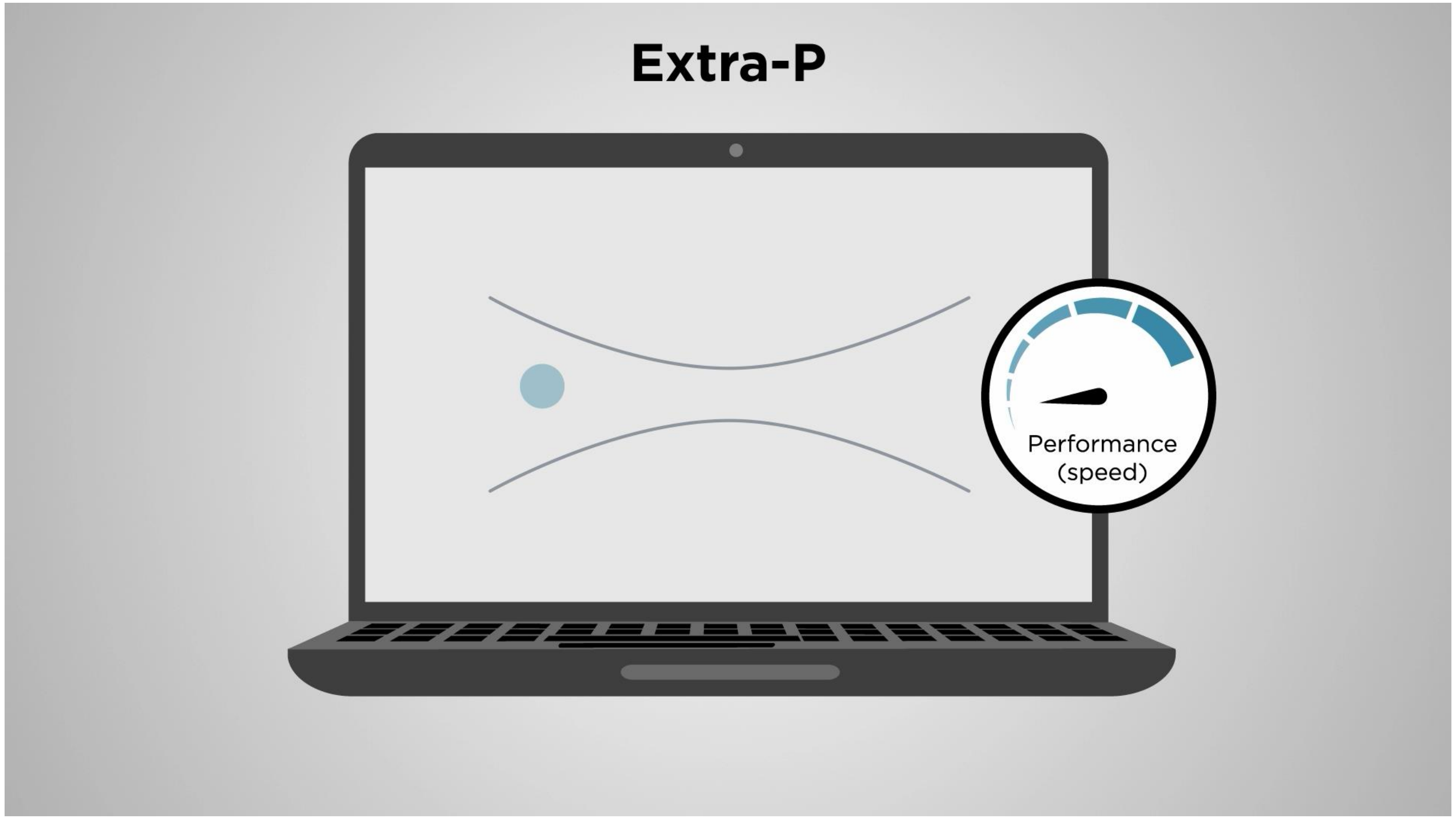
Denoising Application Performance Models with Noise-Resilient Priors

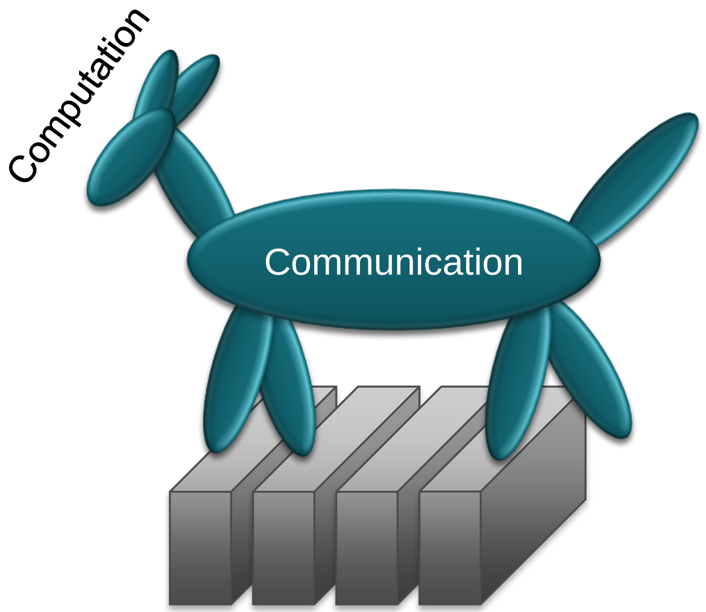
Motivation

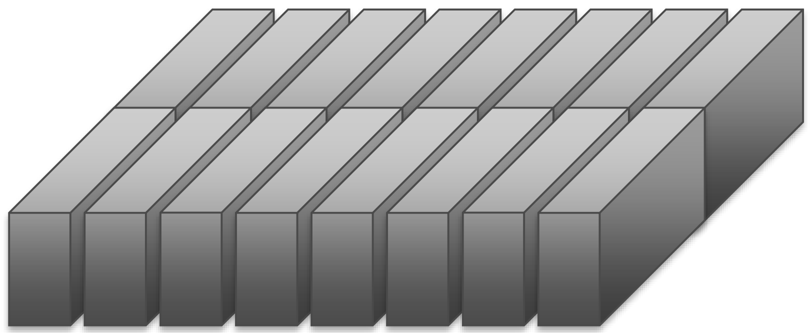


Watch Extra-P overview video

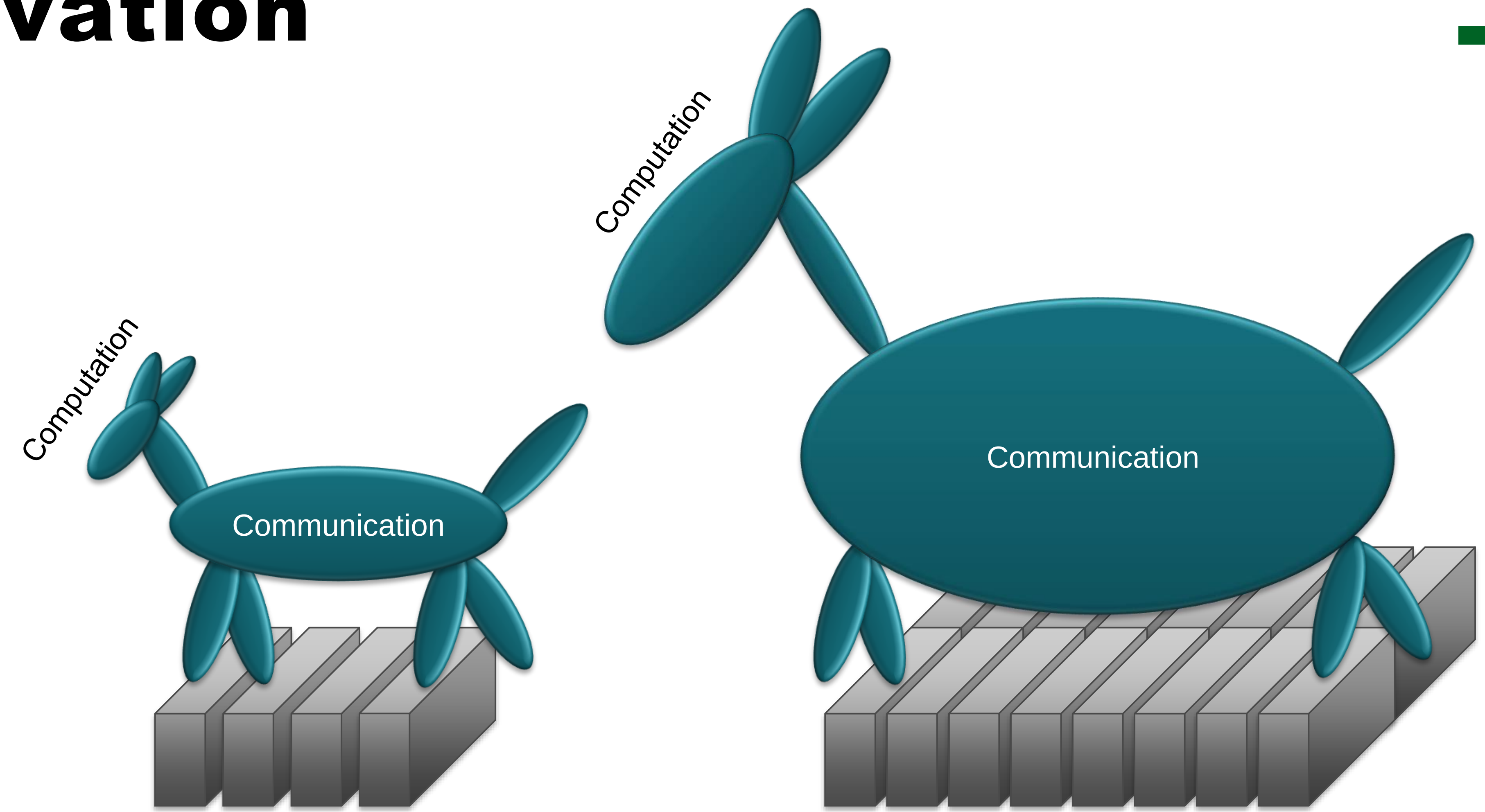
https://www.youtube.com/watch?v=Cv2YRCMWqBM

Motivation



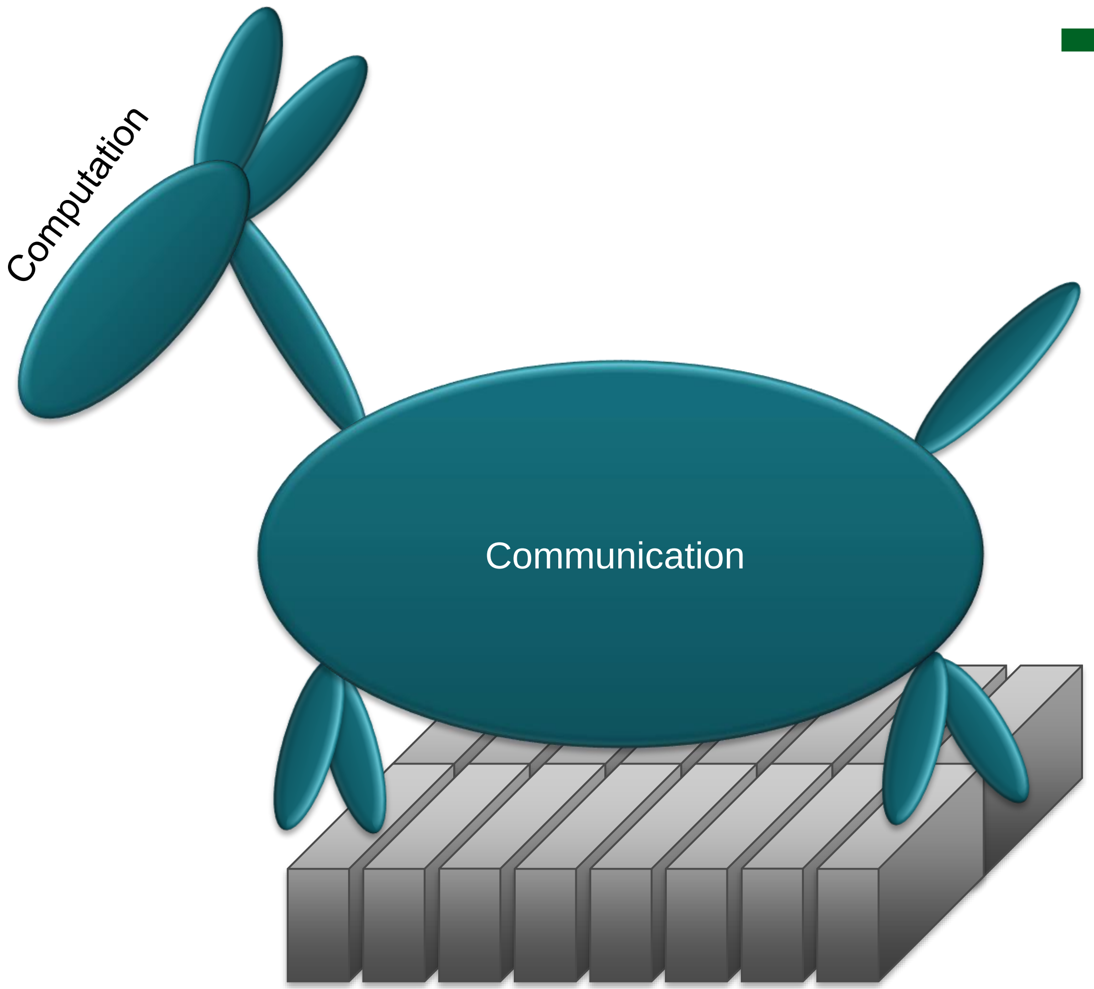


Motivation



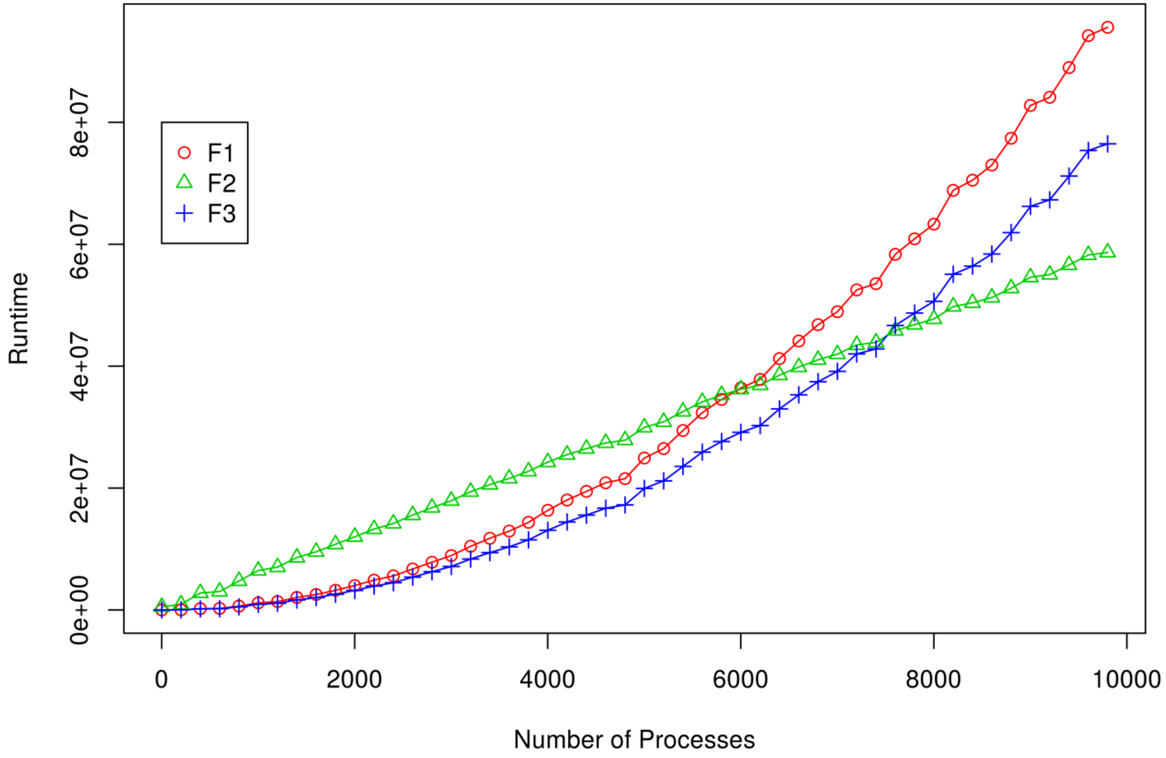
Motivation

We need to find scaling issues before they occur

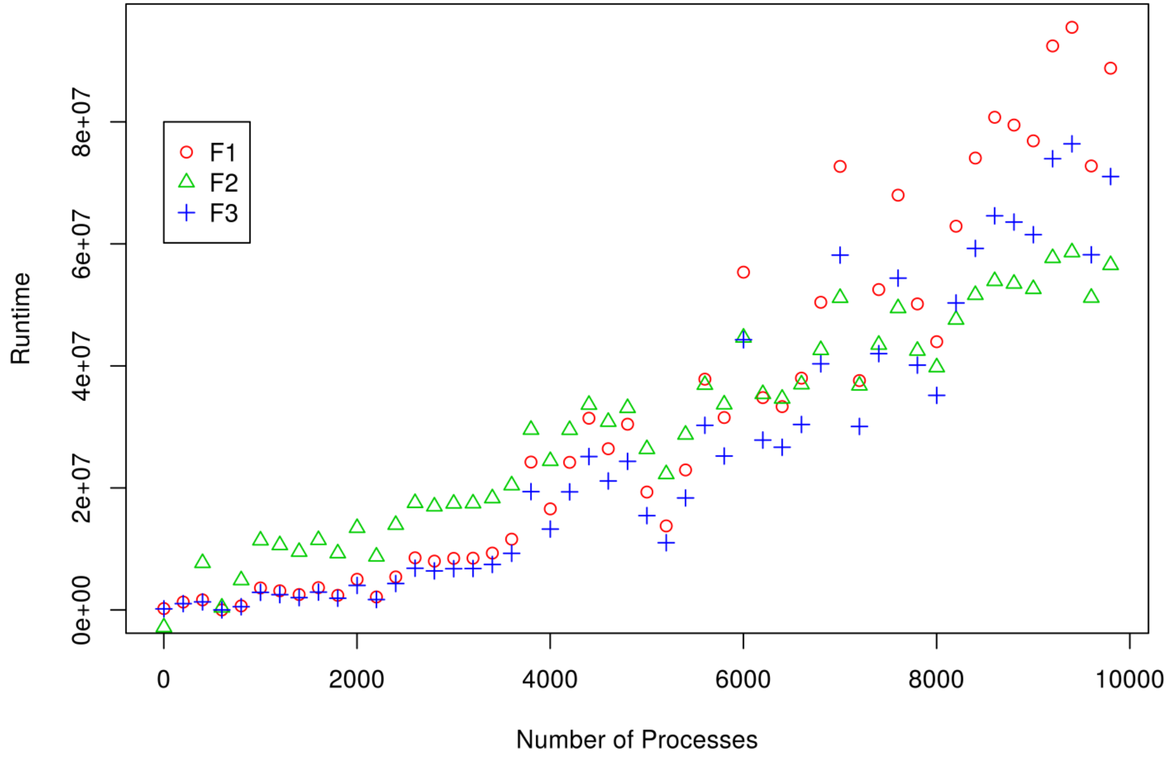


Motivation

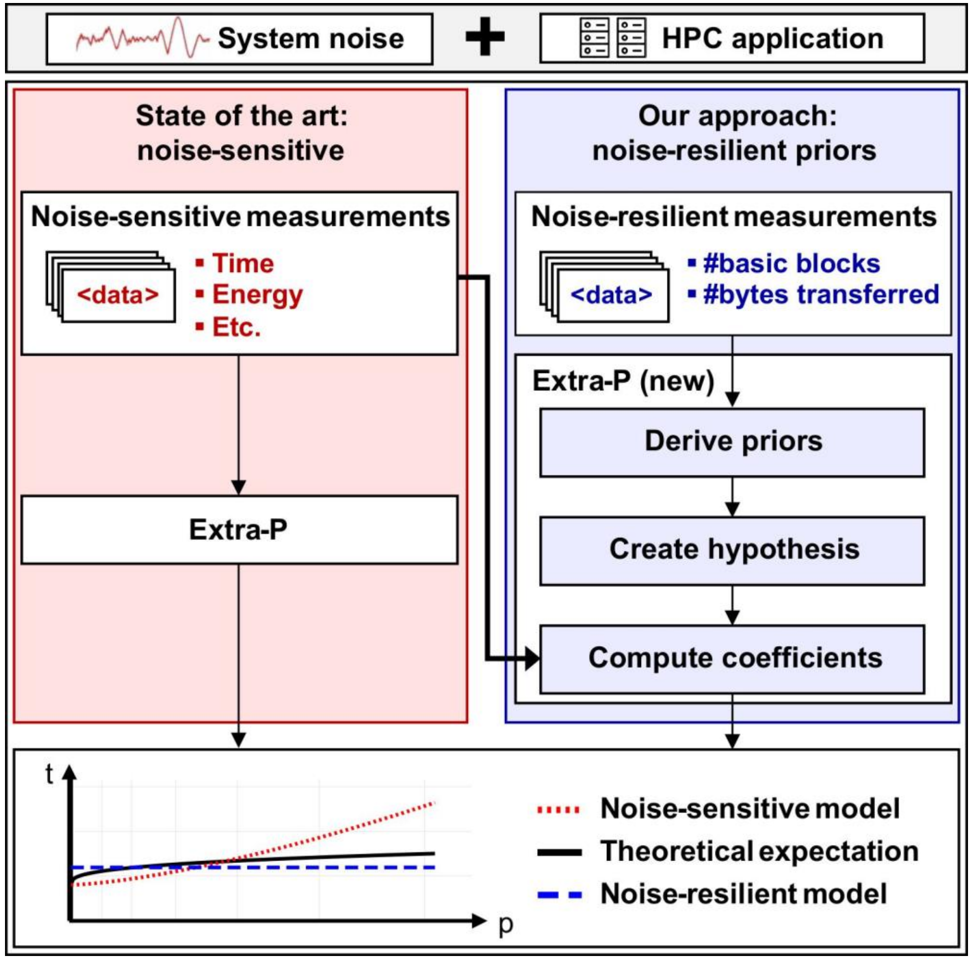
Common performance analysis chart in a paper



Production reality



Motivation



The noise-resilient model aligns with the theoretical expectation more closely

Noise-resilient measurements

- LLVM-IR [2] plug-in into Score-P [3] framework

Scalable performance measurement infrastructure for parallel codes

Computation

```
int foo(int a, int b)
{
    Basic block 1
    @Score-P counter
    ...
    Basic block n
    @Score-P counter
}
```

Communication

```
MPI_Bcast(A, n, MPI_INT,
          0, MPI_COMM_WORLD);
```

- $n \times p \times 4$ bytes sent from the root
- $n \times 4$ bytes received at each process

Multi-parameter performance modeling

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

- Performance Model Normal Format (PMNF) [1]

$$f(x_1, \dots, x_m) = \sum_{k=1}^{n} c_k \prod_{l=1}^{m} x_l^{i_{kl}} \cdot log_2^{j_{kl}}(x_l)$$

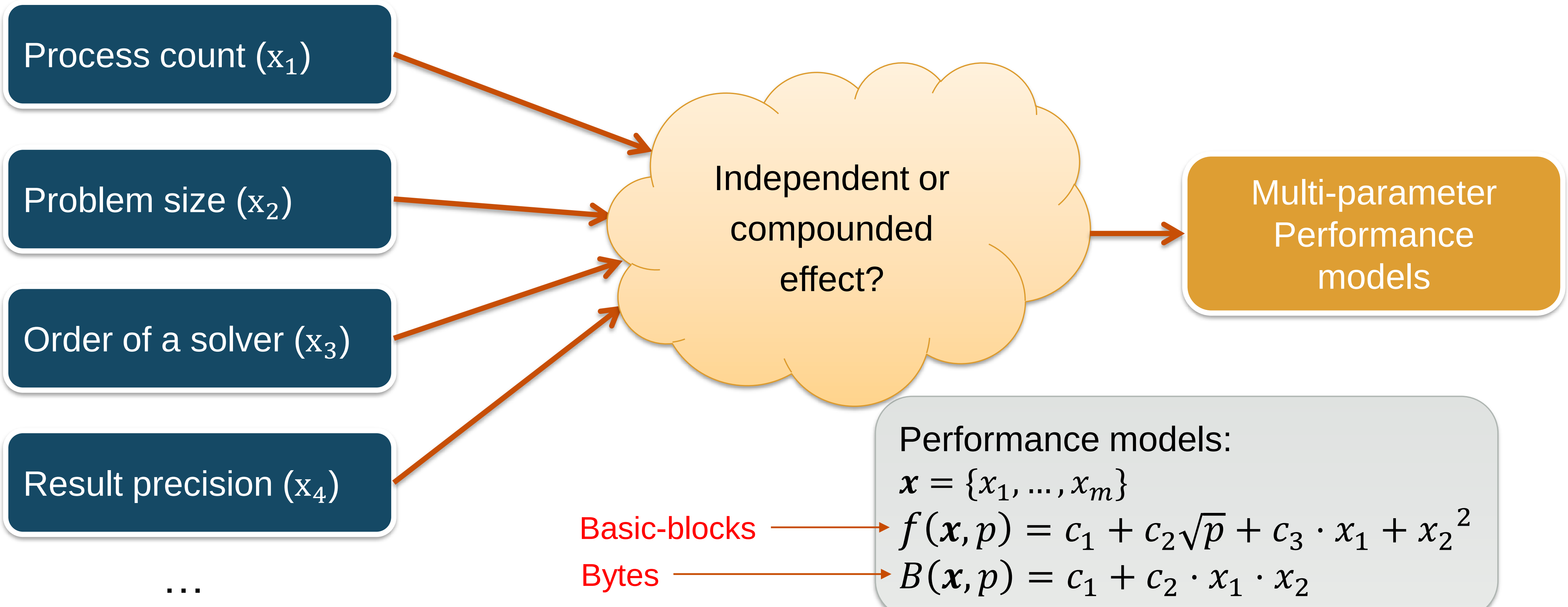
$$m, n \in \mathbb{N} \quad i_k \in I \quad j_k \in J \quad I, J \subset \mathbb{Q}$$

Model candidates

- Constant c_1
- Single parameter $c_1 + c_2 \cdot x_1$
- Multiple parameters
  - Additive $c_1 + c_2 \cdot x_1 + c_3 \cdot x_2$
  - Multiplicative $c_1 + c_2 \cdot x_1 \cdot x_2$
  - Complex $c_1 + c_2 \cdot x_1 \cdot x_2 + c_3 \cdot \log x_2 \cdot x_2^3$

...

Creating models from priors



Performance models:

$\boldsymbol{x} = \{x_1, \ldots, x_m\}$

Basic-blocks → $f(\boldsymbol{x}, p) = c_1 + c_2\sqrt{p} + c_3 \cdot x_1 + {x_2}^2$

Bytes → $B(\boldsymbol{x}, p) = c_1 + c_2 \cdot x_1 \cdot x_2$

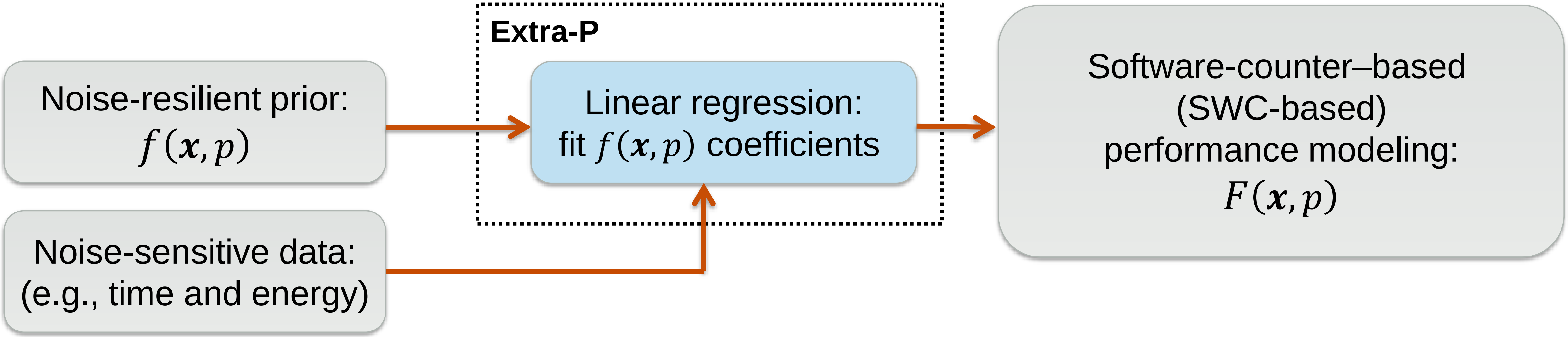
Creating models from priors

- Communication

MPI function	Expected runtime	Ref.
Send	$f(\boldsymbol{x},p) = \alpha + B(\boldsymbol{x},p) \cdot \beta$	[5]
Receive		
Broadcast	$f(\boldsymbol{x},p) = \log_2(p) \cdot \alpha + B(\boldsymbol{x},p) \cdot \beta$	[6]
Scatter	$f(\boldsymbol{x},p) = \log_2(p) \cdot \alpha + B(\boldsymbol{x},p) \cdot \frac{p-1}{p} \cdot \beta$	[6]
Gather		
Allgather		
Reduce	$f(\boldsymbol{x},p) = \log_2(p) \cdot \alpha + \left(\beta + \frac{p-1}{p} \cdot \gamma\right) \cdot B(\boldsymbol{x},p)$	[6]
Allreduce		

	Summary
$f(\boldsymbol{x},p)$	Prior model
$B(\boldsymbol{x},p)$	Bytes model
$\boldsymbol{x}$	Input parameters
p	MPI ranks
α	Latency
β	Bandwidth
γ	Computation cost

Creating models from priors



Creating models from priors

- Example

```
void F0 (int* V, int n, int p) {
  // int V[n*p];
  MPI_Bcast(V, n*p, ...);
  for(int i = 0; i < n/100; i++) {
    // calculate something
    for(int j = 0; j < p; j++) {
      // calculate something
}}}
```

Communication

Computation

Communication

Time: $13.0 \cdot 10^3 + 0.506n + 0.185np$

Bytes transferred: $0.1 \cdot 10^{-3} + 4.0np$

Theoretical model: $\log_2(p)\,\alpha + B(p,n)\beta$

Prior: $c_0 + c_1 \log_2(p) + c_2 np$

Noise-resilient: $7366 + 0 \log_2(p) + 0.189np$

Determine prior

Create model

Computation

Time: $-7782 + 0.755n + 0.002\, np^{\frac{3}{4}} \log_2^{\frac{2}{2}}(p)$

Basic blocks: $275 + 1.6n + 0.48np$

Prior: $c_0 + c_1 n + c_2 np$

Noise-resilient: $-7071 + 0.142n + 0.037np$

Determine prior

Create model

Benefits

Accuracy

Robustness to noise

Experimental cost

Evaluation

Test systems	Comparison baseline	Accuracy metrics
Lichtenberg II Cluster	SWC-based	Exponent deviation (ED) $ED(f_1(x_i), f_2(x_i)) = \|n_1 - n_2\|$
Jureca-DC Cluster	Classic	Relative error (RE) $RE\left(f_1(x_i)\right) = \frac{\|y_i - f_1(x_i)\|}{y} \cdot 100\%$
	DNN-based [6]	

Evaluation with synthetic benchmarks

- Benchmark Generator for parallel codes
  - Allows flexibility on the performance behavior
  - Functions with know theoretical analytical complexity

C++ kernels

for (int $i = 0; i < x^y; i++)\{$
$sum = v_1[i] + v_2[i]\}$

$O(x^y)$

for (int $i = 0; i < x; i++)\{$
for (int j $= 0; j < x; j++)\{$
$mul = v_1[i] * v_2[j]\}\}$

$O(x^2)$

for (int $i = 0; i < x^y; i++)\{$
$a = std :: \max(a, v_1[i]);$
$v_2[i] = a;\}$

$O(x^y)$

Evaluation with synthetic benchmarks

- Benchmark Generator

User definition

Input parameter (e.g., n)

Individual kernel complexity

Multiple kernel composition

C++ code

Kernel 1: MPI Bcast $\mathrm{V}[n]$

Kernel 2: $O(n^{\frac{1}{2}})$

Kernel 3: $O(n^1)$

Kernel 4: $O(n^1)$

Kernel 5: $O(n^0)$

Resulting complexity

Communication:
$O(\log_2(\mathrm{p}) + n)$

Computation:
$O(n^2)$

Evaluation with synthetic benchmarks

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

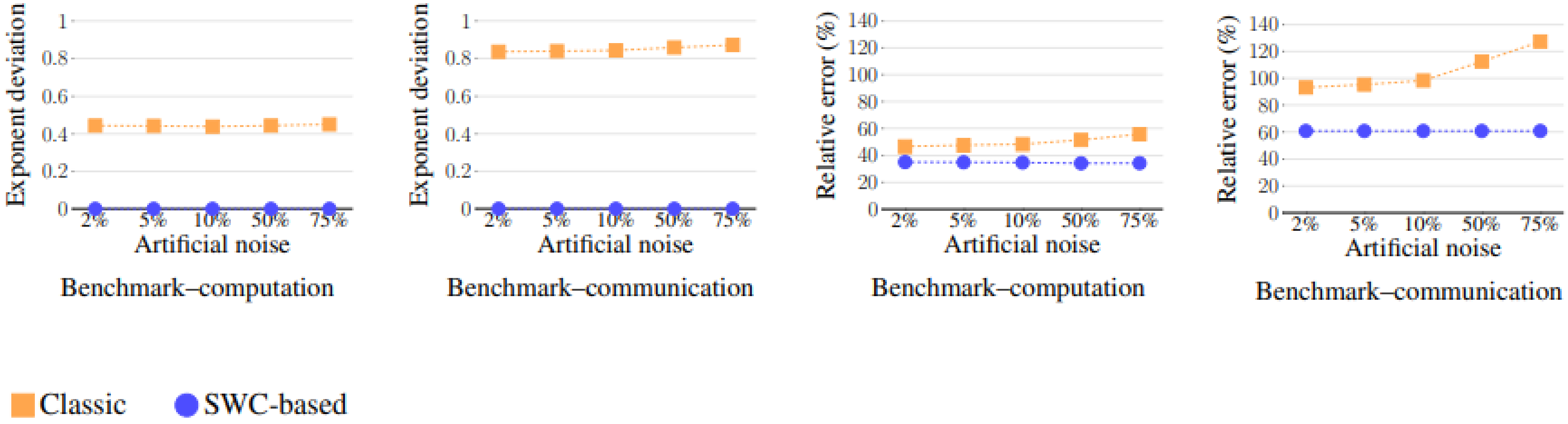
- 200 synthetic functions
- Accuracy
  - Exponent deviation: comparing performance models with their theoretical expectation

Models	Computation	Communication	
		MPI ranks	Message size
SWC-based	0	0	0
Classic	0.44	1.14	0.57

  - Relative error
    - SWC-based: 35% (computation) and 60% (communication)
    - Classic: 45% (computation) and 91% (communication)

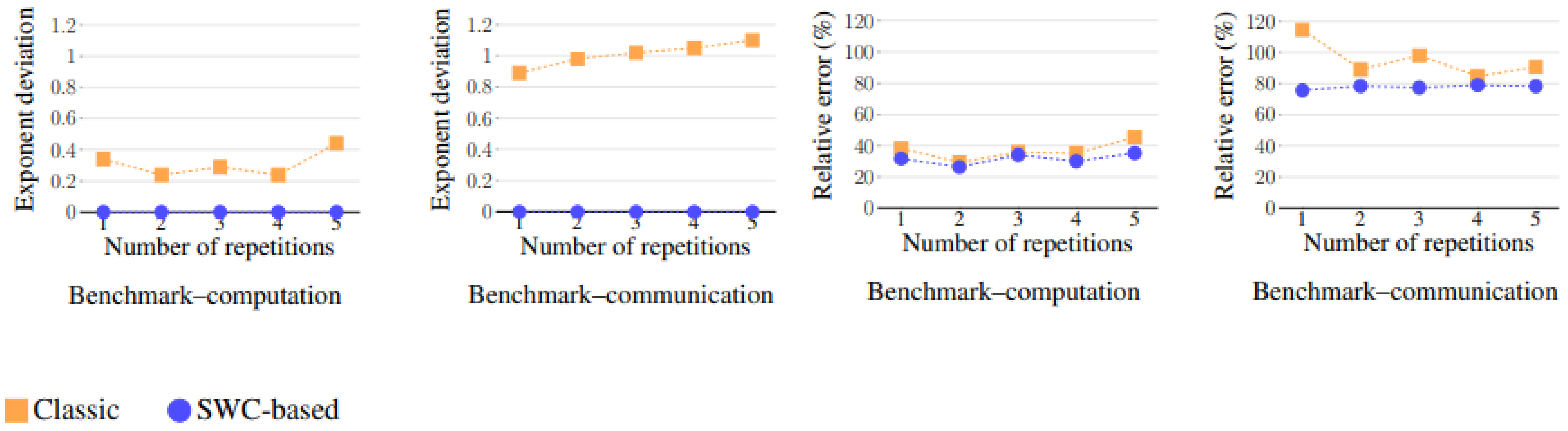
Evaluation with synthetic benchmarks

- Robustness to noise



Evaluation with synthetic benchmarks

- Experimental costs



Application case studies

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

- Kripke [7]
- Relearn [8]

Known theoretical performance

App/System	Training configurations	Test points
Kripke Lichtenberg II	$p \in \{512, 1000, 1728, 2744, 4096\}$ $G \in \{32, 64, 96, 128, 160\}$ $Z \in \{4^3, 8^3, 12^3, 16^3, 20^3\}$	$(p, G, Z) = (5832, 160, 20^3)$ $(p, G, Z) = (4096, 192, 20^3)$ $(p, G, Z) = (4096, 160, 24^3)$
RELeARN Jureca-DC	$p \in \{32, 64, 128, 256, 512\}$ $n \in \{250, 300, 350, 400, 450\}$	$(p, n) = (1024, 450)$ $(p, n) = (512, 500)$

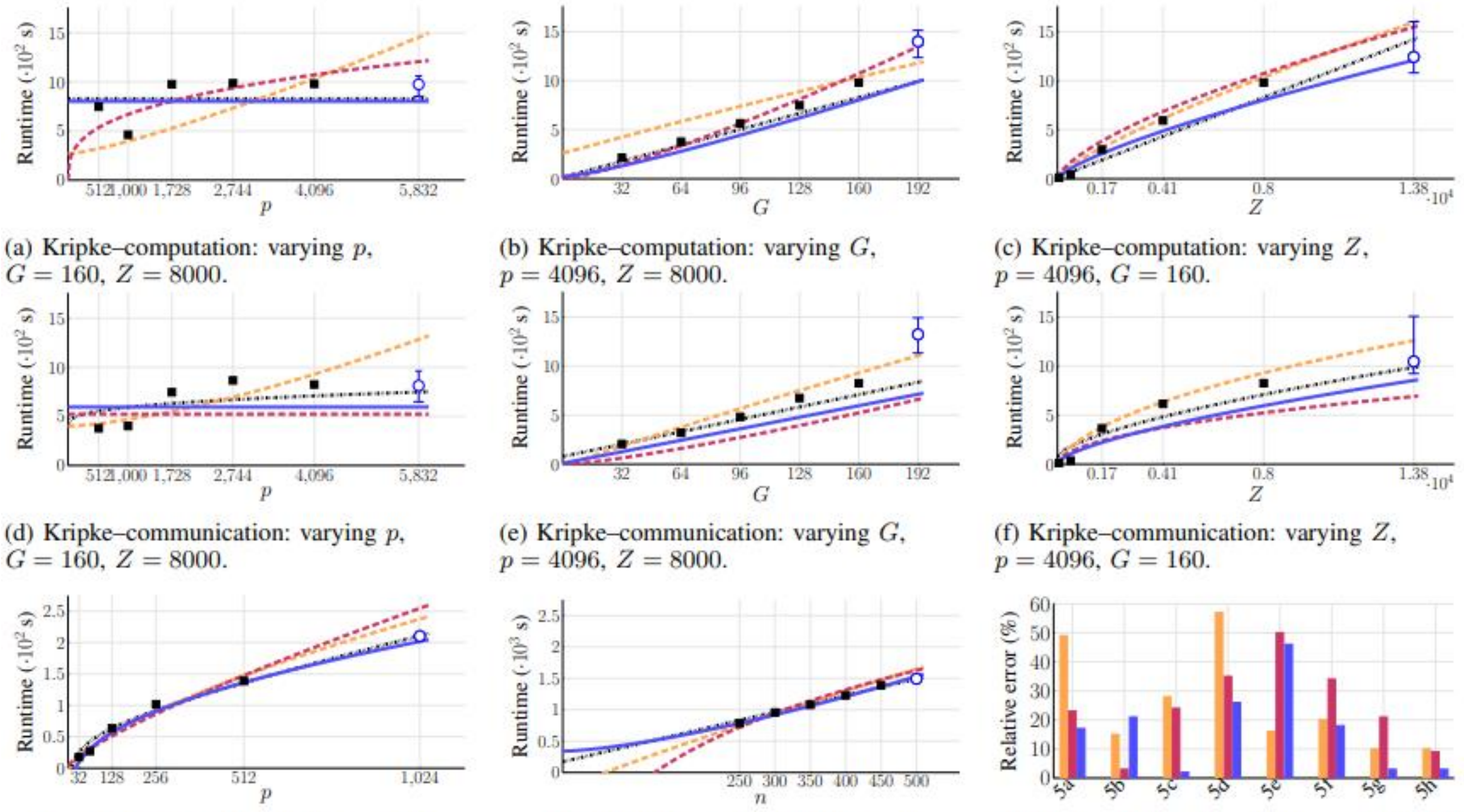
Evaluation

- Application case studies
  - Accuracy

Model	Asymptotic complexity	ED		
Kripke–computation		Δp	ΔG	ΔZ
Theoretical	$\mathcal{O}(G \cdot Z)$			
Classic	$\mathcal{O}(p \cdot \log_2^2(p) \cdot G^{\frac{3}{4}} \cdot \log_2(G) \cdot Z^{\frac{4}{5}})$	1	0.25	**0.20**
DNN-based	$\mathcal{O}(p \cdot G^{\frac{5}{4}} \cdot Z^{\frac{2}{3}})$	1	0.25	0.33
SWC-based	$\mathcal{O}(G \cdot \log_2(G) \cdot Z^{\frac{3}{4}})$	**0**	**0**	0.25
Kripke–communication		Δp	ΔG	ΔZ
Theoretical	$\mathcal{O}(p^{\frac{1}{3}} + G \cdot Z^{\frac{2}{3}})$			
Classic	$\mathcal{O}(p^{\frac{4}{3}} \cdot \log_2(p) \cdot G^{\frac{3}{4}} \cdot \log_2(G) \cdot Z^{\frac{1}{3}} \cdot \log_2^2(Z))$	1	0.25	0.33
DNN-based	$\mathcal{O}(G^{\frac{5}{4}} \cdot Z^{\frac{1}{2}})$	**0.33**	0.25	0.16
SWC-based	$\mathcal{O}(G \cdot Z^{\frac{2}{3}})$	**0.33**	**0**	**0**
RELeARN		Δp	Δn	
Theoretical	$\mathcal{O}(p + n \cdot \log_2(n \cdot p))$			
Classic	$\mathcal{O}(p^{\frac{2}{3}} \cdot n^{\frac{3}{4}} \cdot \log_2(n))$	0.33	**0.25**	
DNN-based	$\mathcal{O}(p^{\frac{2}{3}} \cdot \log_2(p) \cdot n^{\frac{1}{4}})$	0.33	0.75	
SWC-based	$\mathcal{O}(p + n^{\frac{5}{4}} \cdot \log_2(n) \cdot p^{\frac{1}{4}})$	**0**	**0.25**	

Evaluation

- Application case studies
  - Accuracy
  - Better in 6/8 cases



(a) Kripke–computation: varying p, $G = 160$, $Z = 8000$.

(b) Kripke–computation: varying G, $p = 4096$, $Z = 8000$.

(c) Kripke–computation: varying Z, $p = 4096$, $G = 160$.

(d) Kripke–communication: varying p, $G = 160$, $Z = 8000$.

(e) Kripke–communication: varying G, $p = 4096$, $Z = 8000$.

(f) Kripke–communication: varying Z, $p = 4096$, $G = 160$.

(g) RELeARN: varying p, $n = 450$.

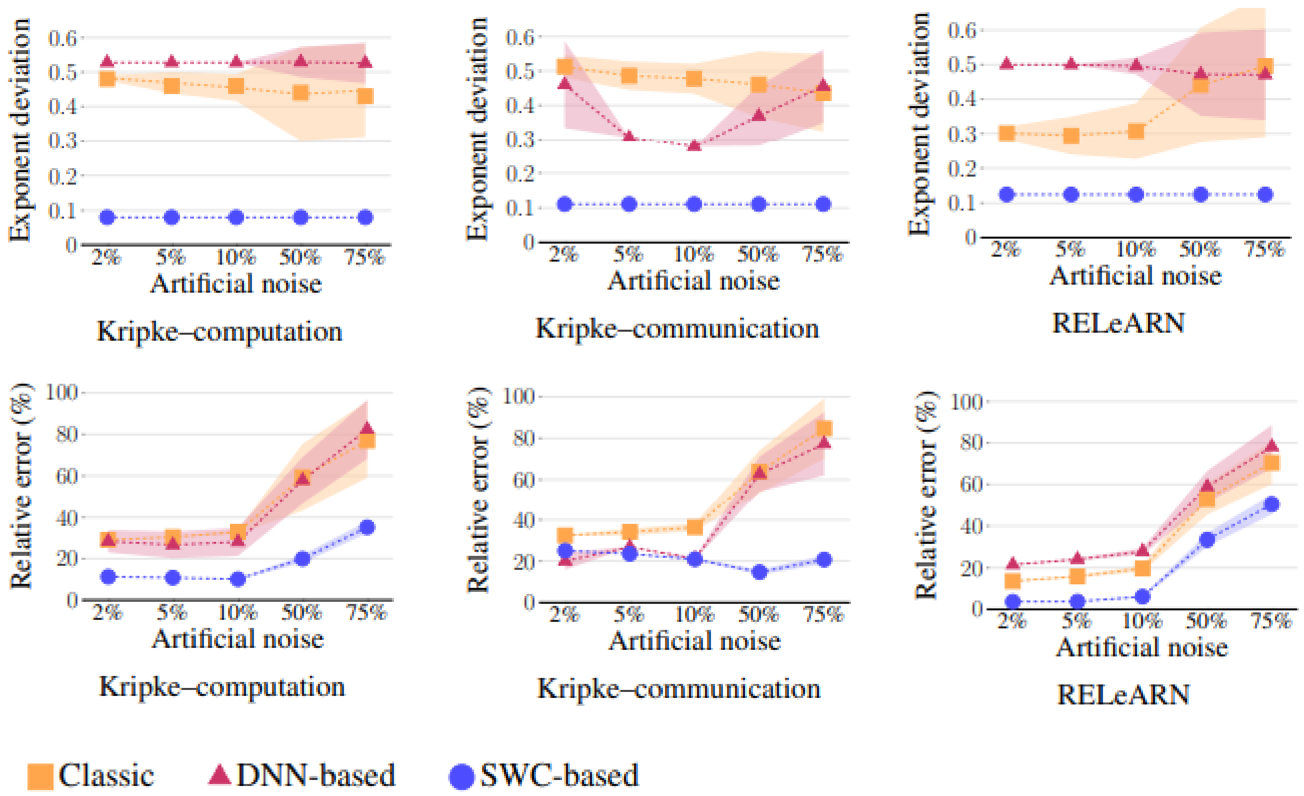
(h) RELeARN: varying n, $p = 512$.

(i) Relative error values of each model on test points.

Theoretical | Classic | DNN-based | SWC-based | Training data | Test data

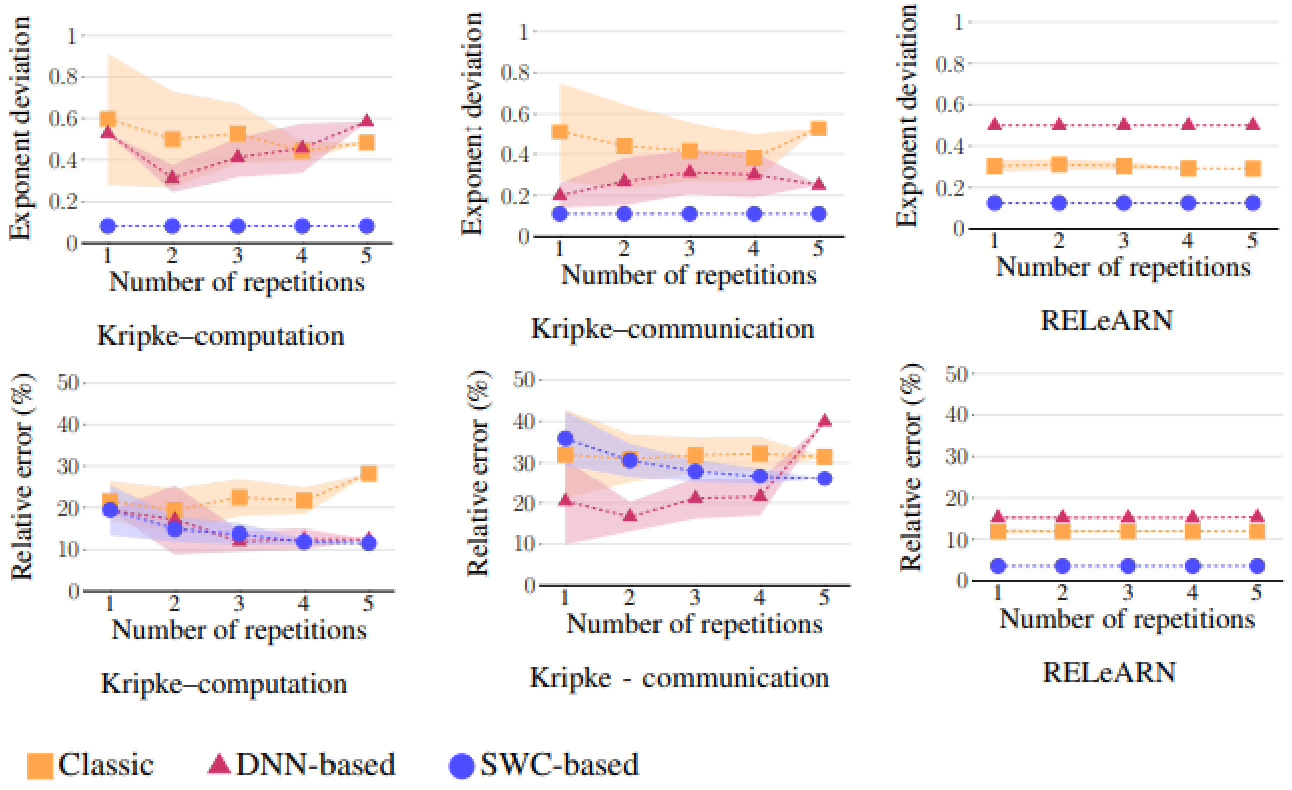
Application case studies

- Robustness to noise



Application case studies

- Experimental costs



Conclusion

- Our method accurately captures the computational effort of an application in close alignment with its theoretical performance model
- We reduce, if not eliminate, the need for multiple time measurements
- Under artificial noise, our models maintained stable error rates

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

References

[1] A. Calotoiu, D. Beckinsale, C. W. Earl, T. Hoefler, I. Karlin, M. Schulz, and F. Wolf, “Fast multi-parameter performance modeling,” in 2016 IEEE International Conference on Cluster Computing (CLUSTER). IEEE, 2016, pp. 172–181.

[2] LLVM admin team. (2023) LLVM website. Accessed 2023/08/21.[Online]. Available: https://llvm.org/

[3] Score-P developer community. (2023) Scalable performance measurement infrastructure for parallel codes (Score-P). Accessed 2023/08/21. [Online]. Available: https://www.vi-hps.org/projects/score-p

[4] W. Zhang, M. Hao, and M. Snir, “Predicting hpc parallel program performance based on llvm compiler,” Cluster Computing, vol. 20, pp.1179–1192, 2017.

[5] E. Chan, M. Heimlich, A. Purkayastha, and R. Van De Geijn, “Collective communication: theory, practice, and experience,” Concurrency and Computation: Practice and Experience, vol. 19, no. 13, pp. 1749–1783, 2007.

References

[6] M. Ritter, A. Geiß, J. Wehrstein, A. Calotoiu, T. Reimann, T. Hoefler, and F. Wolf, “Noise-resilient empirical performance modeling with deep neural networks,” in 2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS). IEEE, 2021, pp. 23–34

[7] A. J. Kunen, T. S. Bailey, and P. N. Brown, “Kripke-a massively parallel transport mini-app,” Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), Tech. Rep., 2015

[8] S. Rinke, M. Butz-Ostendorf, M.-A. Hermanns, M. Naveau, and F. Wolf, “A scalable algorithm for simulating the structural plasticity of the brain,” Journal of Parallel and Distributed Computing, vol. 120, pp. 251–266, 2018

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

Thank you!

- You can contact us via email: extra-p-support@lists.parallel.informatik.tu-darmstadt.de
- Or on GitHub using the issues tool: https://github.com/extra-p/extrap