

SMAUG: System-Level Modelling and Optimized Use of Disruptive Memory Technologies

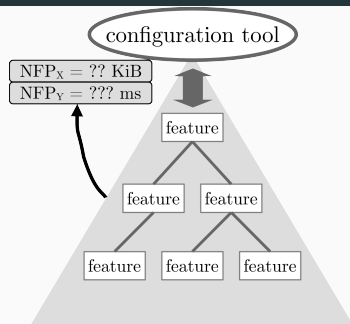
Birte Friesel, Marcel Lütke Dreimann, Olaf Spinczyk

Oct 1st, 2026

ess.cs.uos.de/~bkf

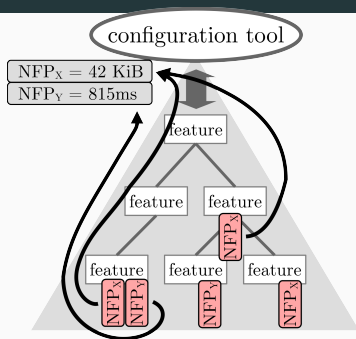
birte.friesel@uos.de

Motivation: Runtime Performance Prediction



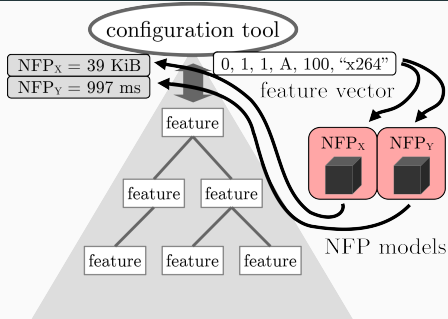
- Common question: how does software configuration affect latency / throughput?

Motivation: Runtime Performance Prediction



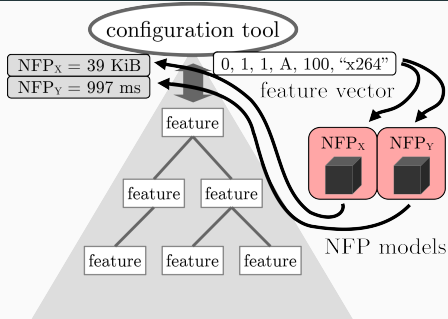
- Common question: how does software configuration affect latency / throughput?
- Accepted solution: feature model + performance model

Motivation: Runtime Performance Prediction



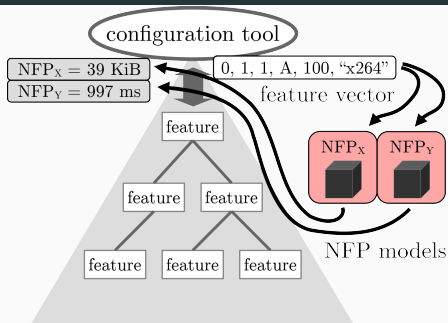
- Common question: how does software configuration affect latency / throughput?
- Accepted solution: feature model + performance model

Motivation: Runtime Performance Prediction



- Common question: how does software configuration affect latency / throughput?
- Accepted solution: feature model + performance model
- But: models only consider **software** variability

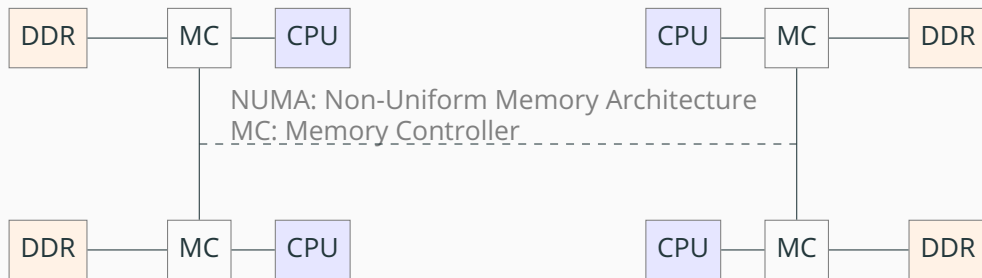
Motivation: Runtime Performance Prediction



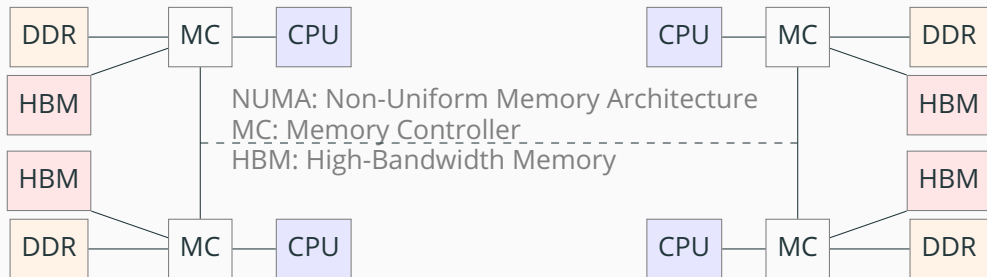
- Common question: how does software configuration affect latency / throughput?
- Accepted solution: feature model + performance model
- But: models only consider **software** variability

- Underlying hardware: affects performance, but not part of model [Jam+17]
- Even on same hardware, model may be incorrect:
runtime placement of compute and data affects performance

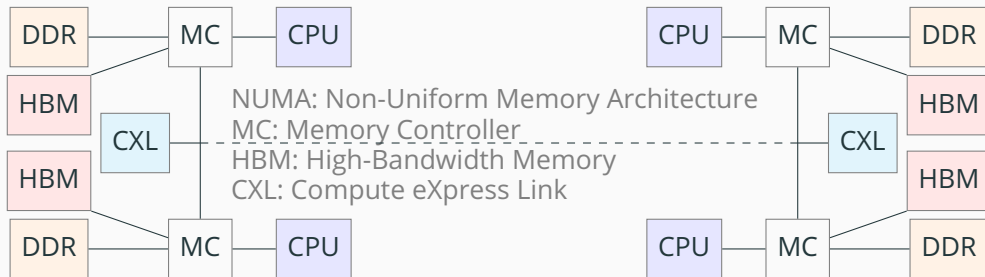
Runtime Performance on Modern Hardware



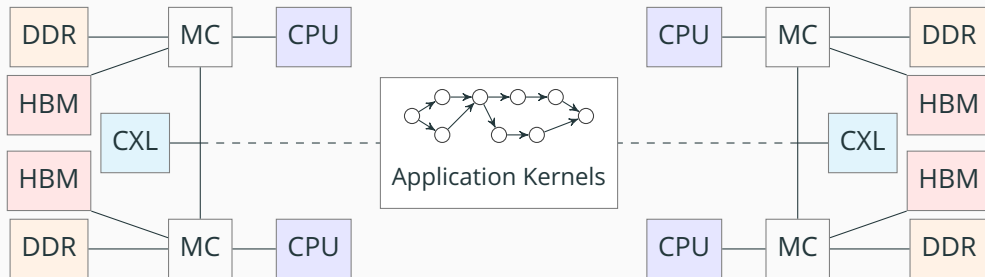
Runtime Performance on Modern Hardware

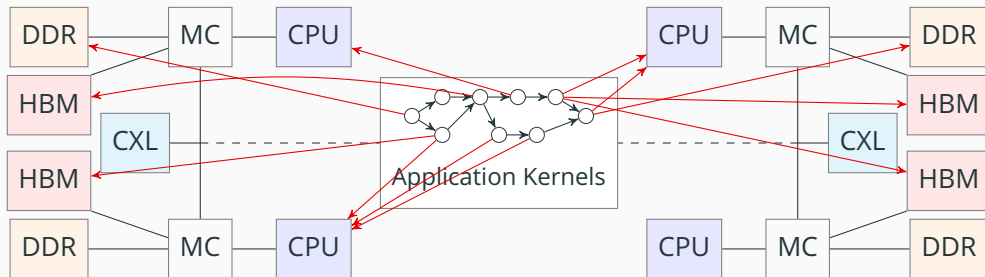


Runtime Performance on Modern Hardware

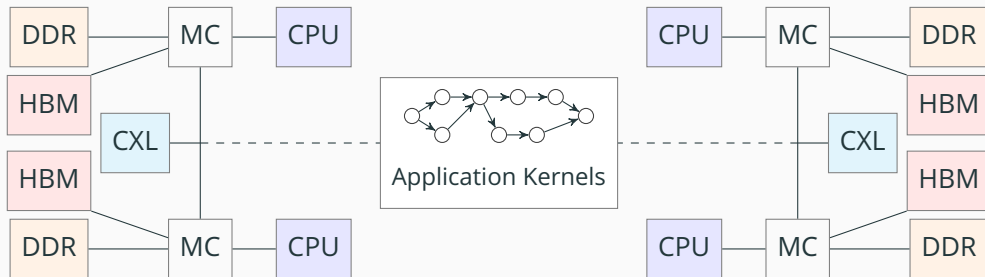


Runtime Performance on Modern Hardware





- Task and data placement are a **source of variability**



- Task and data placement are a **source of variability**
 - Systems perspective: automatic placement by OS is not always optimal
- Plug performance model into OS for **model-guided** placement decisions

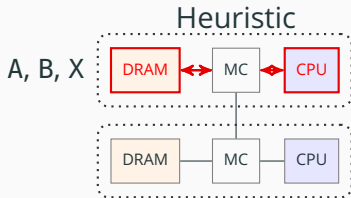
Example: Vector Addition



Example (Vector Addition)

$\vec{x} = \vec{a} + \vec{b} \quad \hat{=}$ `for (i = 0; i < n; i++) { X[i] = A[i] + B[i] }`

Heuristic: A, B, X and compute on same NUMA node (local data)



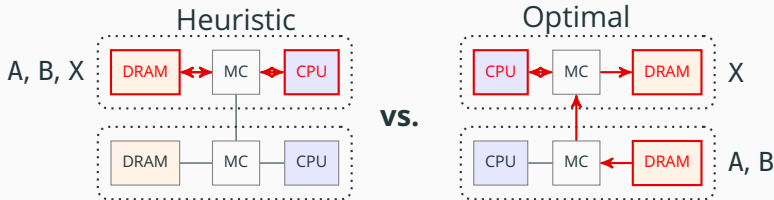
Example: Vector Addition



Example (Vector Addition)

$\vec{x} = \vec{a} + \vec{b} \quad \hat{=}$ for (i = 0; i < n; i++) { X[i] = A[i] + B[i] }

Heuristic: A, B, X and compute on same NUMA node (local data)



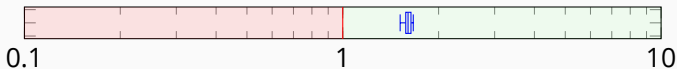
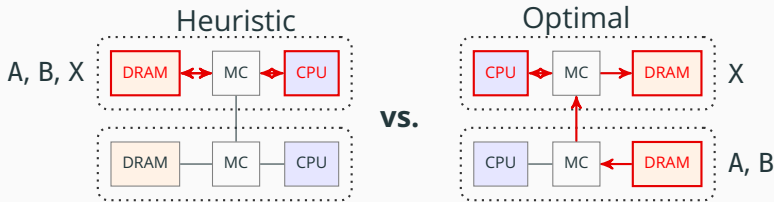
Example: Vector Addition



Example (Vector Addition)

$\vec{x} = \vec{a} + \vec{b} \quad \hat{=}$ for (i = 0; i < n; i++) { X[i] = A[i] + B[i] }

Heuristic: A, B, X and compute on same NUMA node (local data)



Optimal Placement: 1.6× Speedup over Heuristic

Example: Vector Addition

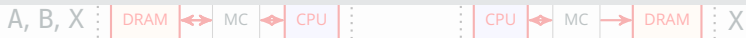


Example (Vector Addition)

$\vec{x} = \vec{a} + \vec{b} \quad \hat{=}$ `for (i = 0; i < n; i++) { X[i] = A[i] + B[i] }`

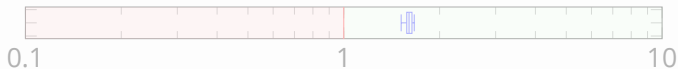
Heuristic **Observation**

Variability in hardware and system topology must be considered when deciding on compute and data placement.



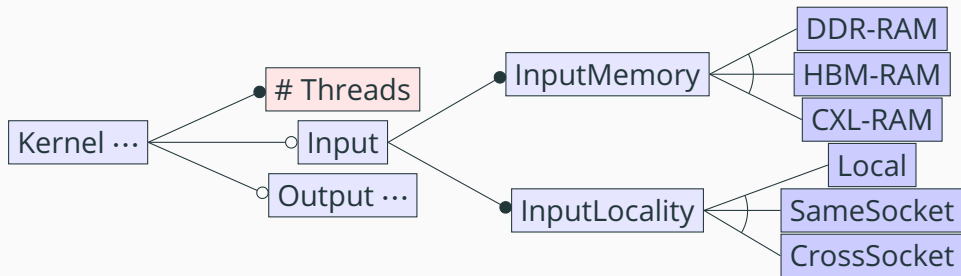
Discussion / Community Perspective

How to describe variability in hardware configuration and/or runtime placement? **Explicit component** of the feature model?

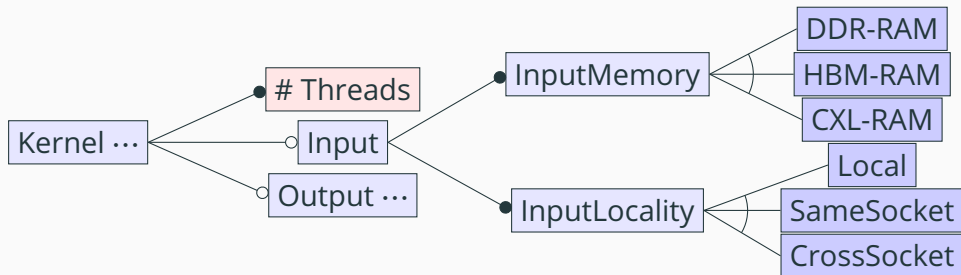


Optimal Placement: $1.6\times$ Speedup over Heuristic

Straightforward Approach

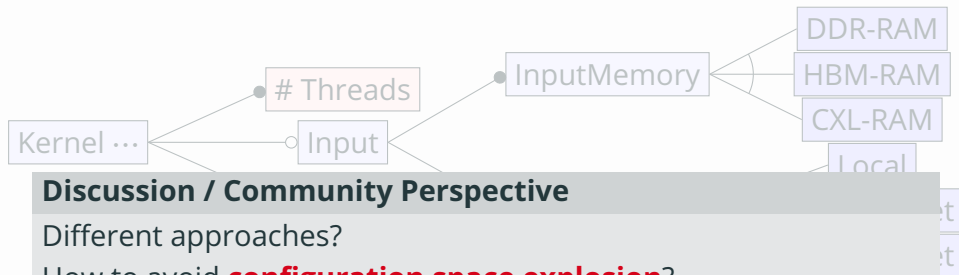


Straightforward Approach



→ $3 \cdot 3 \cdot 3 \cdot 3 = 3^4 = 81$ configurations **for each kernel** (excl. # Threads)

- With m kernels: up to $81 \cdot m$ placement configurations
- With n non-placement software configurations: up to $81 \cdot m \cdot n$ total



Discussion / Community Perspective

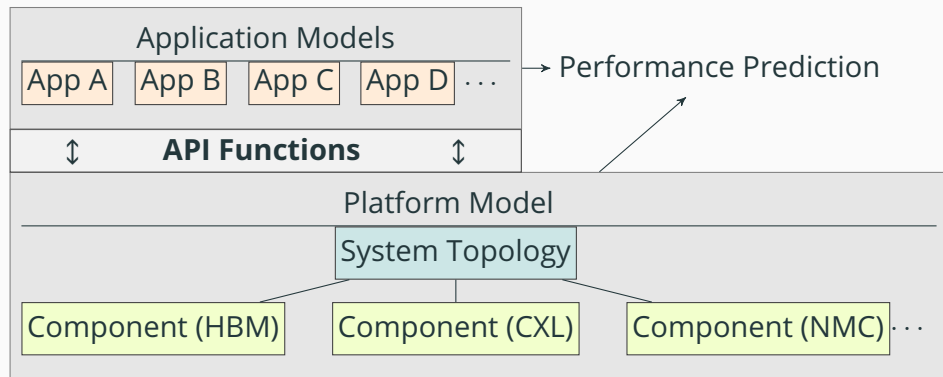
Different approaches?

How to avoid **configuration space explosion**?

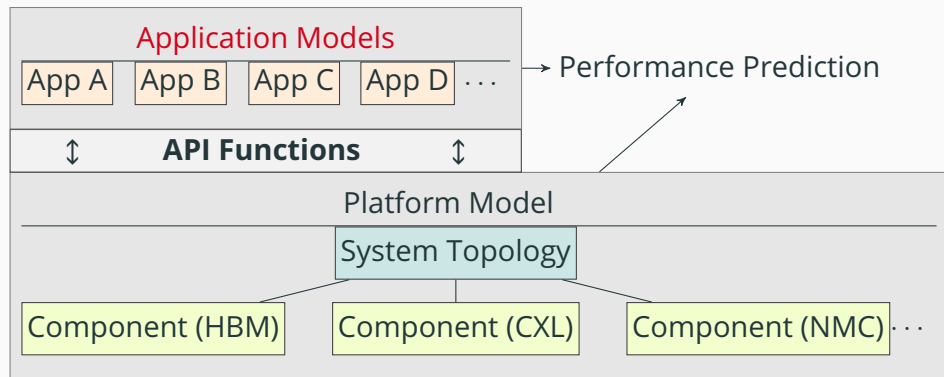
→ $3 \cdot 3 \cdot 3 \cdot 3 = 3^4 = 81$ configurations **for each kernel** (excl. # Threads)

- With m kernels: up to $81 \cdot m$ placement configurations
- With n non-placement software configurations: up to $81 \cdot m \cdot n$ total

SMAUG Approach: Disentangle Variability Sources

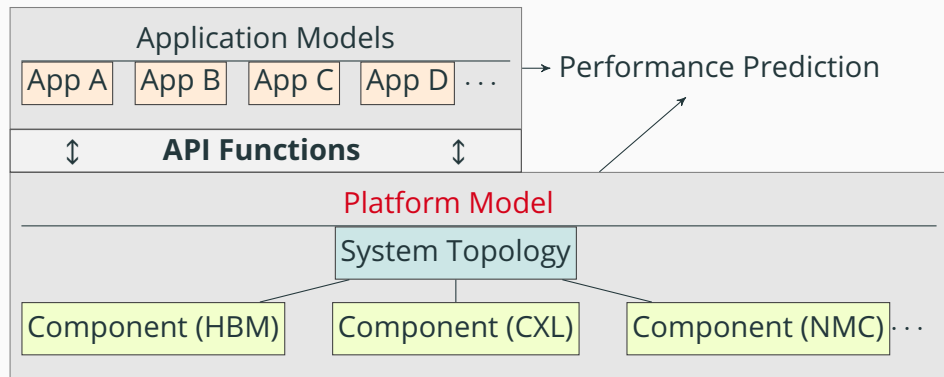


SMAUG Approach: Disentangle Variability Sources



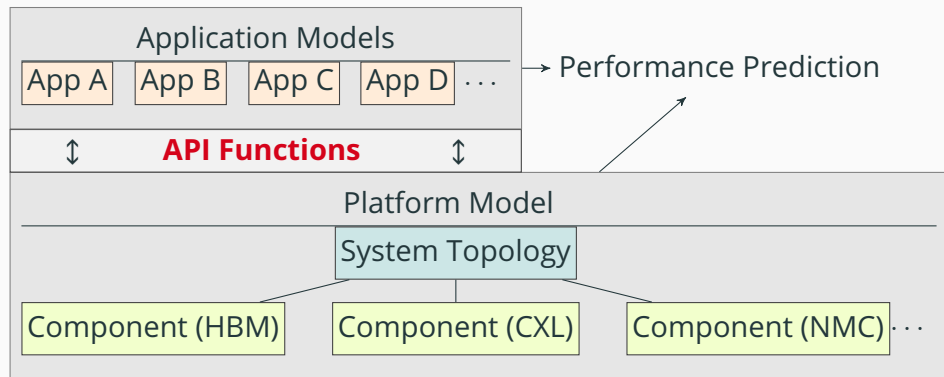
- Application variability: independent of hardware; no performance models

SMAUG Approach: Disentangle Variability Sources



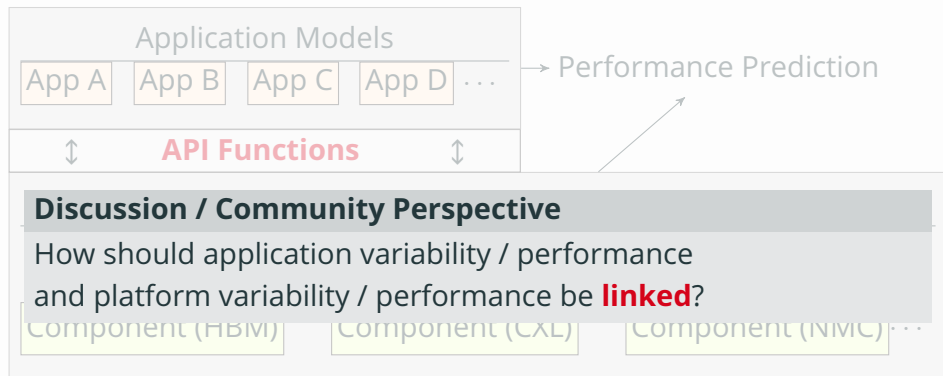
- Application variability: independent of hardware; no performance models
- Platform variability and performance: independent of applications

SMAUG Approach: Disentangle Variability Sources



- Application variability: independent of hardware; no performance models
- Platform variability and performance: independent of applications

SMAUG Approach: Disentangle Variability Sources



- Application variability: independent of hardware; no performance models
- Platform variability and performance: independent of applications



- Variability: configuration and runtime-dependent **workload attributes**
 - E.g. size of input file or database



- Variability: configuration and runtime-dependent **workload attributes**
 - E.g. size of input file or database
- Use **behaviour model** to predict sequence of API calls
 - ≈ featured transition systems / automata [Bou+08; Cla+13; FL17; Ola+16]



- Variability: configuration and runtime-dependent **workload attributes**
 - E.g. size of input file or database
- Use **behaviour model** to predict sequence of API calls
 - ≈ featured transition systems / automata [Bou+08; Cla+13; FL17; Ola+16]
- But: we are dealing with control flows → **featured control flow graphs**
 - Feature guards as-is; actions in states (not transitions)



$(\# = 128 \wedge outer_cols > 128)$
 $\vee (\# = 256 \wedge outer_cols > 256) \vee \dots$
 $\vee \# \in \{1, \dots, 127, 129, \dots, 255, 257, \dots\}$

$(\# = 128 \wedge outer_cols \leq 128)$
 $\vee (\# = 256 \wedge outer_cols \leq 256) \vee \dots$



$(\# = 128 \wedge outer_cols > 128)$
 $\vee (\# = 256 \wedge outer_cols > 256) \vee \dots$
 $\vee \# \in \{1, \dots, 127, 129, \dots, 255, 257, \dots\}$

$(\# = 128 \wedge outer_cols \leq 128)$
 $\vee (\# = 256 \wedge outer_cols \leq 256) \vee \dots$

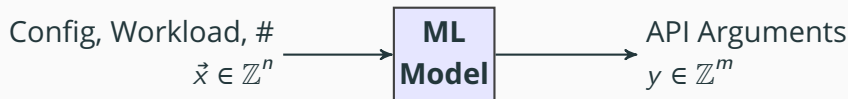
- Predict API call sequence ($\hat{=}$ visited states) from software configuration and workload attributes, while keeping track of **loop iterations** (#)



$(\# = 128 \wedge \text{outer_cols} > 128)$
 $\vee (\# = 256 \wedge \text{outer_cols} > 256) \vee \dots$
 $\vee \# \in \{1, \dots, 127, 129, \dots, 255, 257, \dots\}$

$(\# = 128 \wedge \text{outer_cols} \leq 128)$
 $\vee (\# = 256 \wedge \text{outer_cols} \leq 256) \vee \dots$

- Predict API call sequence ($\hat{=}$ visited states) from software configuration and workload attributes, while keeping track of loop iterations (#)
- Each state is annotated with an ML model $\rightarrow$ predict API call arguments





- L^* / FL^* not (easily) applicable [Ang87; For21; For23]
 - Limited to boolean features, unclear support for dealing with loops



- L^* / FL^* not (easily) applicable [Ang87; For21; For23]
 - Limited to boolean features, unclear support for dealing with loops
- Trace-based learning approach
- Hence the switch to actions in states (not transitions)



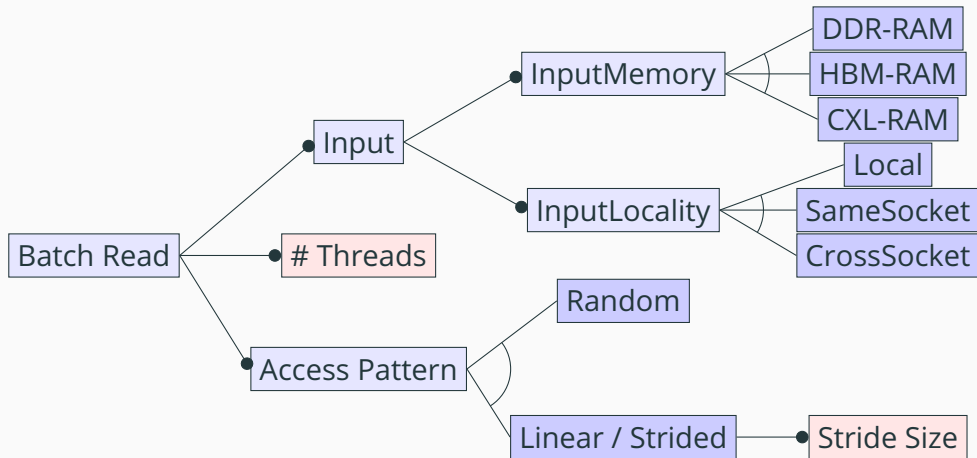
- L^* / FL^* not (easily) applicable [Ang87; For21; For23]
 - Limited to boolean features, unclear support for dealing with loops
- Trace-based learning approach
- Hence the switch to actions in states (not transitions)

Discussion / Variability Modelling Perspective

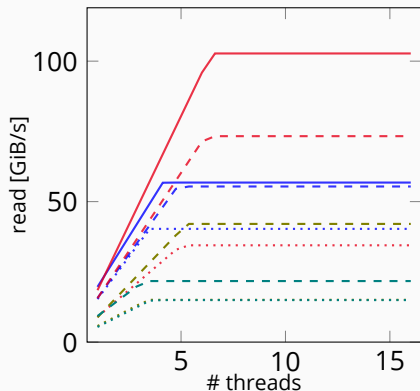
Is this a sensible approach?

Are there better **learning algorithms**?

Platform Model: Variability Example



Platform Model: Performance Example

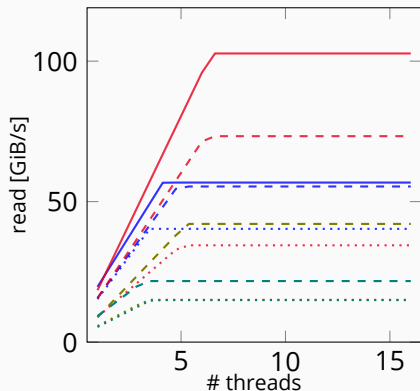


HBM / DDR / CXL x16 / CXL x8

local / intra - / cross -package

- Conventional performance model: $f(\vec{x}) = y$
 - Independent of application
 - Learnt from microbenchmarks
 - $\vec{x}$: see previous slide
 - y : throughput (GiB/s)

Platform Model: Performance Example

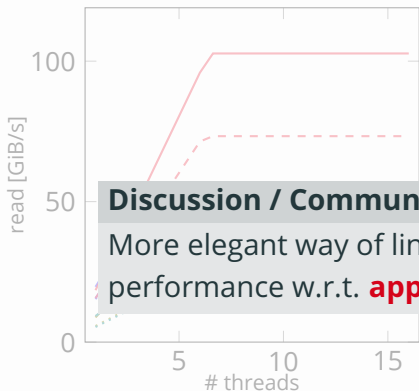


HBM / DDR / CXL x16 / CXL x8

local / intra - / cross -package

- Conventional performance model: $f(\vec{x}) = y$
 - Independent of application
 - Learnt from microbenchmarks
 - $\vec{x}$: see previous slide
 - y : throughput (GiB/s)
- Exception: non-generic, compute-bound application kernels
 - Depend on application
 - $\vec{x}$: application config, # threads

Platform Model: Performance Example



Discussion / Community Perspective

More elegant way of linking application and hardware variability / performance w.r.t. **application-specific** compute kernels?

- Conventional performance model: $f(\vec{x}) = y$
 - Independent of application
 - Learnt from microbenchmarks

application kernels

- Depend on application
- $\vec{x}$: application config, # threads

HBM / DDR / CXL x16 / CXL x8

local / intra - / cross -package



- SMAUG aims to bridge perspectives from two communities:
 - Systems: improve application performance on variable hardware
 - Variability Modelling: formally express variability; learn appropriate models



- SMAUG aims to bridge perspectives from two communities:
 - Systems: improve application performance on variable hardware
 - Variability Modelling: formally express variability; learn appropriate models
- Interaction opportunities:
 - Variability and performance modelling algorithms
 - Behaviour models and behaviour model learning
 - Comprehensive set of benchmark data; see references in paper



- SMAUG aims to bridge perspectives from two communities:
 - Systems: improve application performance on variable hardware
 - Variability Modelling: formally express variability; learn appropriate models
- Interaction opportunities:
 - Variability and performance modelling algorithms
 - Behaviour models and behaviour model learning
 - Comprehensive set of benchmark data; see references in paper

Mail: birte.friesel@uos.de

Fedi: [@derf@social.skyshaper.org](https://social.skyshaper.org/@derf)



- [Ang87] Dana Angluin. **“Learning regular sets from queries and counterexamples”**. In: Information and Computation 75.2 (1987), pp. 87–106. ISSN: 0890-5401. DOI: 10.1016/0890-5401(87)90052-6.
- [Bou+08] Patricia Bouyer et al. **“Infinite Runs in Weighted Timed Automata with Energy Constraints”**. In: Proceedings of the 6th International Conference on Formal Modeling and Analysis of Timed Systems. FORMATS’08. Springer Berlin, Heidelberg, 2008, pp. 33–47. ISBN: 978-3-540-85778-5. DOI: 10.1007/978-3-540-85778-5_4.



- [Cla+13] Andreas Classen et al. **“Featured Transition Systems: Foundations for Verifying Variability-Intensive Systems and Their Application to LTL Model Checking”**. In: IEEE Transactions on Software Engineering 39.8 (2013), pp. 1069–1089. DOI: 10.1109/TSE.2012.86.
- [FL17] Uli Fahrenberg and Axel Legay. **“Featured Weighted Automata”**. In: Proceedings of the 5th International FME Workshop on Formal Methods in Software Engineering. FormaliSE '17. IEEE, 2017, pp. 51–57. DOI: 10.1109/FormaliSE.2017.2.



- [For21] Sophie Fortz. **“LIFTS: learning featured transition systems”**. In: Proceedings of the 25th ACM International Systems and Software Product Line Conference - Volume B. SPLC '21. Leicester, United Kindom: Association for Computing Machinery, 2021, pp. 1–6. ISBN: 9781450384704. DOI: 10.1145/3461002.3473066.
- [For23] Sophie Fortz. **“Variability-aware Behavioural Learning”**. In: Proceedings of the 27th ACM International Systems and Software Product Line Conference - Volume B. SPLC '23. Tokyo, Japan: Association for Computing Machinery, 2023, pp. 11–15. ISBN: 9798400700927. DOI: 10.1145/3579028.3609007.



- [Jam+17] Pooyan Jamshidi et al. **“Transfer learning for performance modeling of configurable systems: an exploratory analysis”**. In: Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering. ASE '17. Urbana-Champaign, IL, USA: IEEE Press, 2017, pp. 497–508. ISBN: 9781538626849.
- [Ola+16] Rafael Olaechea et al. **“Long-term average cost in featured transition systems”**. In: Proceedings of the 20th International Systems and Software Product Line Conference. SPLC '16. Beijing, China: Association for Computing Machinery, 2016, pp. 109–118. ISBN: 9781450340502. DOI: 10.1145/2934466.2934473.