TOWARDS ZERO-WASTE COMPUTING

Ana-Lucia Varbanescu

CAES @ EEMCS

a.l.varbanescu@utwente.nl

GRAPH MASSIVIZER

This project has received funding from the European Union's Horizon Research and Innovation Actions under Grant Agreement № 101093202.

UNIVERSITY OF TWENTE.

TOWARDS
ZERO-WASTE COMPUTING

Ana-Lucia Varbanescu

CAES @ EEMCS

a.l.va

Work & results with Nick Breed, Quincy Bakker, Duncan Bart, Jeffrey Spaan, Jelle van Dijk.

GRAPH MASSIVIZER

This project has received funding from the European Union's Horizon Research and Innovation Actions under Grant Agreement № 101093202.

UNIVERSITY OF TWENTE.

Computing is everywhere … and it’s not free!

- Top 10 videos on YouTube* consumed as much as 600-700 EU persons per year (or about 400 North America persons)
- Training Alpha-Zero for a new game consumes as much as 100 EU persons per year
- A mid-size datacenter alone consumes as much energy as a small town
  - And that is not considering purchasing and secondary operational costs (e.g., cooling)
- In 2019 Dutch datacenters combined consumed 3-times more energy than the national railways
  - And consumption increased by 80% in 3 years
- The ICT sector is predicted to reach 21% of the global energy consumption by 2030

*https://en.wikipedia.org/wiki/List_of_most-viewed_YouTube_videos#Top_videos

Computing is everywhere … and it's not free!

- Top 10 videos on YouTube* consumed as much as 600-700 EU persons per year (or about 400 North America persons)
- Training Alpha-Zero for a new game consumes as much as 100 EU persons per year
- A mid-size datacenter alone consumes as much energy as a small town
  - And that is not considering purchasing and secondary operational costs (e.g., cooling)
- In 201
  nation
  - And

The energy consumption of computing is substantial and constantly increasing!

- The ICT sector is predicted to reach 21% of the global energy consumption by 2030

*https://en.wikipedia.org/wiki/List_of_most-viewed_YouTube_videos#Top_videos

Three types of stakeholders

Developers and users

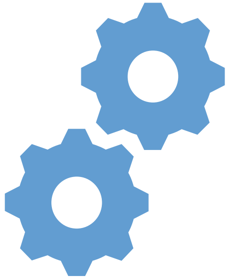
Improve the energy efficiency of their own codes, making use of algorithmic, programming, and hardware tools

Design and implement applications able to adapt to the available system resources

System operators

Ensure efficient scheduling of workloads on system resources.

Harvest energy where resources/systems are massively underutilized.



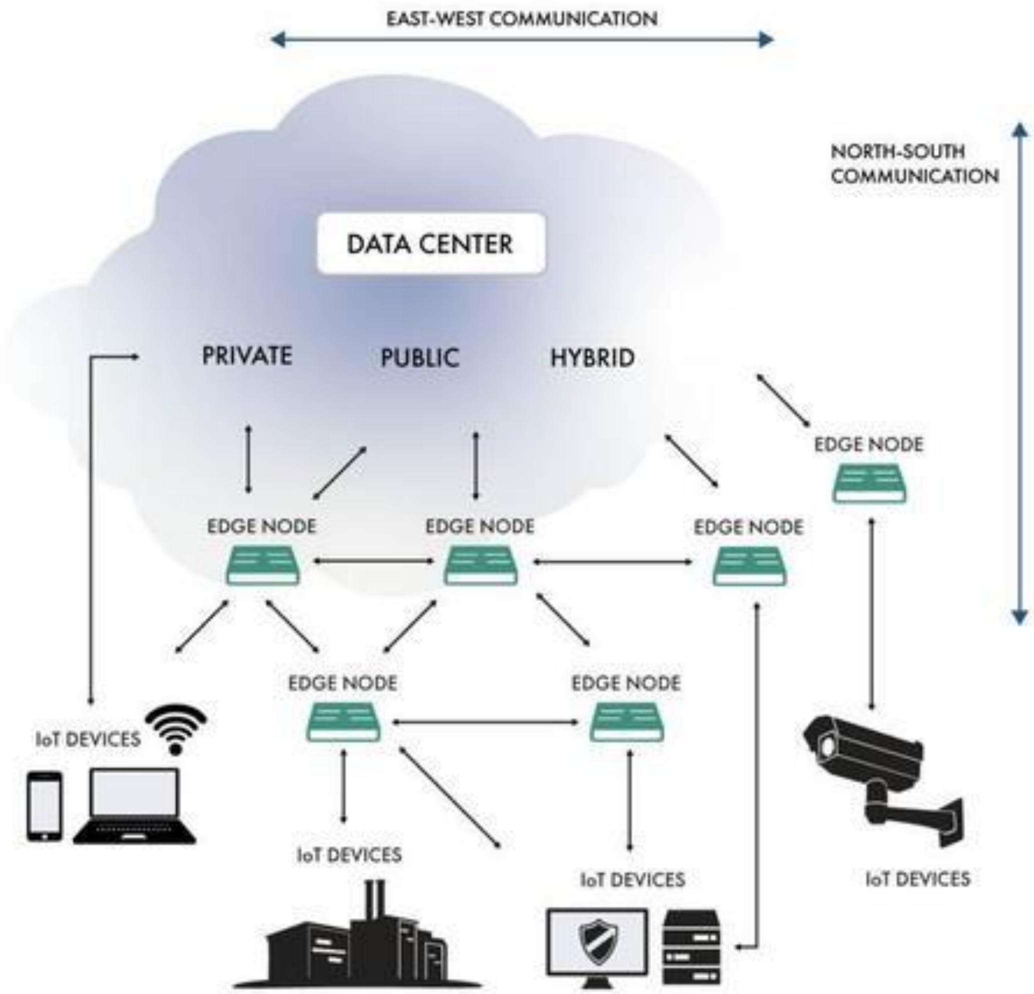
System integrators

Offer the right mix of resources for the application developers and system operators.

Include efficient hardware to enable different application mixes.

Systems

- On-premise hardware
  - Flexible, yet often limited in resources
  - Good for development
  - Limited value for production
- Supercomputers
  - Massive machinery, high-performance
  - Partially shared
  - Less flexible in terms of infra and programming
- Datacenters & Cloud computing
  - Scale-by-credit card
  - Excellent efficiency
  - Possible limitations in terms of performance (SLA)
- Computing continuum
  - New development in distributed computing
  - Unclear for scientific computing
  - Relevant for complete data analysis (sensor-to-result)



Supercomputing/Data-centers

- Supercomputing is extremely high in carbon emissions, mainly due to scale.
- **Embodied carbon*:** Indirect emissions, e.g., production, shipping, and disposal of system components.
- **Operational carbon:** Electricity, heating, cooling, etc. for the site operation.

Fugaku
0.44Exa @30MW

Frontier
1.2Exa @23MW

Aurora
2Exa? @60MW?

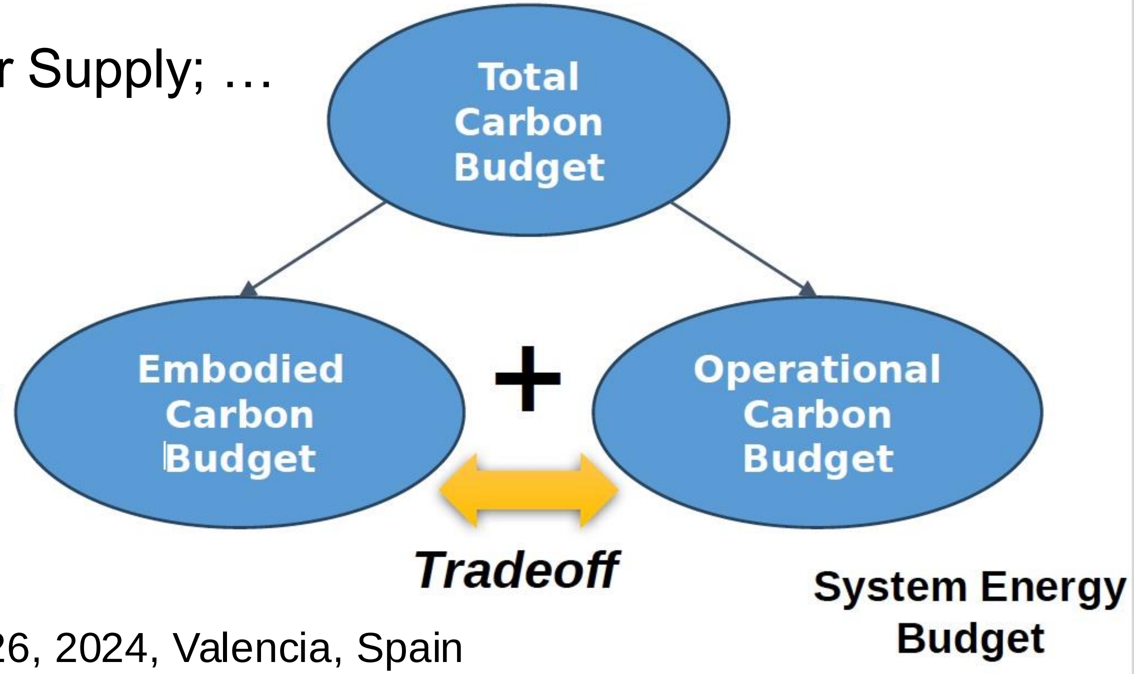
*Data and tools: https://boavizta.org/en

Sustainable acquisition

- We need new lifecycle assessment & procurement procedures

- Current Goal: Maximize Throughput (Workloads)
- Constraints:
  - Budget = Machine Cost + Electricity;
  - System Footprint/Weight; Cooling Capacity; Power Supply; …

- New Constraint: Carbon Budget

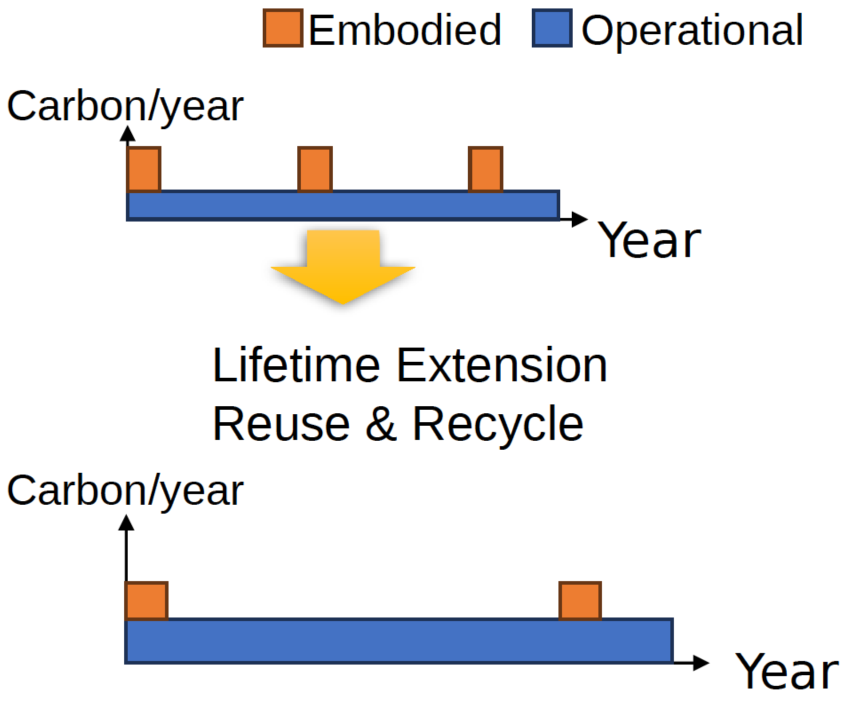


Extend lifetime

- We need to Extend Lifetime, Reuse, and Recycle

- System Lifetime: Typically 4-6 years
  - Extended lifetime => embodied carbon reduction.

- Reuse & Recycle: Reduce carbon emissions caused by disposal & production
  - Reuse: e.g., LRZ offers decommissioned machines for free.
  - Recycling: accelerators, DRAM chips, heat pipes, cooling infra, ...



Three types of stakeholders

Developers and users

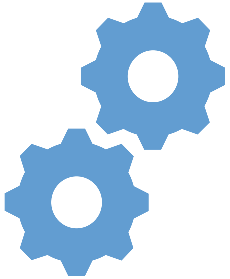
Improve the energy efficiency of their own codes, making use of algorithmic, programming, and hardware tools

Design and implement applications able to adapt to the available system resources

System operators

Ensure efficient scheduling of workloads on system resources.

Harvest energy where resources/systems are massively underutilized.



System integrators

Offer the right mix of resources for the application developers and system operators.

Include efficient hardware to enable different application mixes.

New scheduling & RM opportunities

- Efficient operation
  - Schedulers
  - Automated tools
  - Support/education for users

- Additional opportunities
  - Shared resources
  - Location shifting
  - Time/Peak shifting

Three types of stakeholders

Developers and users

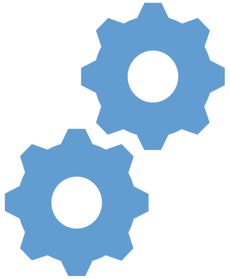
Improve the energy efficiency of their own codes, making use of algorithmic, programming, and hardware tools

Design and implement applications able to adapt to the available system resources

System operators

Ensure efficient scheduling of workloads on system resources.

Harvest energy where resources/systems are massively underutilized.



System integrators

Offer the right mix of resources for the application developers and system operators.

Include efficient hardware to enable different application mixes.

Developers & users

- Measure/Quantify
- Select the right systems
- Select the right implementation tools
- Select the right algorithms
- Tweak and tune … and iterate

Non-trivial!
But we must start somewhere …

Agenda

- ~~Stakeholders & actions~~
- Different views on performance
  - Zero-waste computing
- 2.5 case-studies
- (re)Defining systems codesign
- Take home message

"Larry, do you remember where we buried our hidden agenda?"

This project has received funding from the European Union's Horizon Research and Innovation Actions under Grant Agreement № 101093202.

Assumptions

- Modern (and future) systems are parallel and heterogeneous
  - In many dimensions
- Systems are characterized by peak performance (with various “roofs”)
- All applications want more performance
  - Applications must enable parallelism
- One Application => ***n*** algorithms => ***n*m*** implementations
  - Algorithms: characterized by complexity
  - Algorithms/implementations: characterized by arithmetic/operation intensity – ops/byte

Some relevant performance metrics*

- Speed-up: how much faster do we get with new machines, algorithms, …

```
S(workload) = Perf(Old)/Perf(New)
```

High-performance computing

- Efficiency: how efficient are we in getting performance

```
E(workload) = Perf / Resources
```

- Energy efficiency: how energy efficient are we in getting performance

```
EE(workload) = Perf / Energy
```

High-efficiency computing

- Utilization: how efficient are we utilizing our resources

```
U(resource) = Achieved / Peak
```

**please accept the naïve notation and pseudo-definitions*

Waste in computing

Unneccesary time (or energy) spent in (inefficient) computing is compute waste.

To reduce compute waste, we must focus on efficiency-to-solution

Detecting waste [1]

- We assume computing waste is a consequence of underutilized resources.

- Informally, assume:

```
P1 = performance(algorithm, workload, system1)
P0 = idealPerformance(algorithm, workload, system1)
```

- “Strict” definition:

```
if (P1 < P0) => waste in P1
```

- “Relaxed” definition:

```
if (P0 - P1 > T ) => waste in P1
    with T = threshold for performance loss
```

Ideal performance is non-trivial to quantify.

**performance is not necessarily runtime.*

Detecting waste [2]

- We assume computing waste is a consequence of underutilized resources.

- Informally, assume:

```
system1 > system2
P1 = performance(algorithm, workload, system1)
P2 = performance(algorithm, workload, system2)
```

- “Strict” definition:

```
if (P1 == P2) => waste in P1
```

- “Relaxed” definition:

```
if ( abs (P1 - P2) > T ) => waste in P1
    with T = threshold for performance loss
```

Challenges in both efficiency quantification and improvement.

**performance is not necessarily runtime.*

Reducing waste in computing

Raise awareness

- Monitor (energy) efficiency
- Quantify waste

Improve efficiency

- Improve applications for the **systems at hand**
  - Make applications more efficient
  - Make applications share systems
- Improve systems for the **applications at hand**
- **Co-design** applications and systems

- Analysis
- Modeling
- Code++ optimization
- Efficient scheduling and resource sharing
- Application-centric system design
- ??

Systems at hand & efficiency knobs

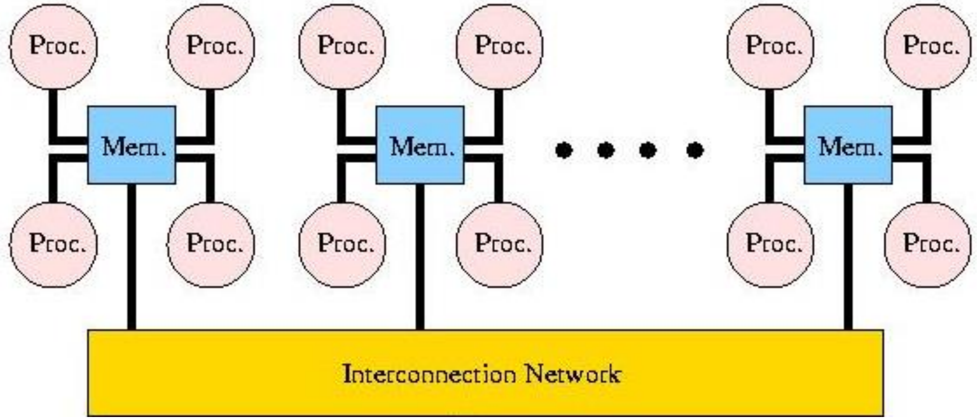
Cores, power, energy…

- Multi-core CPU
  - Multi-core energy consumption != N * energy/core
    - Complex architecture, different clocks, shared resources
  - Various ways to implements DVFS + power reduction techniques
  - Non-trivial correlation with performance

- GPU
  - Power is significantly impacted by the type of workload and occupancy
  - Always check the power cap, too!

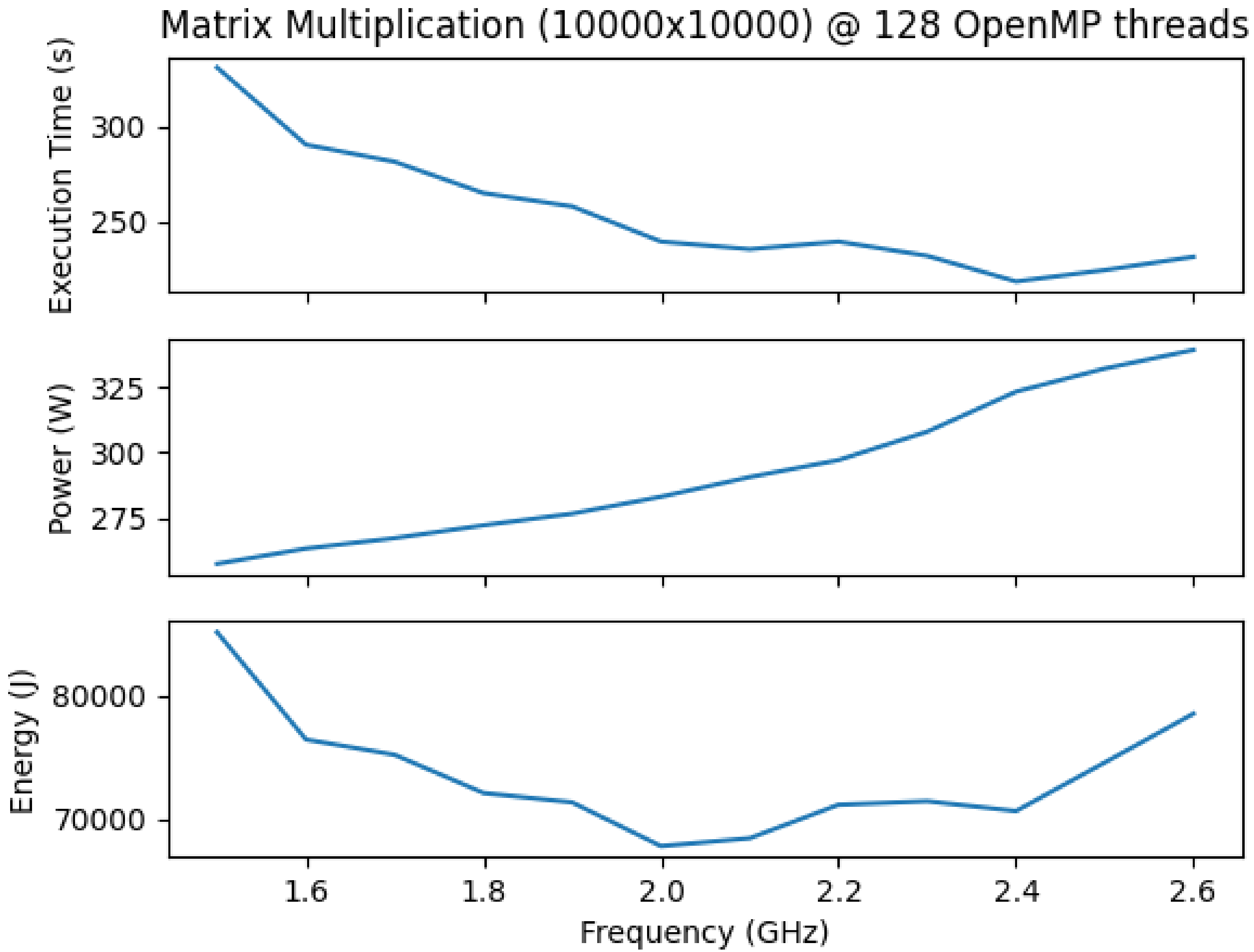
- Heterogeneous & multi-node computing
  - Sum of energy by components
  - Networking energy lacking

Hybrid



CPU example

- AMD EPYC CPU
- Running SGEMM
  - Different frequencies

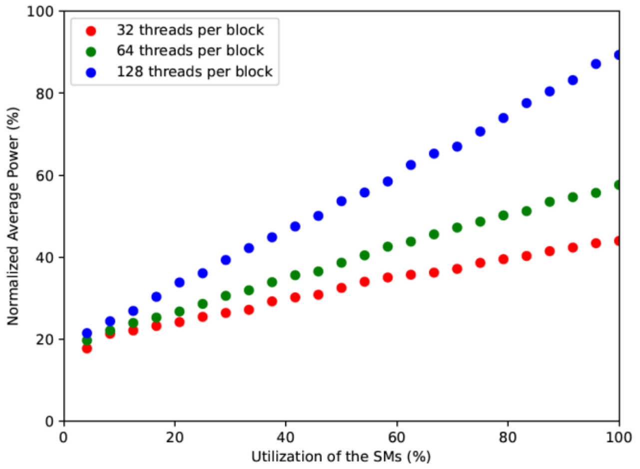


Courtesy of Benjamin Czaja, SURF, NL

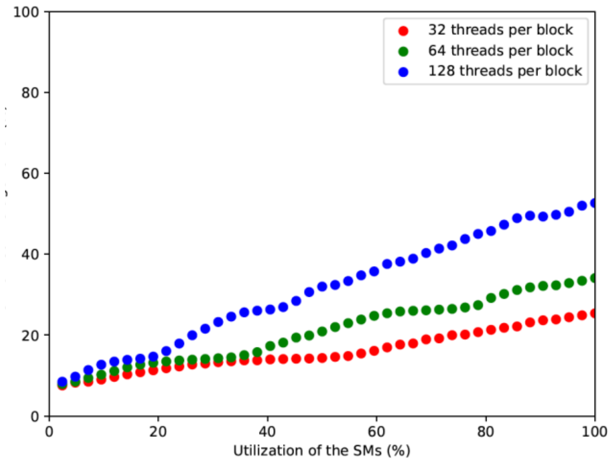
GPU example

GPU	SMs	Cores/SM	Total Cores	Max Power [W]	Idle Power [W]
A4000	48	128	6144	140	39.5
A6000	84	128	10572	300	71.5
A2	10	128	1280	60	18.1
A100	108	64	6912	250	37.9

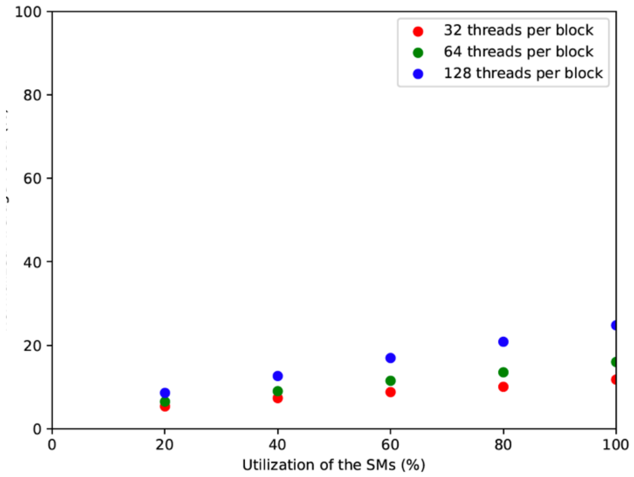
A4000



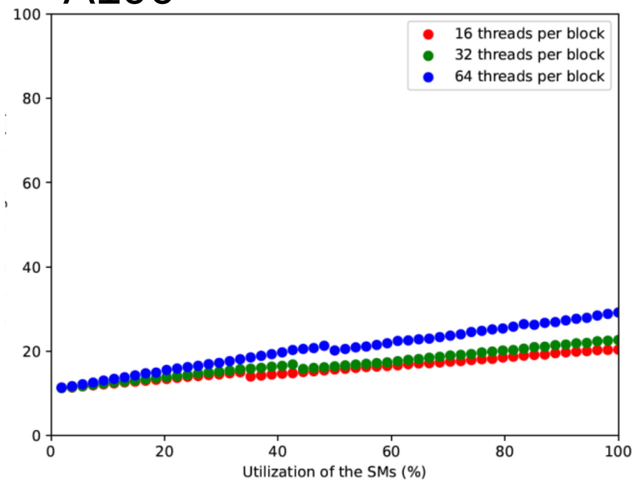
A6000



A2

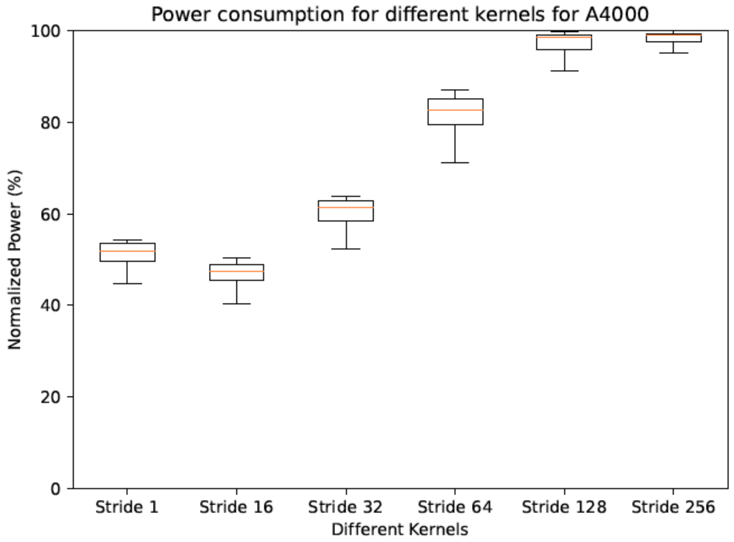


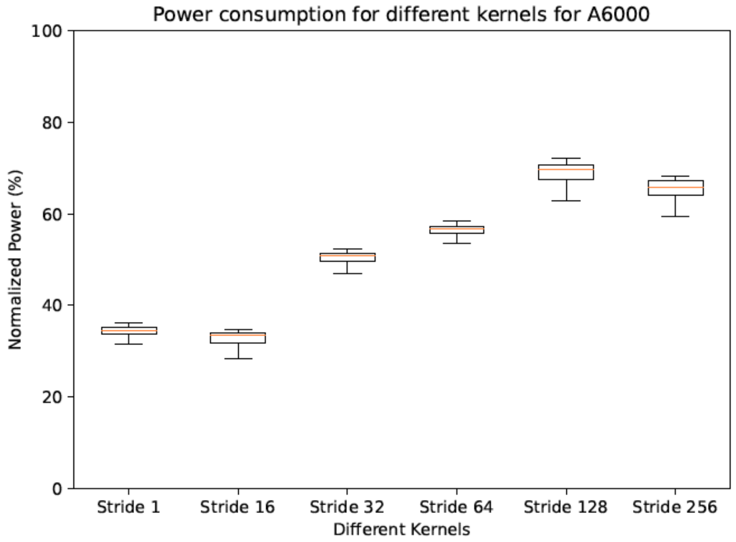
A100

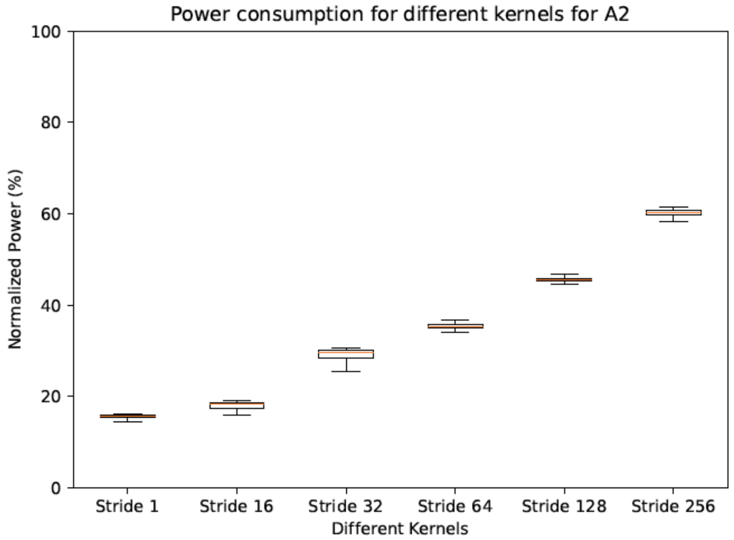


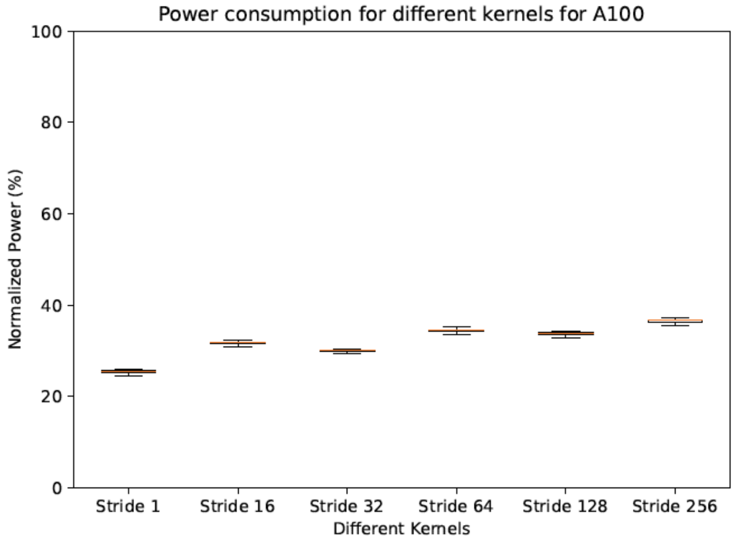
GPU example (cont'd)

- Caching patterns make a significant difference
- Compute vs memory intensive – mem consumes more.
- Memory coalescing, negligible
- Data types & instruction mix show some differences









Performance vs. energy

- Low performance ➔ waste in computing
  - We power resources that are not needed

- High performance ➔ faster execution ➔ less energy consumed
  - Max energy efficiency ⬄ max performance (i.e., lowest runtime…)
    - Strong assumption that power is constant

In the context of multi-core and heterogeneous systems, minimizing execution time might not guarantee lowest energy consumption.

Auto-tuning, anyone?

Goel and McKee – "A Methodology for Modeling Dynamic and Static Power Consumption for Multicore Processors"

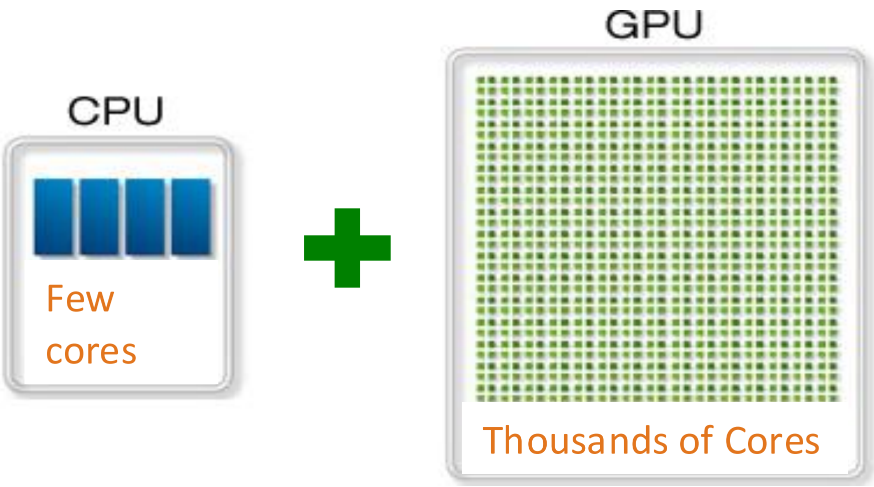
Case study #1: Heterogeneous systems

https://gitlab.qub1.com/vrije-universiteit/master-project/thesis

https://scripties.uba.uva.nl/search?id=record_27683

Heterogeneous computing

- A heterogeneous platform = a CPU + a GPU (the starting point)
- An application workload = an application + its input dataset
- Workload partitioning = workload distribution among the processing units of a heterogeneous system



Heterogeneous computing for all??

- Model-based load-balancing for heterogeneous computing
  - Analytical model
  - Empirical calibration
  - Embedded in the Glinda framework

- Challenging programming
  - Leverages performance portable programming models

- Maximizes performance *and/or* resource utilization => minimizes waste
  - Uses *all* types of resources in the system

- Driven by performance
  - Could/should be extended for energy efficiency

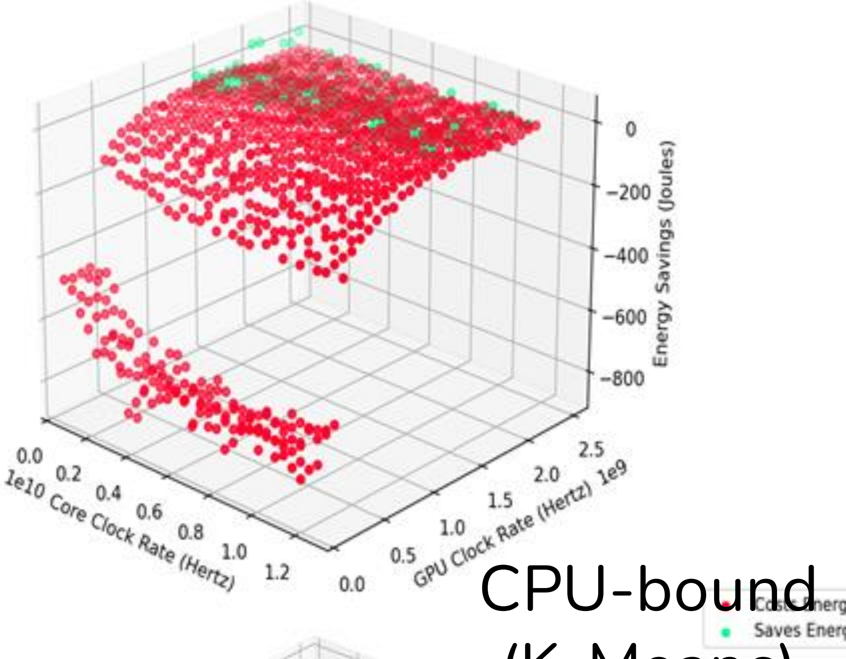
*Jie Shen et al., IEEE TPDS. 2015
"Workload partitioning for accelerating applications on heterogeneous platforms"

Energy improvements

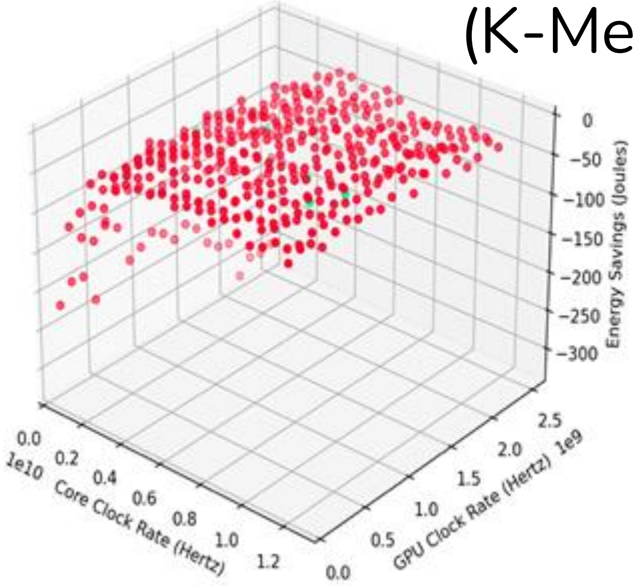
- Basic assumptions
  - Tasks run on different processors
  - Idle processors waste energy
  - Higher/lower operating frequencies
    - => more/less power respectively
    - => reduce or increase runtime respectively

- Opportunities
  - Dynamic Voltage and **Frequency** Scaling (DVFS)
  - Reducing operating frequencies in idle states to save energy
    - No active task => no runtime increase
  - Increasing operating frequencies in busy states to save energy
    - Lower runtime => less time to consume energy

GPU-bound
(Matrix Multiply)

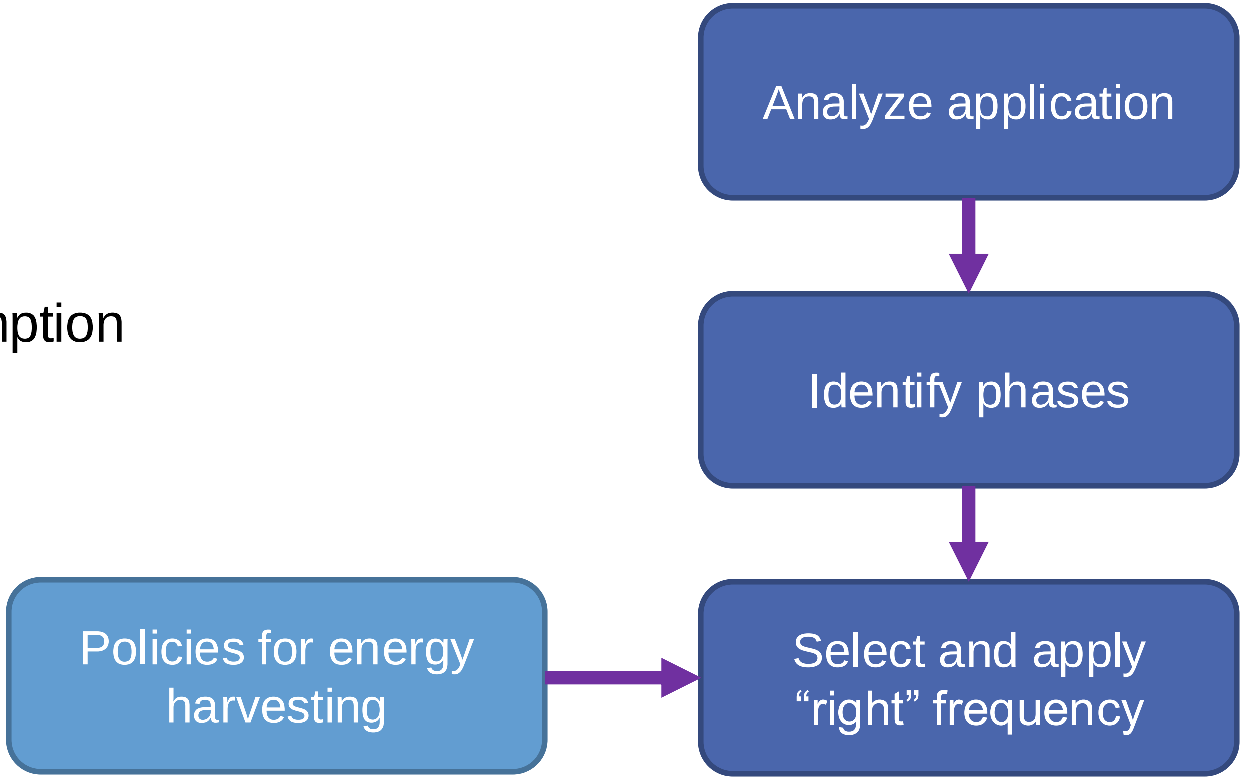


CPU-bound
(K-Means)



Approach

- **Framework** to monitor and improve the energy consumption of heterogeneous applications
  - Analyze application at runtime
    - Use live execution data
  - Determine application states
    - CPU/GPU-utilization patterns
  - Apply DVFS for this phases
    - Observe energy changes
  - Design policies to maximize energy consumption
    - What, when, and how to apply DVFS



Empirical analysis

- Workload: 10 different applications from different benchmarking suites
- System: Geforce GTX 960 GPU and an AMD Ryzen 7 3700x CPU.
- Metrics of interest: runtime and energy consumption

- Reference implementation = “do nothing”
  - Gain and/or loss against reference

- Five policies :
  - Maximum Frequency
  - System
  - MinMax
  - Ranked MinMax
  - Scaled MinMax

Results

Applications	No Action		MinMax		System		Maximum frequency		Ranked MinMax		Scaled MinMax	
	Policy											
	Energy	Time	Energy	Time	Energy	Time	Energy	Time	Energy	Time	Energy	Time
BFS	5248.7 J	60.5 s	6499.7 J (23.8%)	70.6 s (16.7%)	5669.3 J (8.0%)	69.7 s (15.2%)	6276.2 J (19.6%)	60.2 s (-0.5%)	5294.3 J (0.9%)	61.2 s (1.2%)	5496.3 J (4.7%)	70.8 s (17.0%)
Myocyte												
LavaMD	7454.3 J	52.1 s	6962.4 J (-6.6%)	52.6 s (1.0%)	7024.6 J (-5.8%)	52.3 s (0.4%)	7473.5 J (0.3%)	51.0 s (-2.1%)	6951.1 J (-6.8%)	52.9 s (1.5%)	7125.0 J (-4.4%)	53.8 s (3.3%)
NW	6103.3 J	64.9 s	6465.5 J (5.9%)	77.0 s (18.6%)	7132.7 J (16.9%)	74.1 s (14.2%)	7787.6 J (27.6%)	70.4 s (8.5%)	5619.0 J (-7.9%)	78.5 s (21.0%)	5635.6 J (-7.7%)	82.5 s (27.1%)
Particlefilter-float	8540.8	89.5 s	9245.1 J (8.2%)	99.6 s (11.3%)	10028.8 J (17.4%)	96.9 s (8.3%)	10301.2 J (20.6%)	91.5 s (2.2%)	7666.4 J (-10.2%)	102.8 s (14.8%)	7578.4 J (-11.3%)	107.6 s (20.2%)
Kmeans	5729.4 J	66.2 s	6248.0 J (9.1%)	77.0 s (16.3%)	6303.4 J (10.0%)	74.4 s (12.4%)	6633.3 J (15.8%)	66.5 s (0.5%)	5514.4 J (-3.8%)	68.9 s (4.1%)	5932.2 J (3.5%)	77.9 s (17.7%)
Bandwidth	6337.7 J	50.4 s	5957.7 J (-6.0%)	54.0 s (7.1%)	6128.0 J (-3.3%)	52.3 s (3.8%)	6165.4 J (-2.7%)	51.0 s (1.2%)	6029.5 J (-4.9%)	53.5 s (6.2%)	6004.9 J (-5.3%)	54.7 s (8.5%)
UnifiedMemoryPerf	33188.3 J	266.1 s	28612.8 J (-13.7%)	263.1 s (-1.1%)	32491.1 J (-2.1%)	257.5 s (-3.2%)	34542.5 J (4.1%)	258.4 s (-2.9%)	27956.7 J (-15.8%)	262.5 s (-1.4%)	27810.9 J (-16.2%)	258.6 s (-2.8%)
matrixMul	9295.6 J	66.6 s	10442.3 J (12.3%)	67.6 s (1.5%)	10962.8 J (17.9%)	67.0 s (0.6%)	10086.7 J (8.5%)	66.5 s (-0.2%)	10913.3 J (17.4%)	67.5 s (1.4%)	10264.3 J (10.4%)	68.0 s (2.1%)
Jacobi unoptimized	10980.4 J	118.1 s	7802.1 J (-28.9%)	124.6 s (5.5%)	8192.6 J (-25.4%)	128.0 s (8.4%)	8039.1 J (-26.8%)	109.0 s (-7.7%)	8958.9 J (-18.4%)	109.3 s (-7.5%)	8440.3 J (-23.1%)	124.8 s (5.7%)
Jacobi optimized	7697.2 J	95.3 s	5467.1 J (-29.0%)	101.9 s (6.9%)	5280.8 J (-31.4%)	101.4 s (6.4%)	5021.9 J (-34.8%)	85.8 s (-10.0%)	6090.9 J (-20.9%)	86.6 s (-9.1%)	5400.4 J (-29.8%)	102.1 s (7.1%)

Results

Applications	Best Policy					
	Single Core			Multi Core		
	Name	Energy	Time	Name	Energy	Time
BFS	Scaled MinMax	-0.5%	0.2%	Ranked MinMax	0.9%	1.2%
LavaMD	Maximum Frequency	-0.7%	-0.1%	MinMax	-6.6%	1.0%
NW	Ranked MinMax	4.8%	4.4%	Ranked MinMax	-7.9%	21.0%
Particlefilter-float	Ranked MinMax	-0.0	1.5%	Ranked MinMax *	-10.2%	14.8%
Kmeans	Ranked MinMax	3.7%	0.6%	Ranked MinMax	-3.8%	4.1%
Bandwidth	Maximum Frequency	-2.3%	0.1%	Maximum* Frequency	-2.7%	1.2%
UnifiedMemoryPerf	MinMax	-1.5%	-3.8%	Scaled MinMax	-16.2%	-2.8%
matrixMul	Maximum Frequency	3.5%	-0.0%	Maximum Frequency	8.5%	-0.2%
Jacobi unoptimized	MinMax	-3.5%	-7.4%	Maximum Frequency	-26.8%	-7.7%
Jacobi optimized	MinMax	-2.7%	-9.4%	Maximum Frequency	-34.8%	-10.0%

Lessons learned

- Heterogeneous computing => high performance, high energy consumption
- Energy harvesting can work
  - Depends a lot on the implementation
- There is a broader question: how can we explore trade-offs between energy and performance ?
  - Harvesting = how to keep performance fixed
  - Energy budgets = how to maximize performance

Git repository:
https://gitlab.qub1.com/vrije-universiteit/master-project/energymanager
Thesis:
https://gitlab.qub1.com/vrije-universiteit/master-project/thesis

Lessons learned

- Heterogeneous computing => high performance, high energy consumption
- Energy harvesting can work
  - Depends a lot on the implementation
- There is a broader question: how can we explore trade-offs between energy and performance ?
  - Harvesting = how to keep performance fixed

- Heterogeneous systems require heterogeneous applications
- Not using the CPU/GPU is by definition wasteful

Reduced waste by frequency scaling.

https://gitlab.qub1.com/vrije-universiteit/master-project/energymanager

Thesis: https://gitlab.qub1.com/vrije-universiteit/master-project/thesis

Jeffrey Spaan

Case study #2: Shrinking the platform

Possible workflow to identify waste

1. Pick a workload
2. Pick a baseline platform
3. **Reduce** resources
4. **Measure** performance
5. **Compare** performance

No difference or better performance? Found waste and/or a better system.

The devil is in the details.

How to reduce resources? **How** to measure performance? **How** to compare?

~~Measuring~~ Predicting performance

- Benchmarking

Impossible

- Co-location

Difficult to setup

    - Simultaneous execution with a (specific) resource-consuming application

- Partitioning

Not available on many systems

    - Partitions with isolated GPU resources

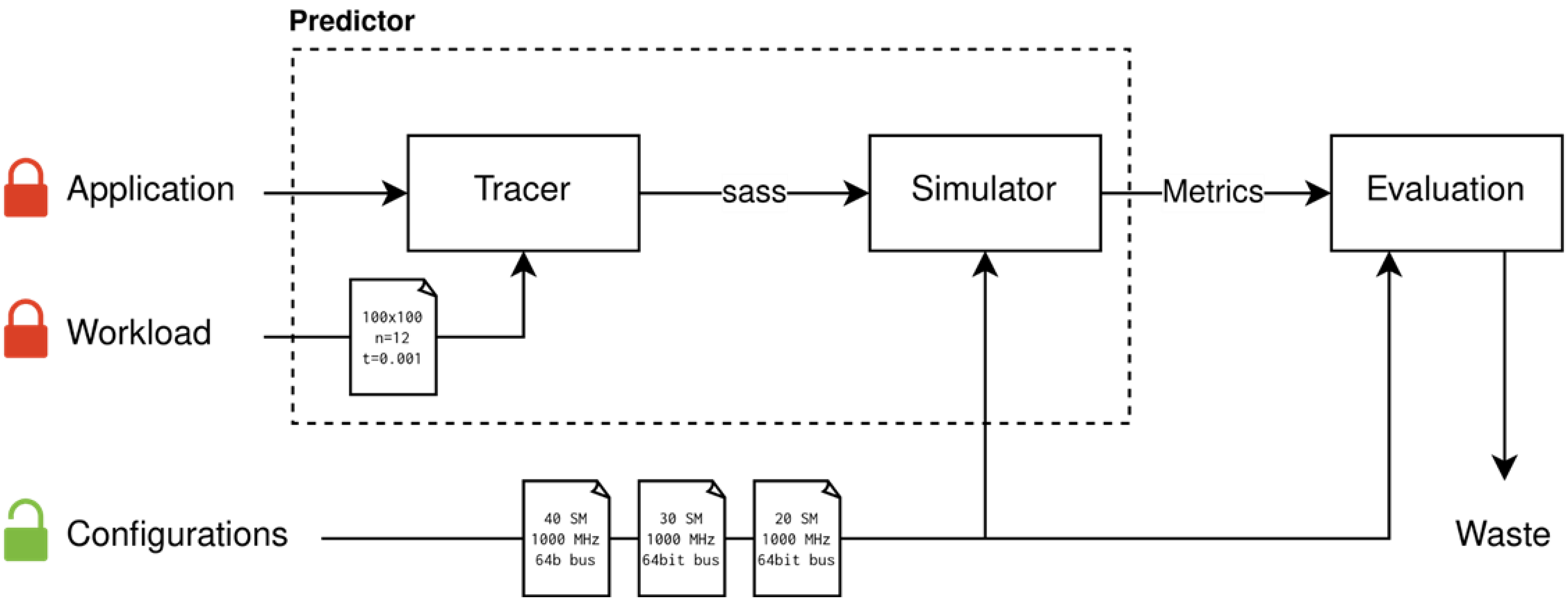
- Analytical modelling
- Statistical modelling

Not sufficiently accurate

- Simulation

best option (currently)

Proposed workflow



Experimental setup

Applications:

- 5 Rodinia kernels:
    - **Compute**-bound: hotspot, k-means (2)
    - **Memory**-bound: k-means (1), backpropagation (1), backpropagation (2)

Systems:

- Baseline: RTX 2060 Super
- Variables:
    - **SMs**: 25, 30, …., 40
    - **Core clock**: 1000, 1150, …., 1900
    - **Memory clock**: 800, 1250, …, 3500

Simulation run-time ≈ 24-40 hours

Simulated with:

AccelSim

Ask me more!

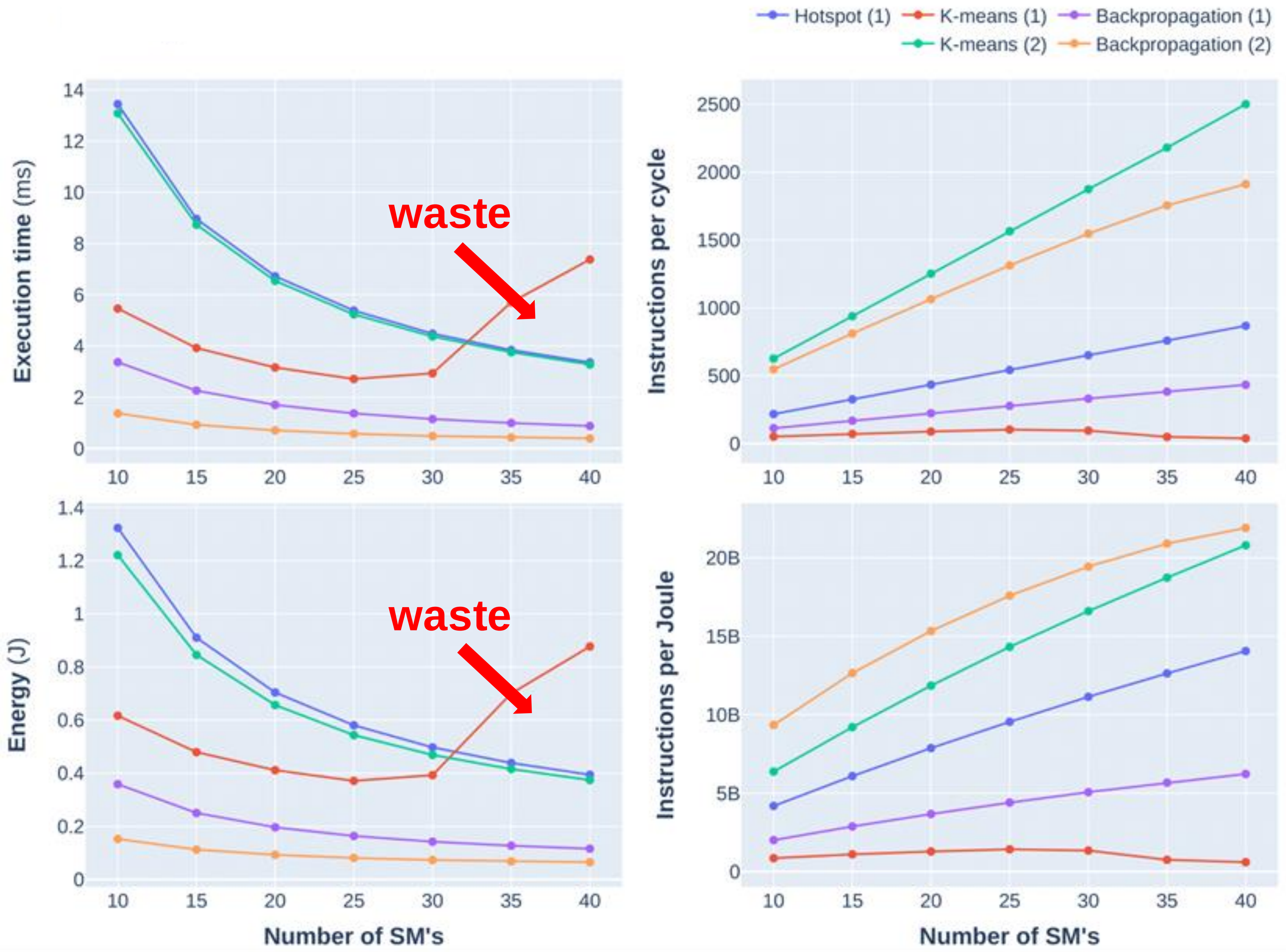
https://github.com/romnn/gpucachesim

Varying SMs

Compute-bound:
- Hotspot
- K-means (2)

Memory-bound:
- K-means (1)
- Backprop (1)
- Backprop (2)

More resources ≠ better performance



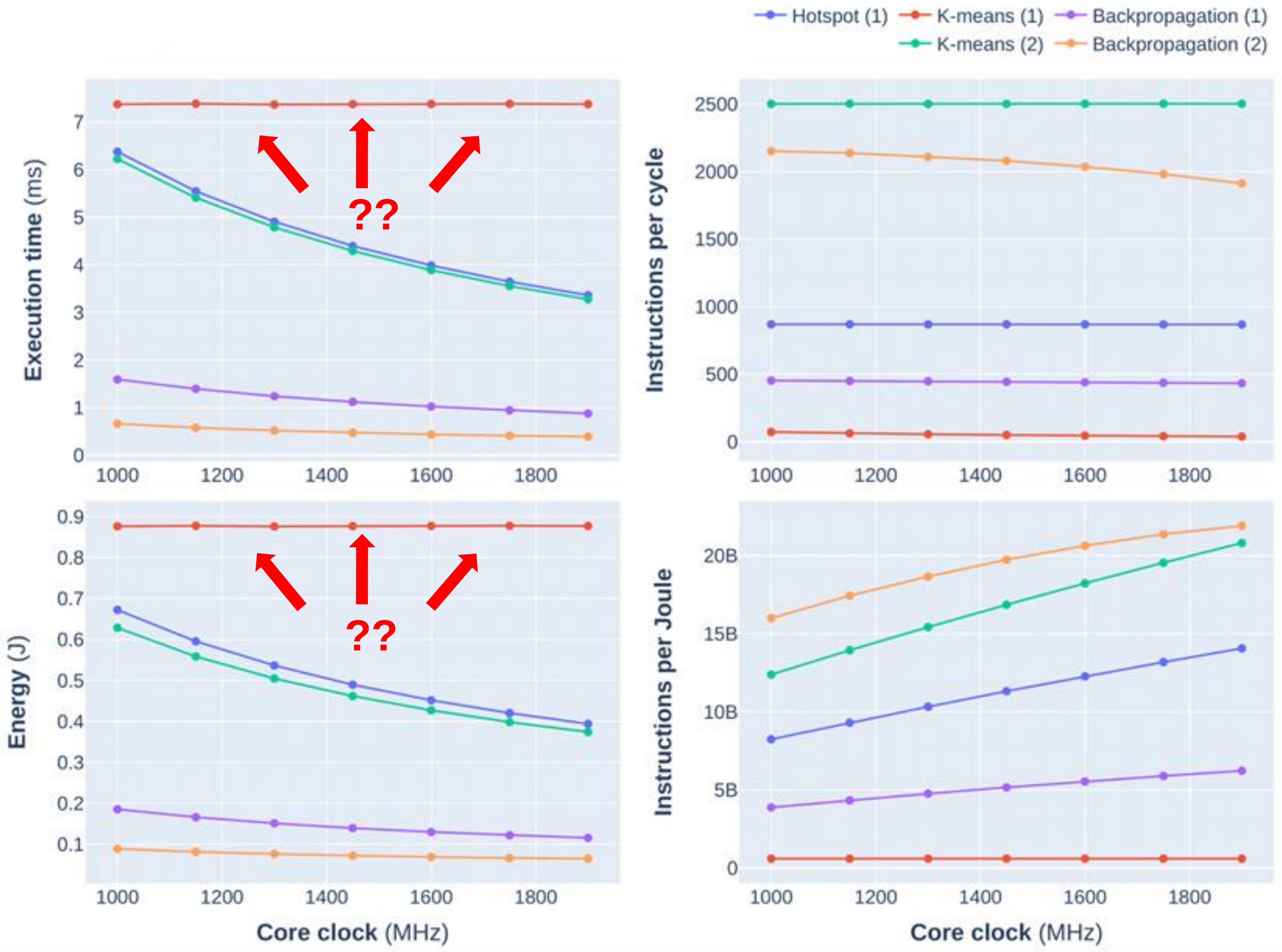
Core clock

Compute-bound:

- Hotspot
- K-means (2)

Memory-bound:

- K-means (1)
- Backprop (1)
- Backprop (2)



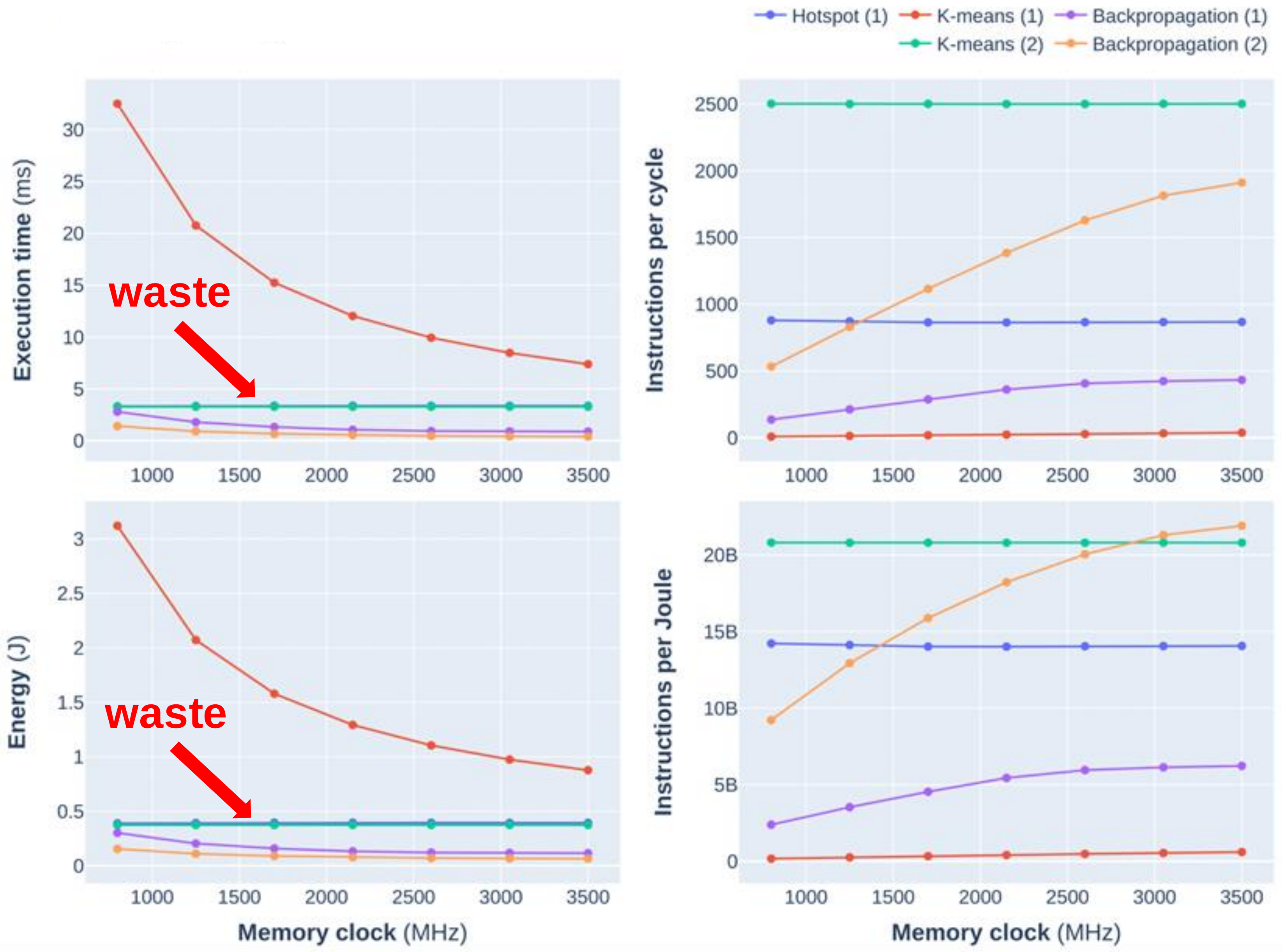
Memory clock

Compute-bound:

- Hotspot
- K-means (2)

Memory-bound:

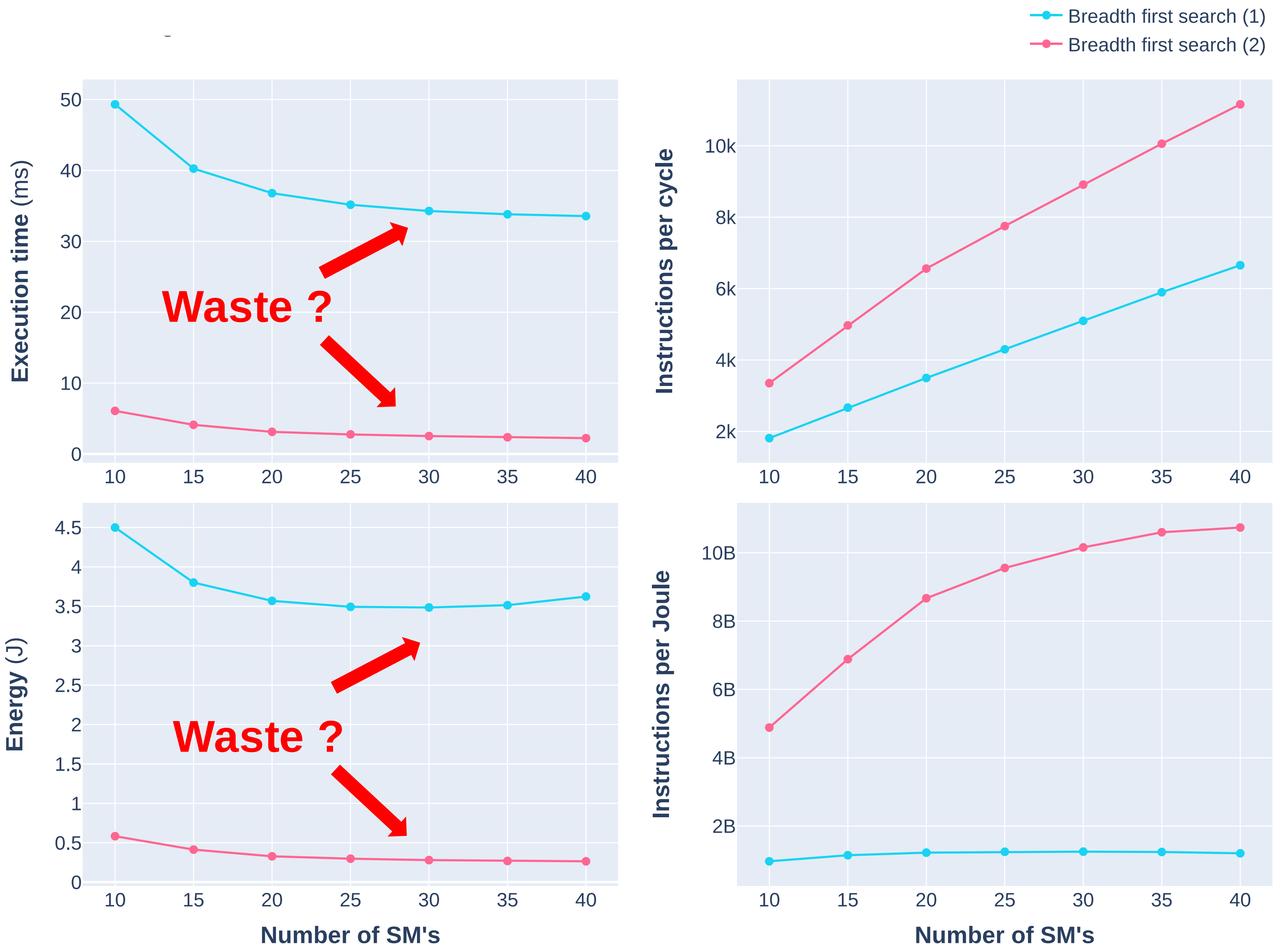
- K-means (1)
- Backprop (1)
- Backprop (2)



SMs: BFS

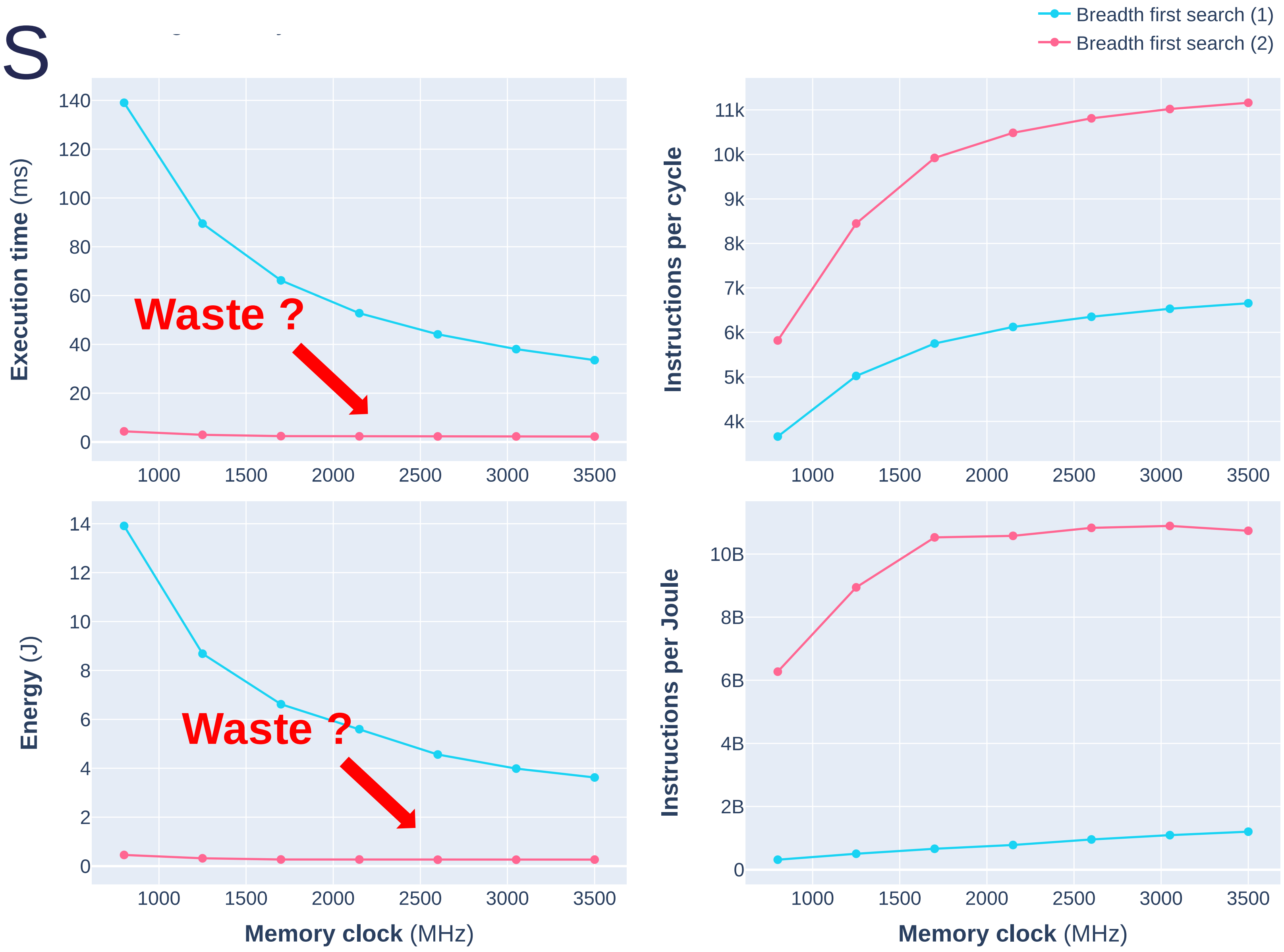
BFS is memory bound.

Is the strict definition reasonable? Should we use the relaxed definition?



Memory clock: BFS

As BFS is memory bound, we do expect to see performance gain when the memory clock speed increases.



Lessons learned

- We demonstrated **waste at resource-level can be significant**

- We demonstrated it is possible* (in simulation) to update platforms

- New opportunities for …
  - Partitioning
  - Scheduling
  - Runtime systems

- Waste can (/should?) be investigated per resource.
- Difficult to model, trivial (but sloooow) to simulate.

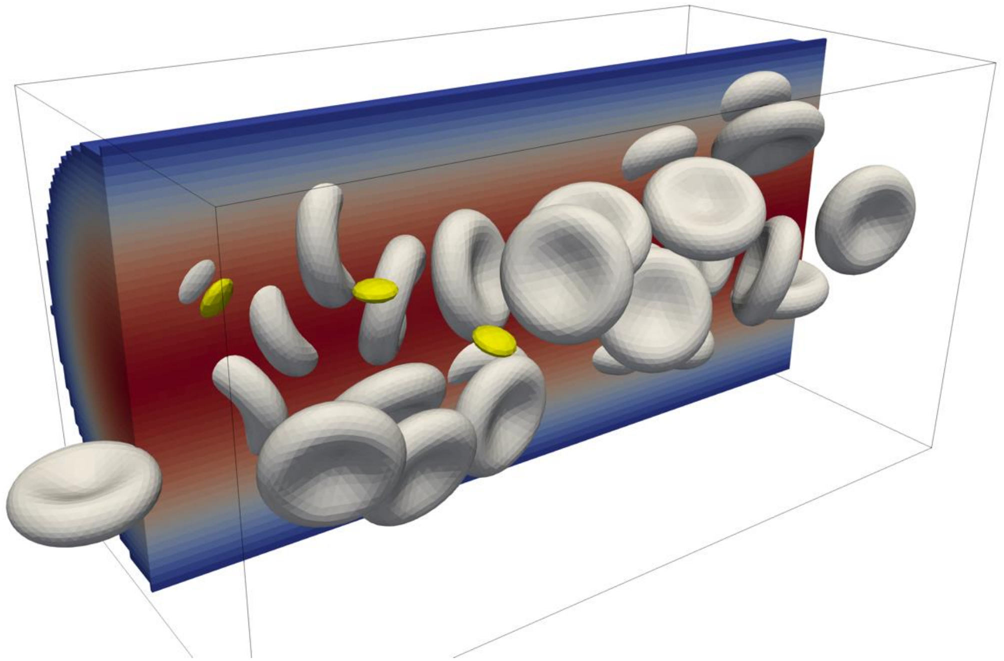
Reconfiguring the system can reduce compute waste.

Case study #3: Embracing load imbalance

HemoCell – coupled simulation

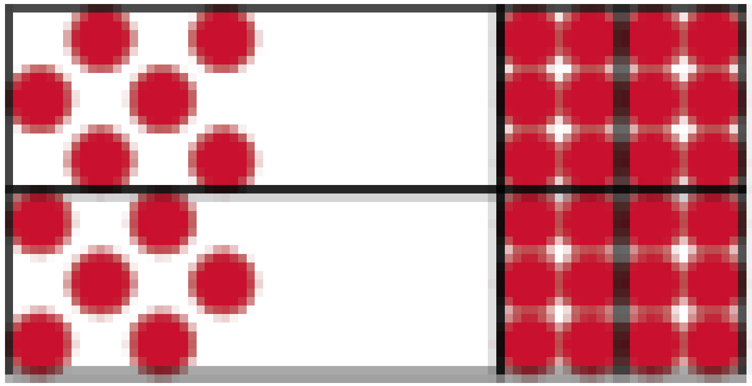
- Simulated blood flow using …
  - Fluid simulation
  - Particle simulation
- Aims for high-performance
  - Using distributed processing & MPI

- Observations:
  - Load imbalance is difficult to fix
  - … but can be easier to detect
  - Adapting the frequency of nodes to their load can lead to energy savings!
- Technique: DVFS per node.

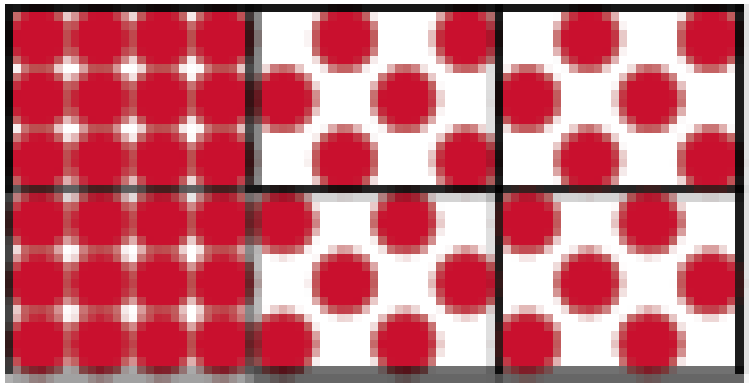


https://hemocell.eu/

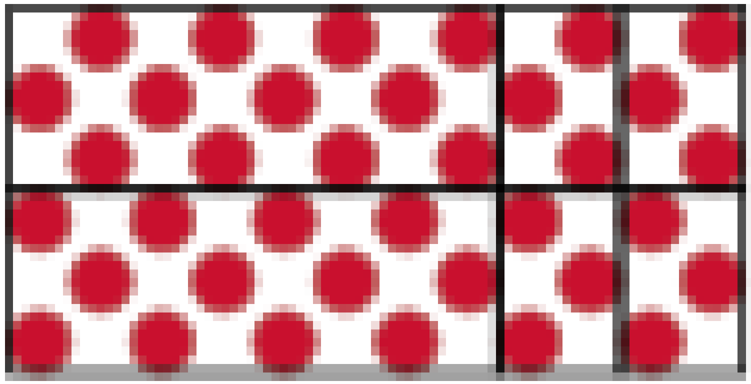
Load imbalance



C1: Fluid imbalance.



C2: Particle Imbalance.



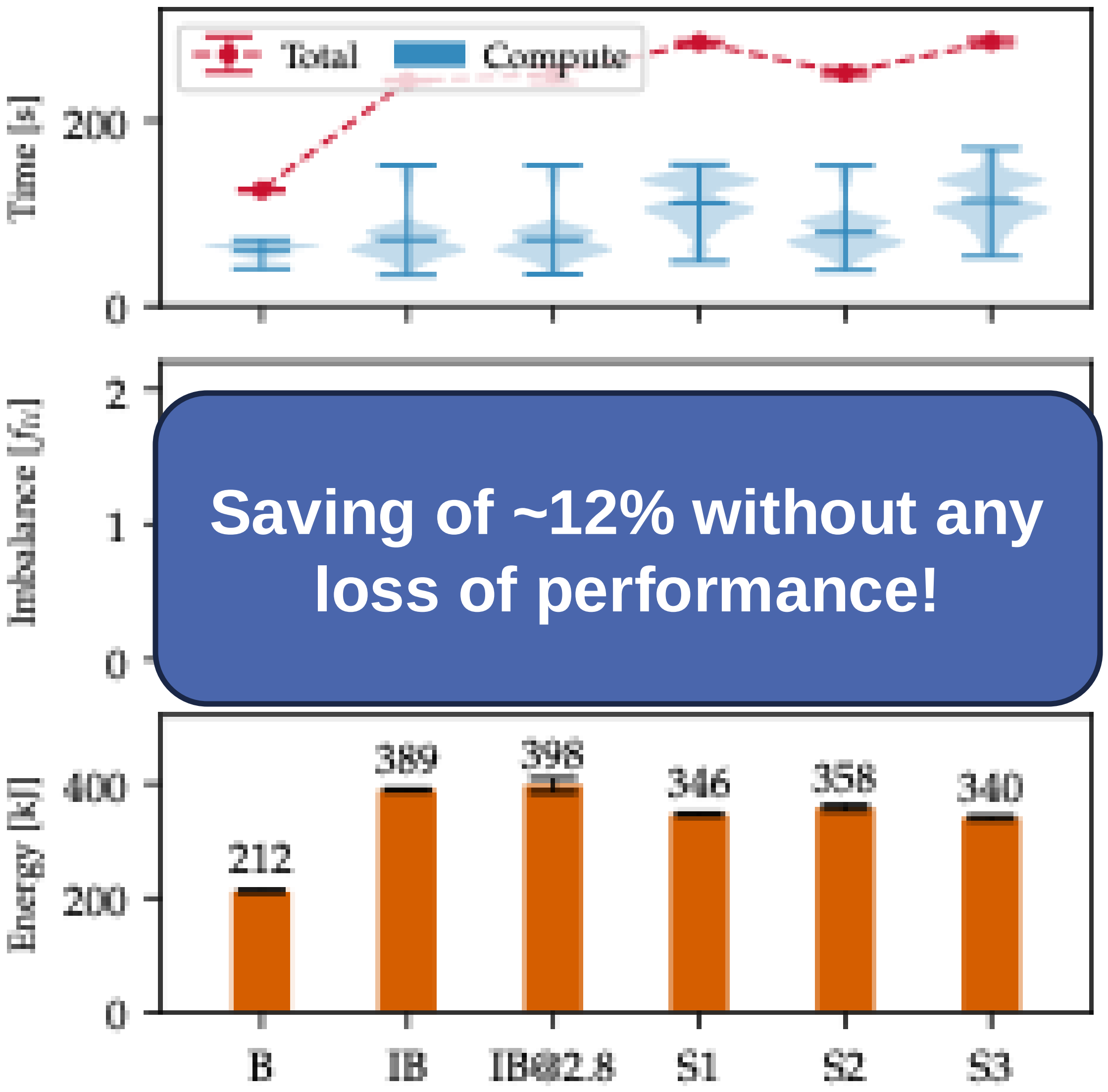
C3: Fluid and particle imbalance

Empirical analysis

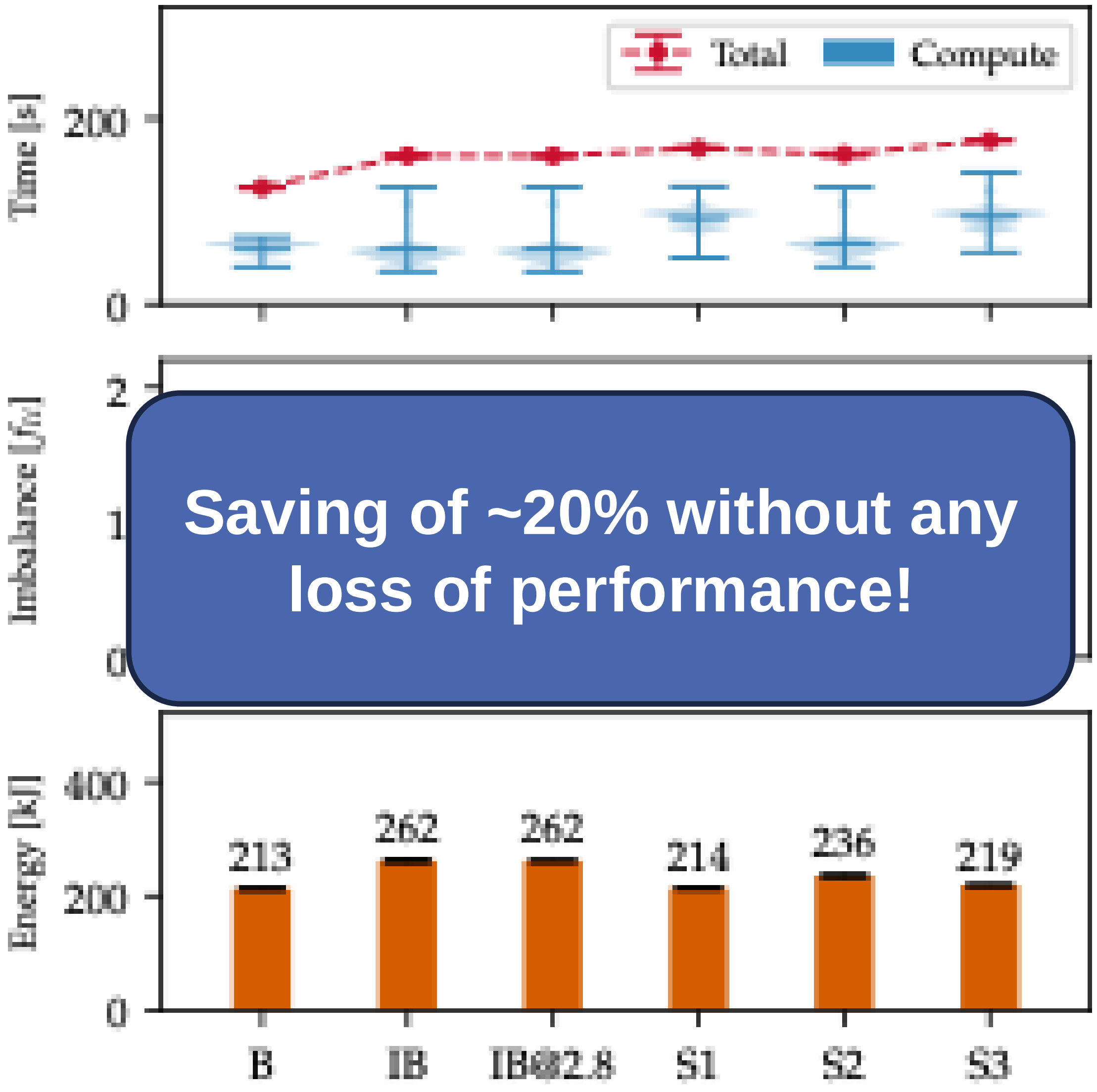
- 16 node experiment (DAS6 machine)
  - 1,2 have higher workloads
  - 3,6 have “normal” workloads
- 3 DVFS strategies
  - Reduce the frequency of the underutilized nodes

	Nodes	
Strategy	1-2	3-16
S1	2.8 GHz	1.5 GHz
S2	2.8 GHz	2.4 GHz
S3	2.4 GHz	1.5 GHz

Energy savings [C1, C2]

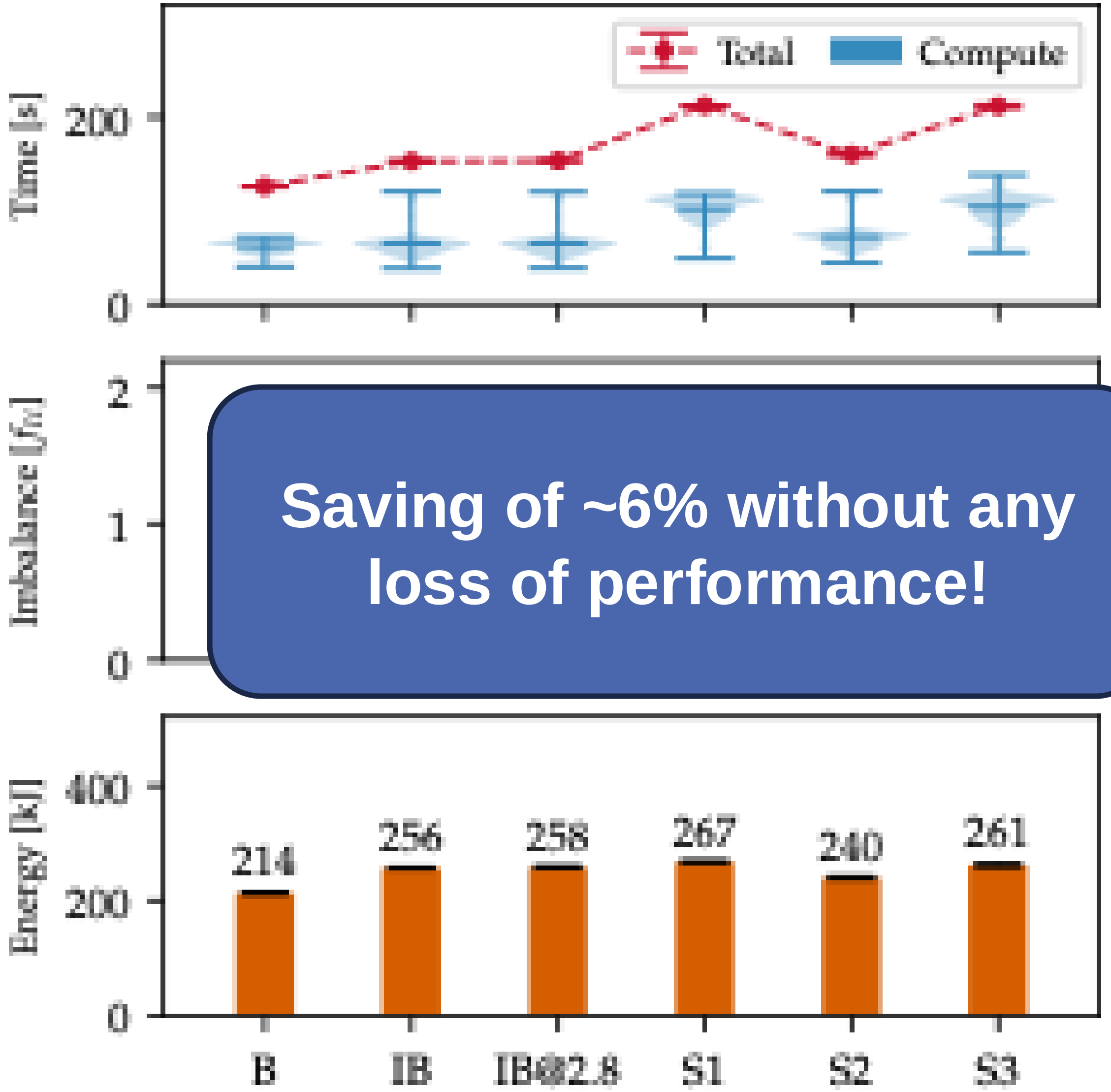


Saving of ~12% without any loss of performance!

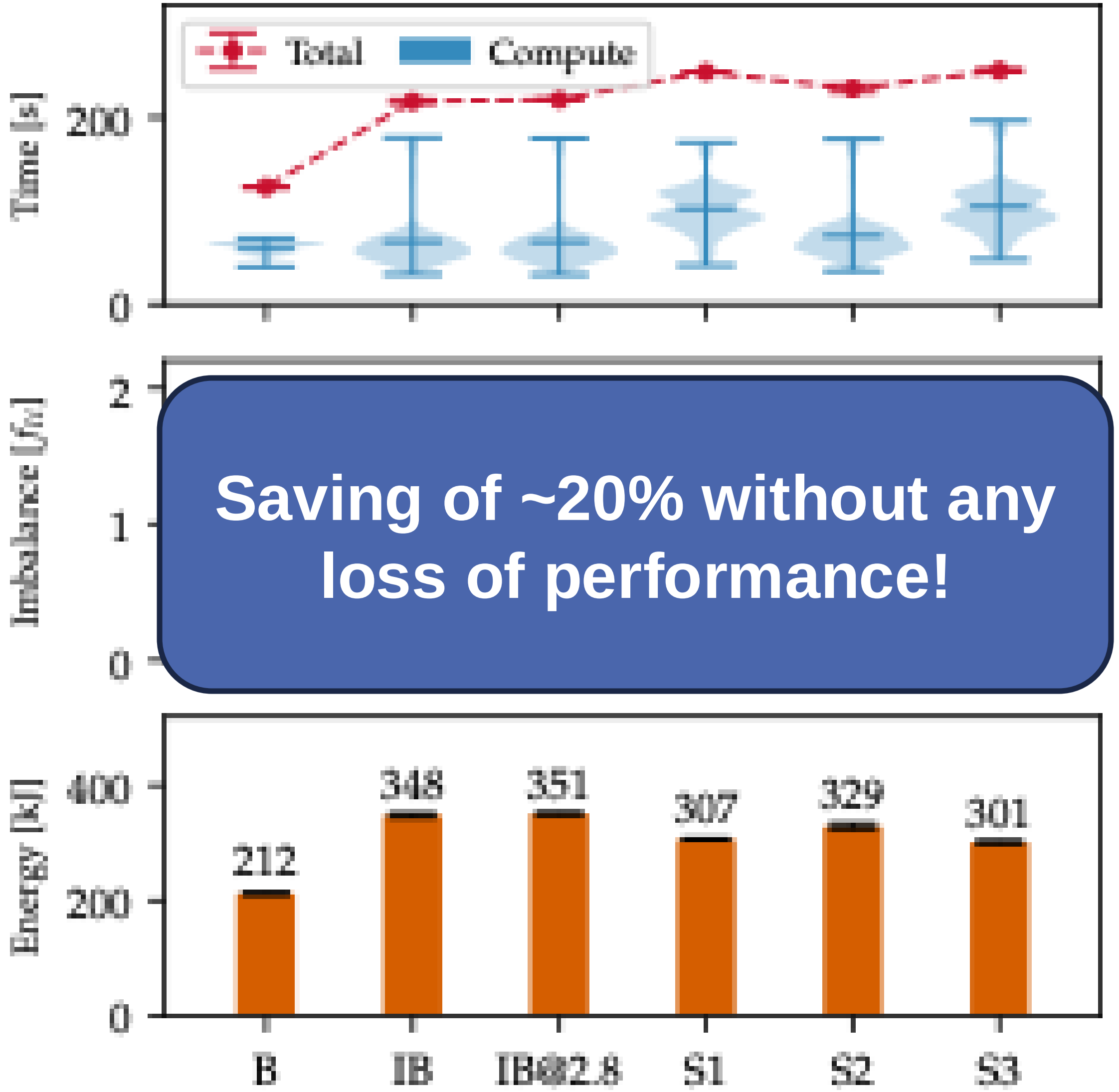


Saving of ~20% without any loss of performance!

Energy savings [C3]



Saving of ~6% without any loss of performance!



Saving of ~20% without any loss of performance!

Lessons learned

- We demonstrated **waste due to load imbalance**

- We demonstrated it is possible* to make use of load imbalance to reduce energy consumption

- New opportunities for …
  - Runtime systems
  - Scheduling

- Key challenge: can we automate the process?
  - Detect load imbalance
  - Select correct frequencies
  - Apply DVFS

Lessons learned

- We demonstrated **waste due to load imbalance**
- We demonstrated it is possible* to make use of load imbalance to reduce energy consumption
- New opportunities for …
  - Runtime systems
  - Scheduling

- Waste due to load imbalance does happen
- Difficult to re-balance, easier to save energy

DVFS remains is a valid approach, when permitted.

Can we co-design?

Co-designing systems and applications

Co-design

“Co-design is the process of involving multiple stakeholders in the design and development of products, services, or systems with the goal of creating solutions that are more relevant, effective, and satisfying to the people who will use them.” [1]

“Hardware/Software Codesign is the design of cooperating hardware components and software components in a single design effort.” [2]

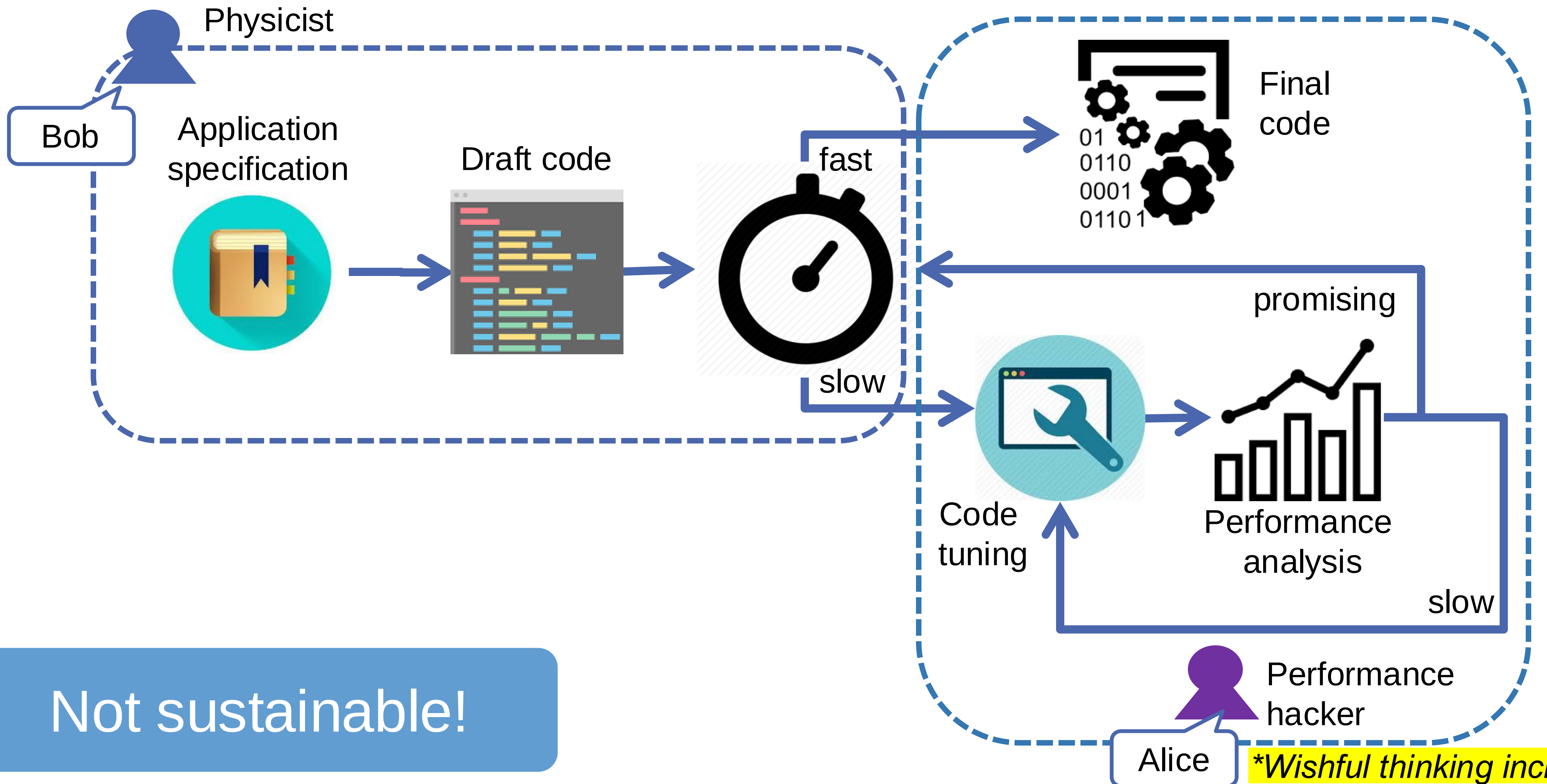
“Co-design: to design (something) by working with one or more others **:** to design (something) jointly” [3]

[1] https://www.mural.co/blog/co-design-method

[2] Patrick R. Schaumont, “A Practical Introduction to Hardware/Software Codesign”

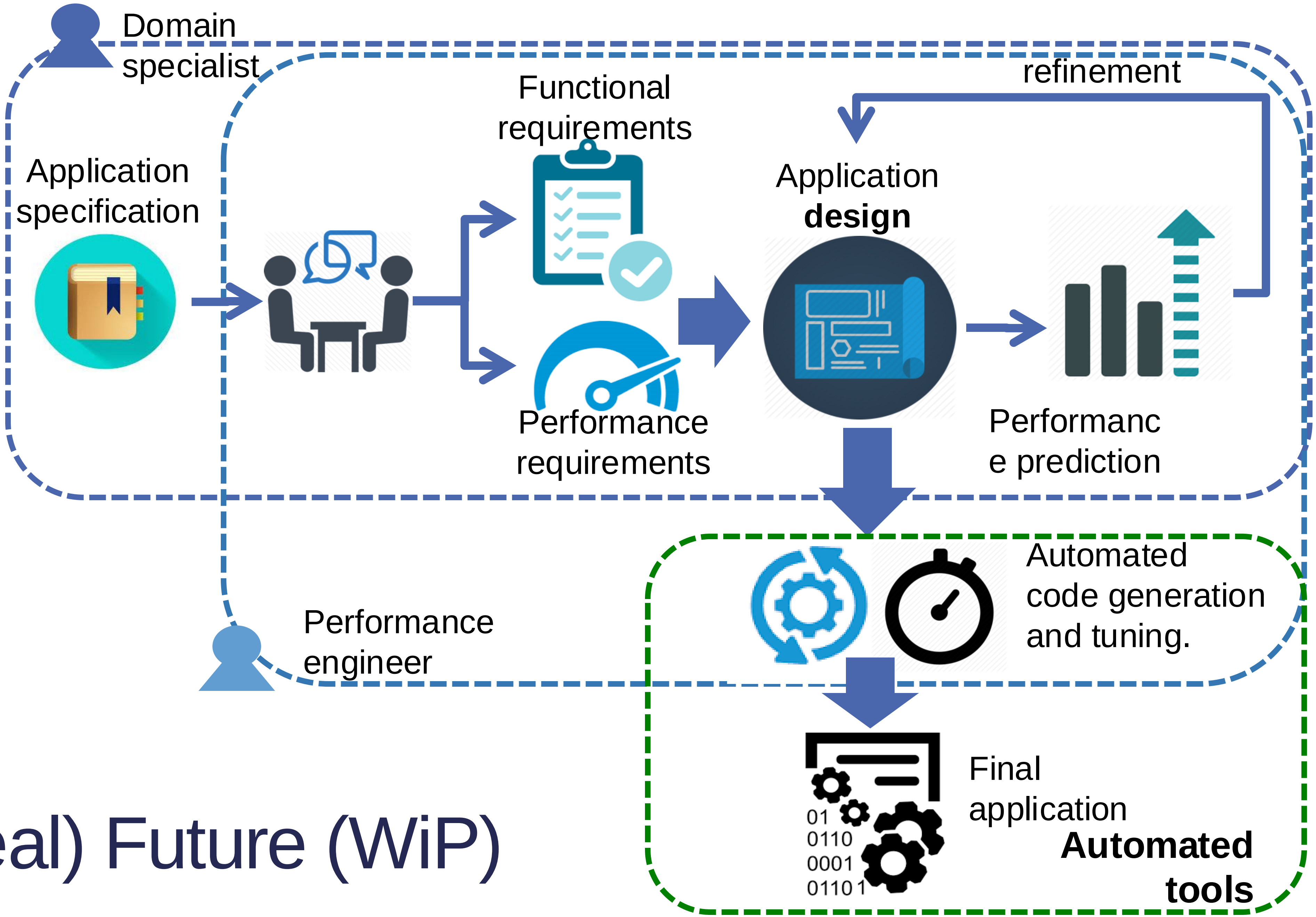
[3] https://www.merriam-webster.com/dictionary/codesign

Today’s approach to high-performance



Not sustainable!

**Wishful thinking included…*



(An ideal) Future (WiP)

Open questions

- What is the right abstraction for the input?
- How do we split the workload in “basic units”?
- How do we build “basic units” performance models?
- How do we prune the search space?
- How do we do code building and tuning?
- What about the data?
- …

In summary …

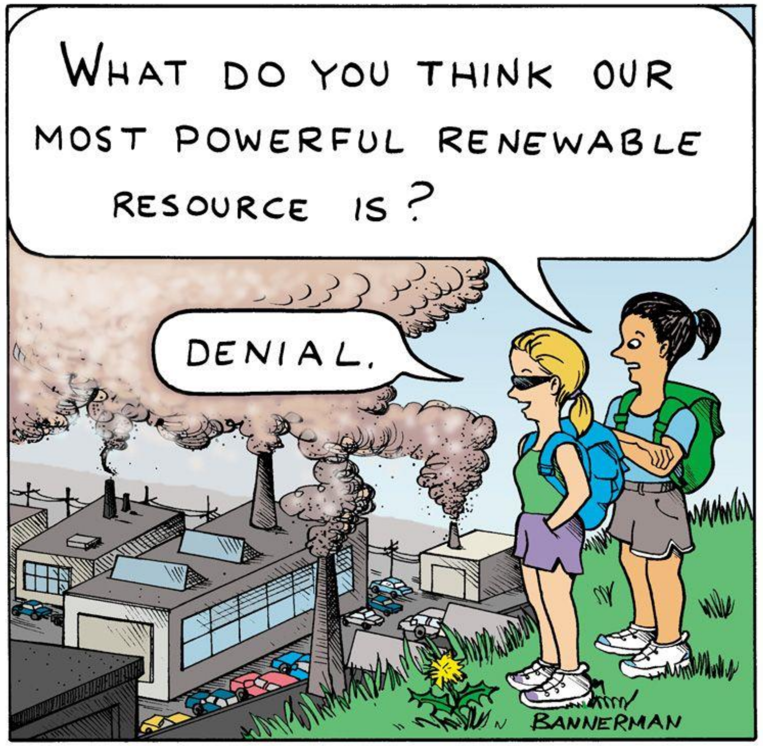
Take ~~home~~ message to-the-office

- High-efficiency computing is needed to avoid a (computing) energy crisis
  - Zero-waste computing is a strong motivating example …
  - … but we need tools and methods for it.

- Reduce waste in computing is within reach
  - Improve applications **and** improve systems

- Tempting to co-design applications and systems
  - Performance engineering to the rescue
    - Many practical questions still arise …

- We propose a co-design approach in Graph-Massivizer



Towards Zero-waste Computing

- **Awareness**: utilizing computing resources with little efficiency is equivalent to wasting computing.

- **Performance and efficiency**: non-functional properties, such as performance and efficiency, are essential to understand computing waste.

- **Design-time**: performance/efficiency must be essential concerns, like functionality

- **Stakeholders**: domain-specialists/application owners must (also) take responsibility in reducing waste in computing.

To do: Zero-waste computing

- Design and development:

“Build the right computing system for the job at hand”

  - Better hardware
    - Design and modeling to build the right infrastructure
  - Better software
    - Performance and energy analysis is essential to improve efficiency
  - Better tools
    - For design, analysis, and modeling

- Awareness:

“Acknowledge and improve the efficiency of ‘generic’ systems”

  - Better metrics
    - To demonstrate the waste in computing
  - Better methods
    - To analyse the complex tradeoffs between performance, energy, QoS, …