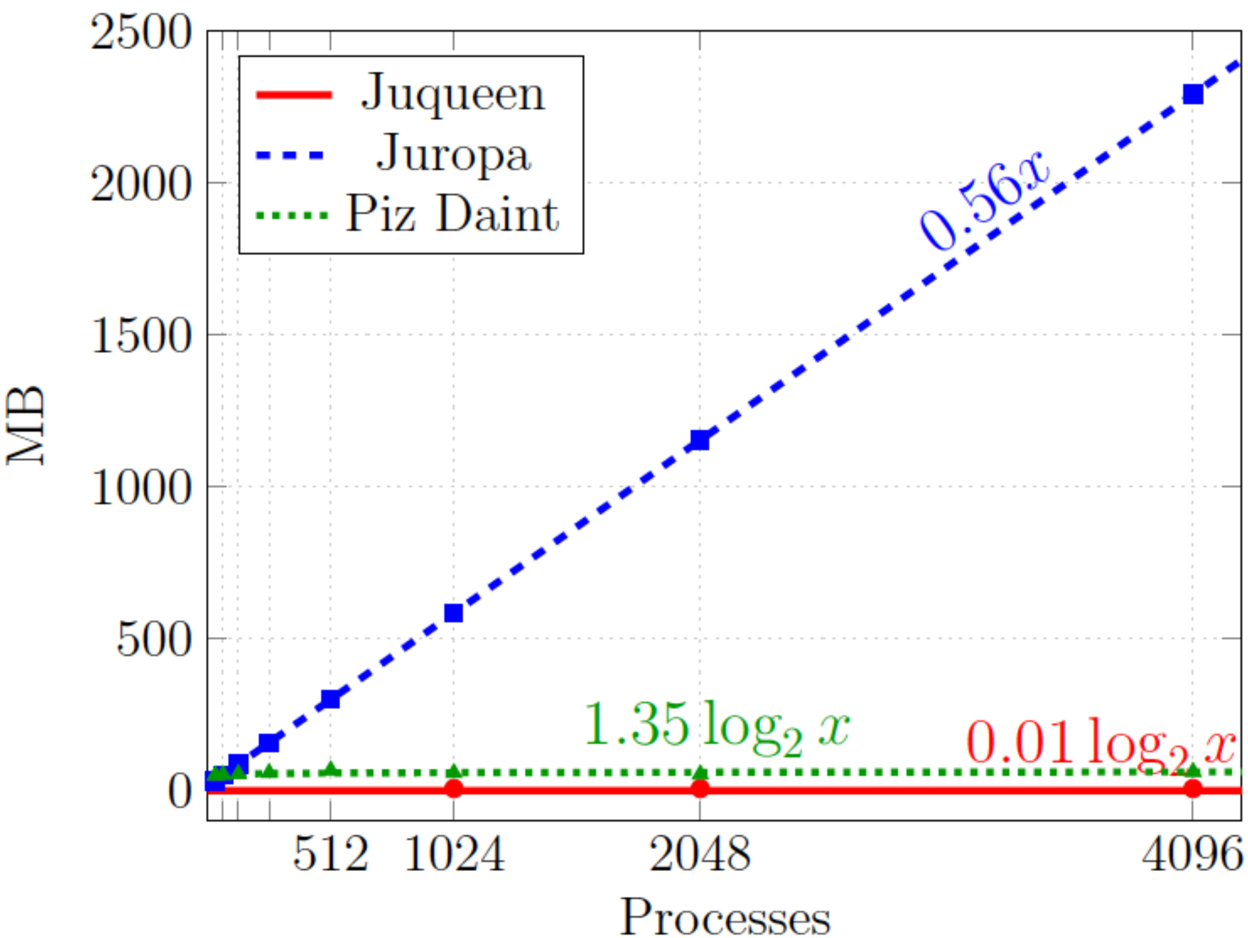
3. Network hardware differences (MU on BG/Q)

   T. Hoefler and M. Snir. Generic Topology Mapping Strategies for Large-scale Parallel Architectures, ICS' 11



computing.llnl.gov/tutorials/bgq

MPI memory consumption on all three systems



Linear memory consumption on Juropa

ParaStation MPI uses RC over IB

MAFIA

Sub-space clustering code used in data-mining

- Cluster dimensionality ***k*** is the model parameter
- Result: observed behavior matched the expectations

	gen	dedup	pcount	unjoin
Expectation	$O\ (k^3 2^k)$	$O\ (k^4 2^k)$	$O\ (k 2^k)$	$O\ (k^3 2^k)$
Model	$O\ (k^4 2^k)$	$O\ (k^4 2^k)$	$O\ (k 2^k)$	$O\ (k^2 2^k)$
Divergence	$O\ (k)$	$O\ (1)$	$O\ (1)$	$O\ (1/k)$
Match	~	✓	✓	~

Mass-producing performance models

- Is feasible
- Offers insight
- Requires low effort
- Improves code coverage

References

[1] Christian Iwainsky, Sergei Shudler, Alexandru Calotoiu, Alexandre Strube, Michael Knobloch, Christian Bischof, Felix Wolf: How Many Threads will be too Many? On the Scalability of OpenMP Implementations. In Proc. of the 21st Euro-Par Conference, Vienna, Austria of Lecture Notes in Computer Science, pages 451–463, Springer, August 2015.

Euro-Par 2015

[2] Andreas Vogel, Alexandru Calotoiu, Alexandre Strube, Sebastian Reiter, Arne Nägel, Felix Wolf, Gabriel Wittum: 10,000 Performance Models per Minute - Scalability of the UG4 Simulation Framework. In Proc. of the 21st Euro-Par Conference, Vienna, Austria of Lecture Notes in Computer Science, pages 519–531, Springer, August 2015.

Euro-Par 2015

[3] Sergei Shudler, Alexandru Calotoiu, Torsten Hoefler, Alexandre Strube, Felix Wolf: Exascaling Your Library: Will Your Implementation Meet Your Expectations?. In *Proc. of the International Conference on Supercomputing (ICS), Newport Beach, CA, USA, pages 1-11, ACM, June 2015*

ICS 2015

[4] Alexandru Calotoiu, Torsten Hoefler, Marius Poke, Felix Wolf: Using Automated Performance Modeling to Find Scalability Bugs in Complex Codes. In Proc. of the ACM/IEEE Conference on Supercomputing (SC13), Denver, CO, USA, pages 1-12, ACM, November 2013.

Thank you!

Upcoming tutorial @ SC15
Insightful Automatic Performance Modeling

Austin, Texas, USA, November 15