TOWARDS
ZERO-WASTE COMPUTING

Ana-Lucia Varbanescu

a.l.varbanescu@utwente.nl

Computing is everywhere … and it's not free!

- Top 10 videos on YouTube* consumed as much as 600-700 EU persons per year

- Training Alpha-Zero for a new game consumes as much as 100 EU persons per year

- A mid
  - And

We (=big science) are part of this!

- In 2019 Dutch datacenters combined consumed 3-times more energy than the natior
  - A cc

We must act to improve the energy efficiency of computing!

- The

*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 integrators

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

Include efficient hardware to enable different application mixes.

System operators

Ensure efficient scheduling of workloads on system resources.

Harvest energy where resources/systems are massively underutilized.

Agenda

- From performance to waste in computing
- Performance Engineering in a nutshell
- N Case-studies
  - *N <= 3*
- Towards Zero-waste computing

Why care about compute performance?

As an (impatient) user …

- Your application is not responsive
- Your simulation is not ready in time
- Your data is not fully processed
- …

Just buy a newer/bigger computer !
Run it "in the cloud" !
Or ask Stephen, he's a computer scientist !

Waste in computing

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

Why care about compute performance?

As a (mindful) user/developer …

- Must be aware of how you use computing resources
  - Reduce waste in computing
- Must be proactive about performance
  - If someone else needs to do it, it is already too late …
- Must argue for a more sustainable answer than more hardware

We **all can and must** improve software and hardware *efficiency* to minimize waste in computing!

Waste in computing

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

To reduce compute waste, we must shift from time-to-solution towards efficiency-to-solution

Why is compute efficiency challenging?

It is a **nonfunctional** requirement

Focuses on user-“irrelevant” issues like resource utilization, scalability, …

We all make a lot of excuses

It’s s

It’s ju

- Mor

… and new applications and new computing systems emerge monthly …

It’s easy to fix later

It’s “just engineering”

Requires effort,

and there’s (often) little glory in it.

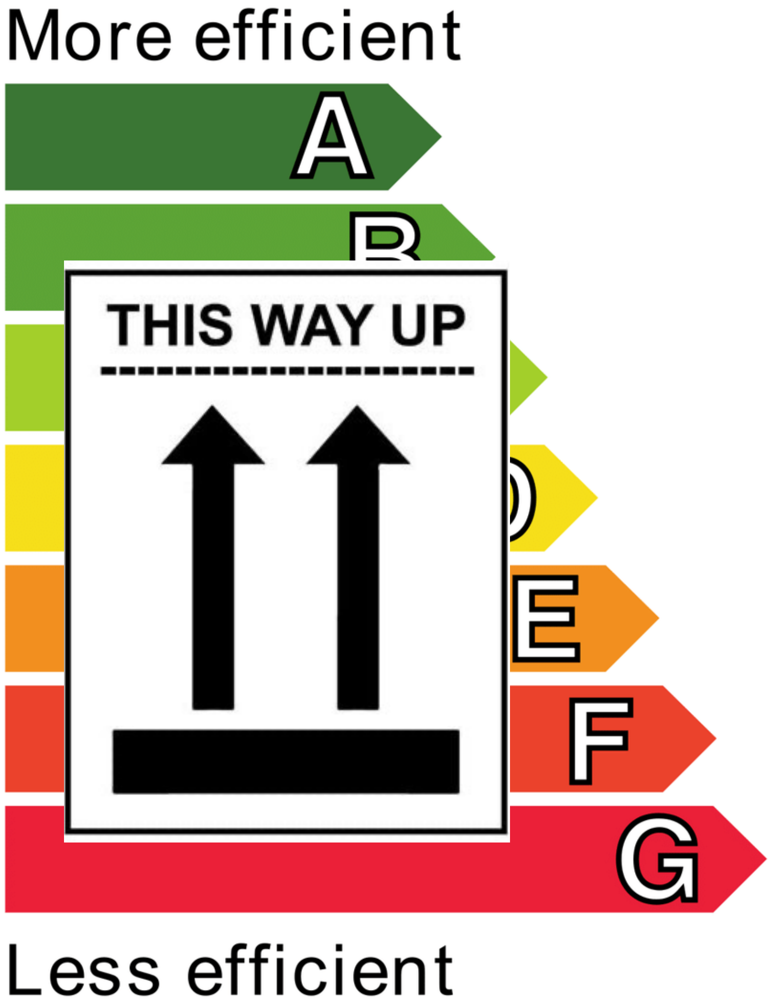
Reducing waste in computing

Raise awareness

- Quantify (energy) efficiency
- Quantify waste

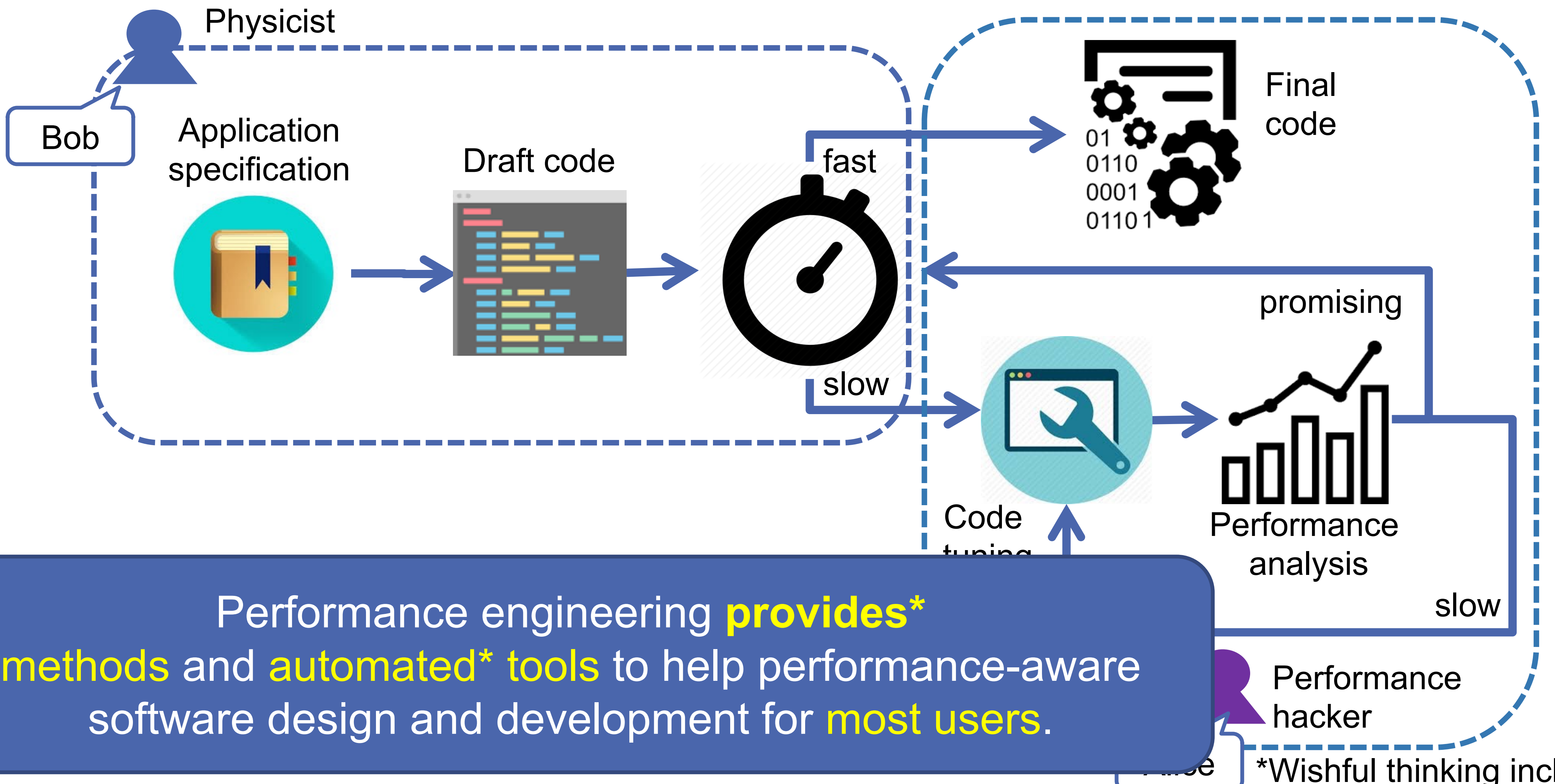
Improve compute efficiency

- 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



Introducing performance engineering

Today's approach to high-performance



Performance engineering **provides***
methods and automated* tools to help performance-aware
software design and development for most users.

*Wishful thinking included…

Performance engineering is systematic … and iterative…

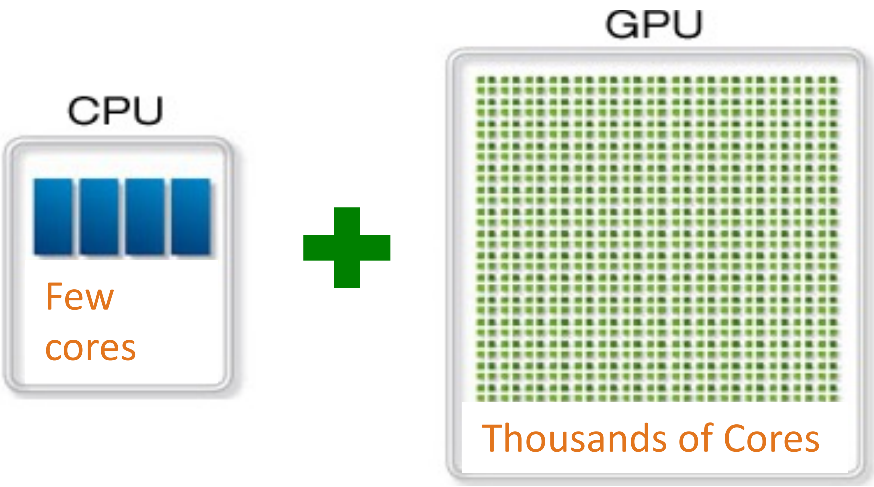
1. Capture **requirements**
2. Monitor **performance**
   (micro)benchmarking & hardware counters
3. Analyze **feasibility**
   Performance modeling
4. Design and implement **new algorithms**
   Parallel/distributed computing languages
5. **“Optimize”** code performance
   Tool design and development
6. **Document** results
   Metrics, visualization, user-interaction

Case-studies in heterogeneous computing

Model-based heterogeneous computing

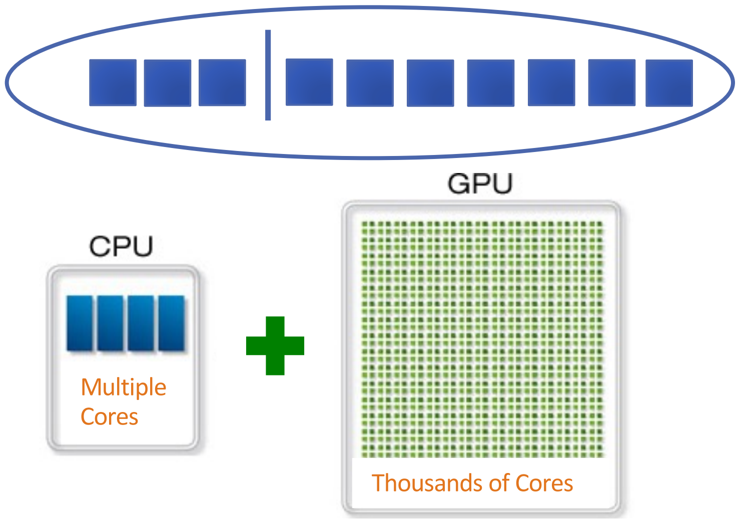
Heterogeneous computing?

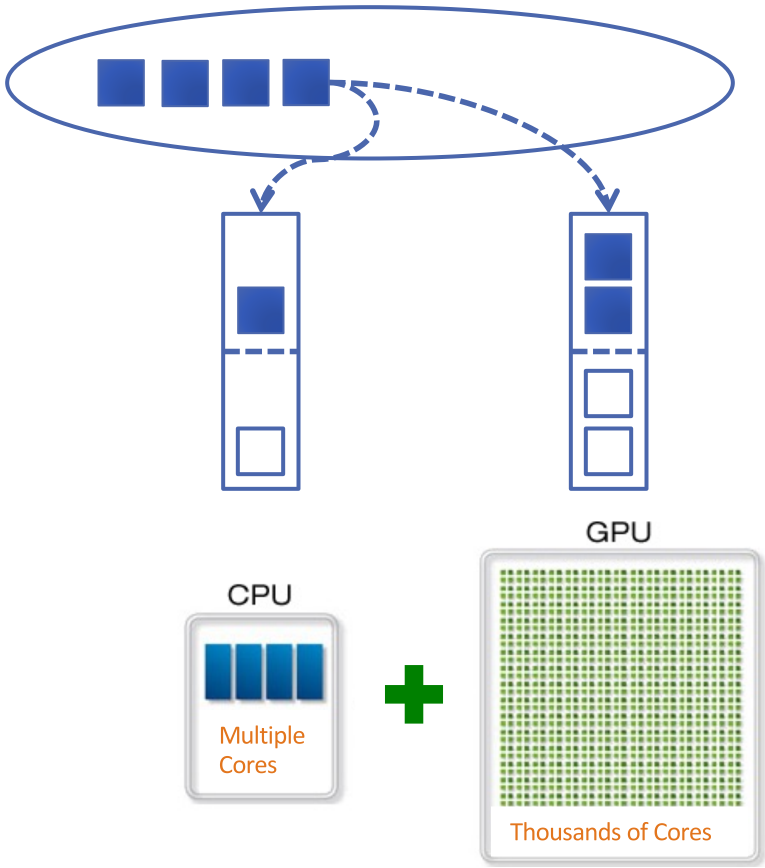
- A heterogeneous platform = a CPU + a GPU (the starting point)
- An application workload = an application + its input dataset
- Workload partitioning = workload distribution among the processing units of a heterogeneous system



Application execution

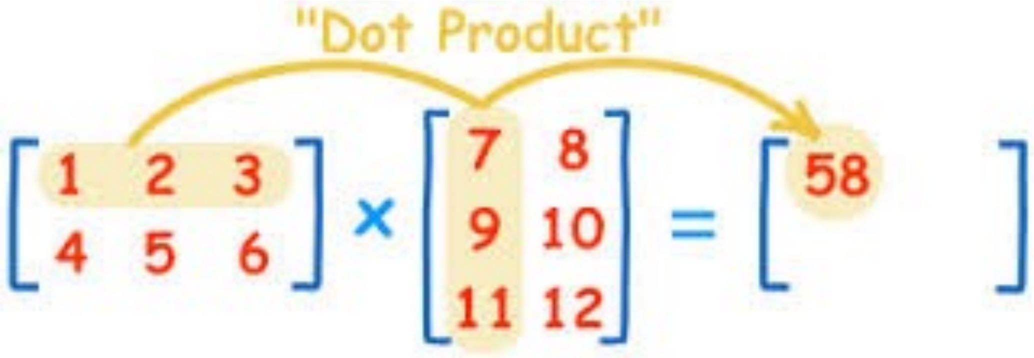
- Static partitioning (SP) vs. Dynamic partitioning (DP)



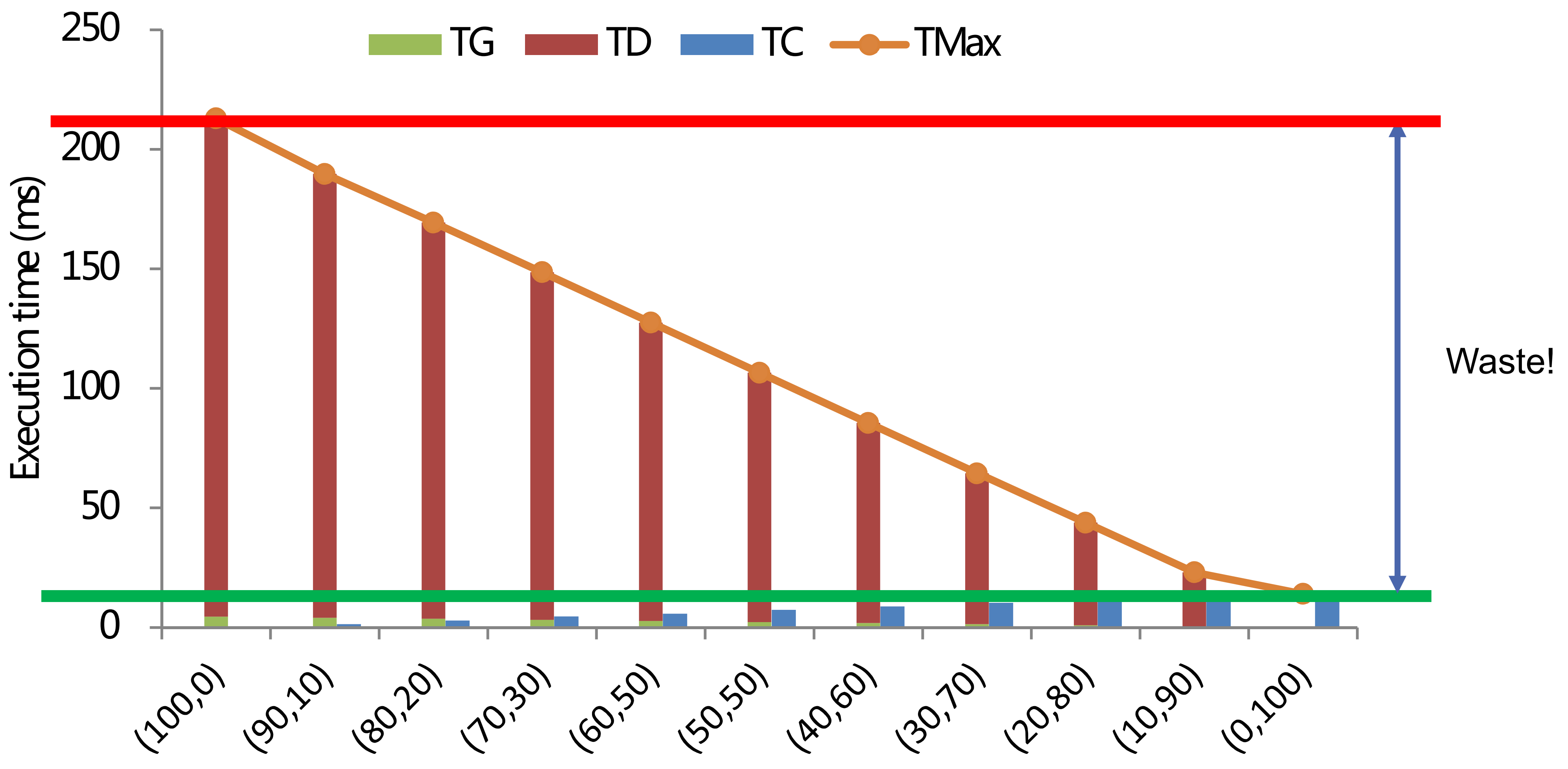


Example 1: dot product

- Dot product
  - Compute the dot product of 2 (1D) arrays
- Performance
  - T_G = execution time on GPU
  - T_C = execution time on CPU
  - T_D = data transfer time CPU-GPU
- GPU best or CPU best?

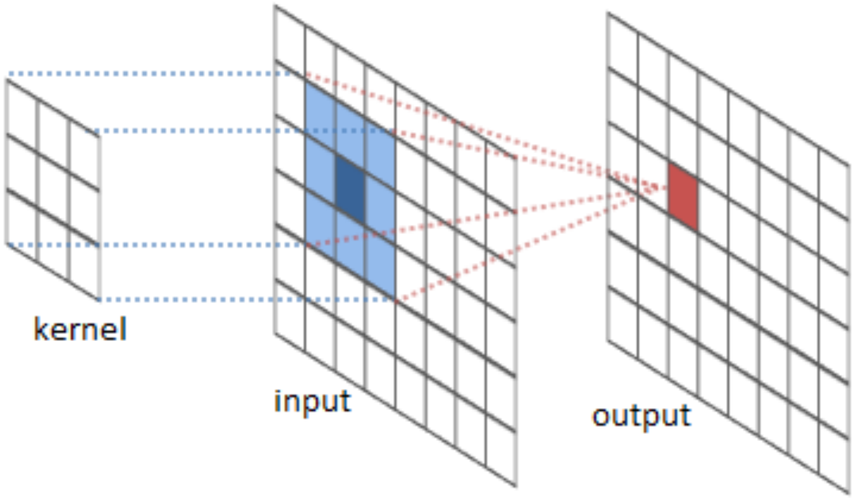


Example 1: dot product

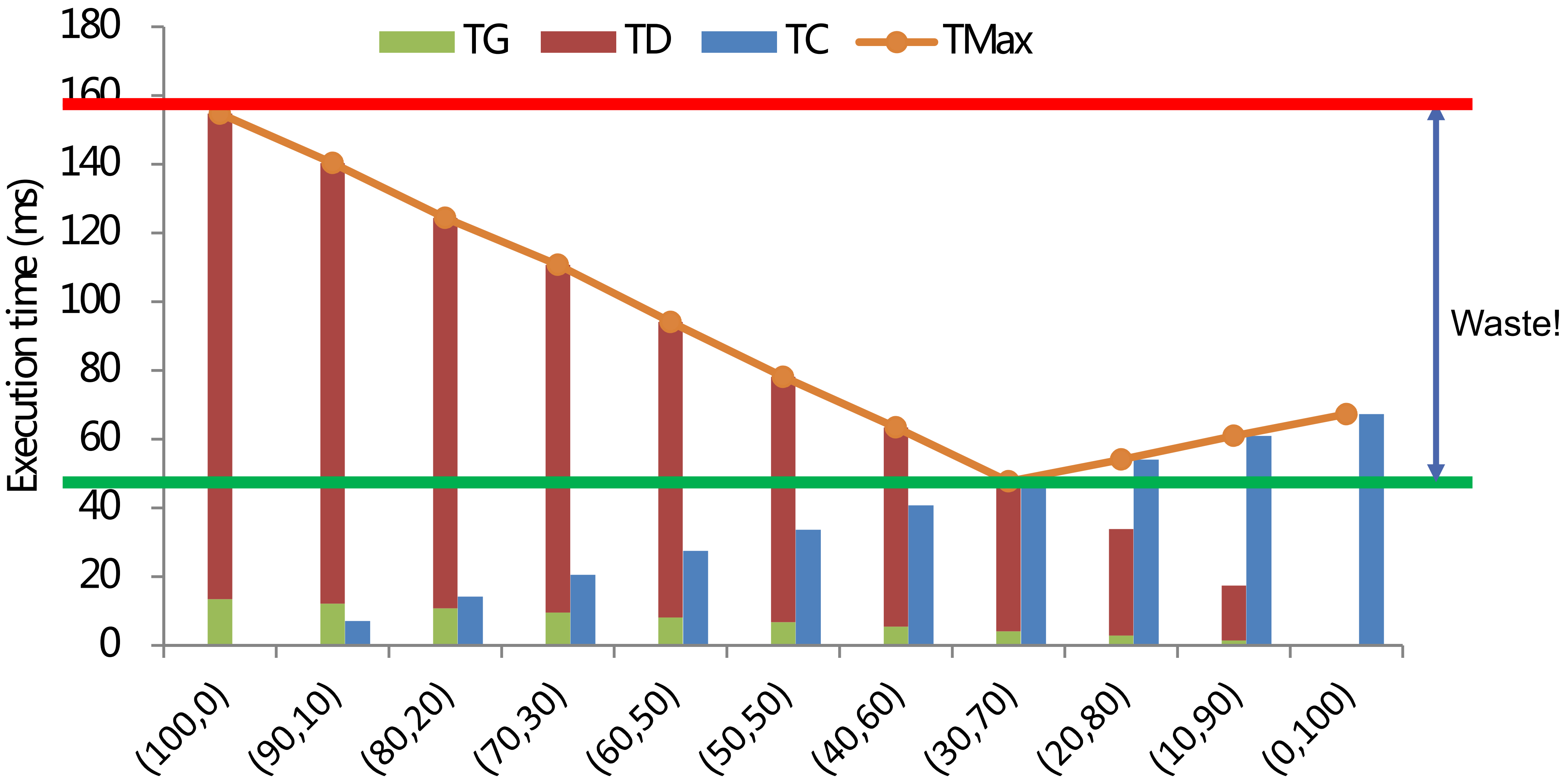


Example 2: separable convolution

- Separable convolution (CUDA SDK)
  - Apply a convolution filter (kernel) on a large image.
  - Separable kernel allows applying
    - Horizontal first
    - Vertical second
- Performance
  - T_G = execution time on GPU
  - T_C = execution time on CPU
  - T_D = data transfer time
- GPU best or CPU best?

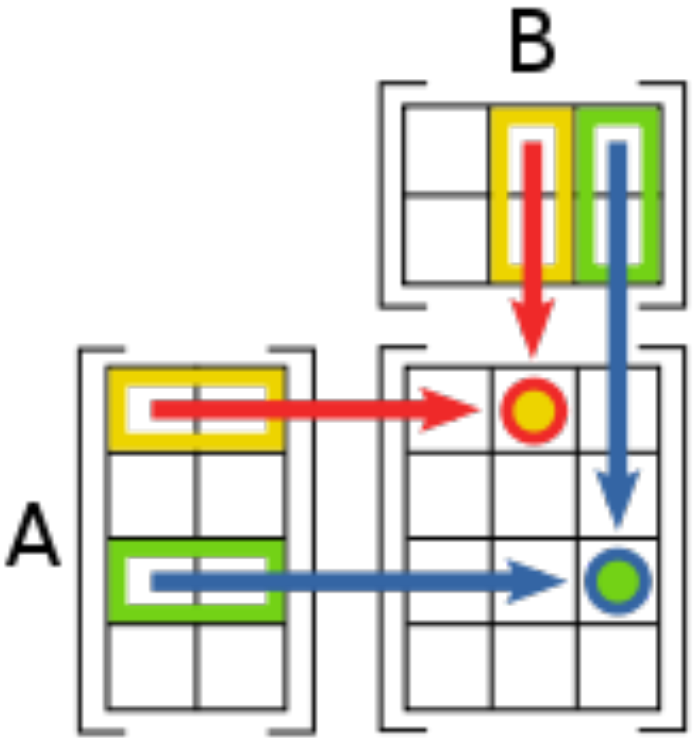


Example 2: separable convolution

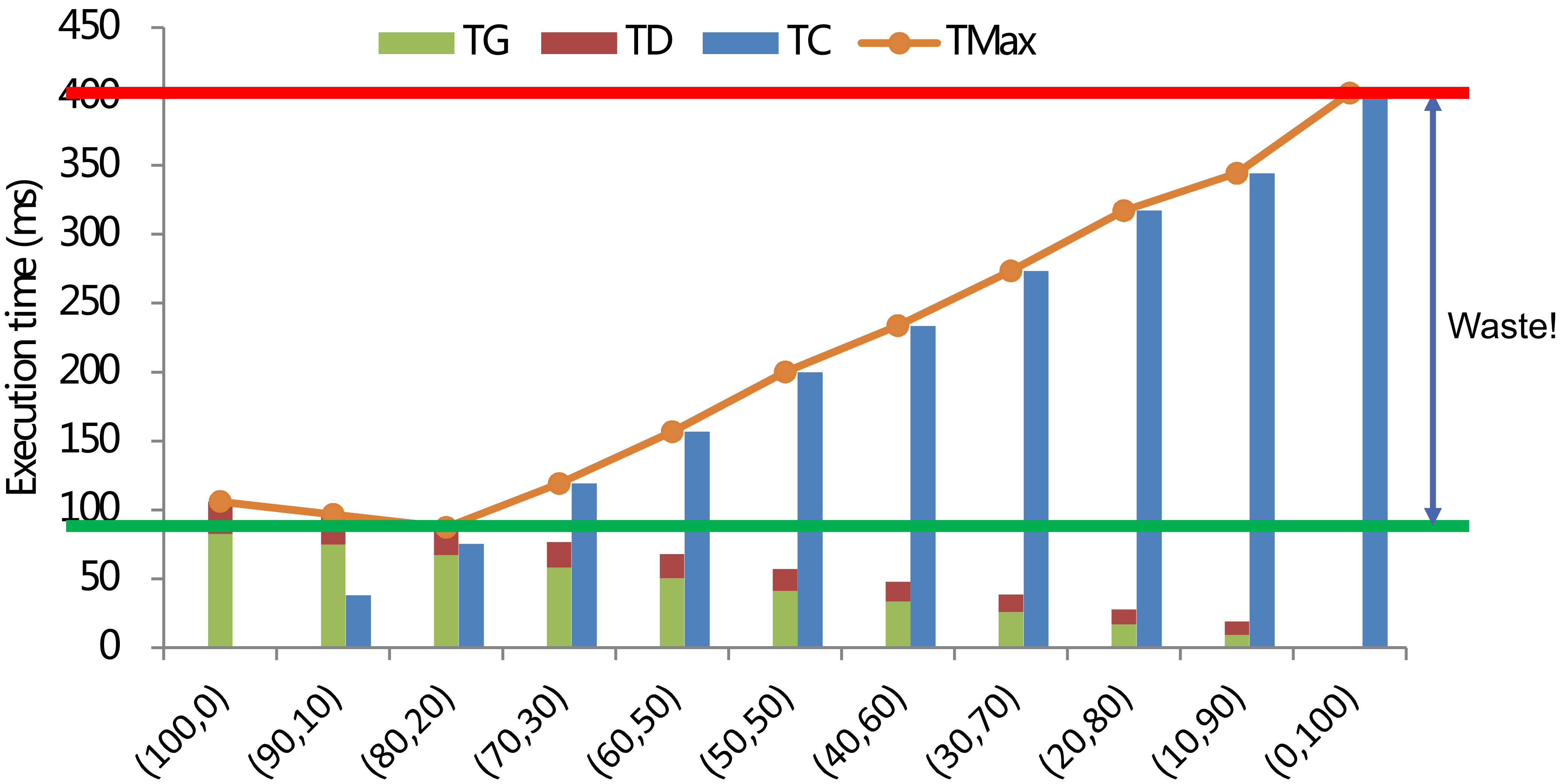


Example 3: matrix multiply

- Matrix multiply
  - Compute the product of 2 matrices
- Performance
  - T_G = execution time on GPU
  - T_C = execution time on CPU
  - T_D = data transfer time CPU-GPU
- GPU best or CPU best?

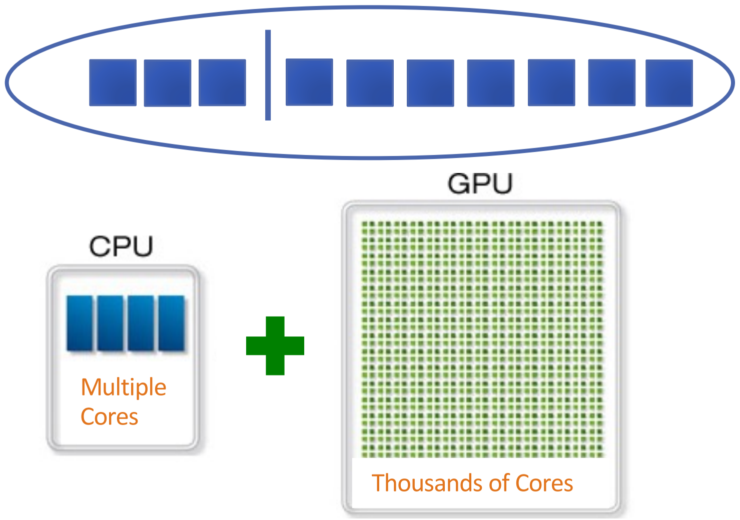


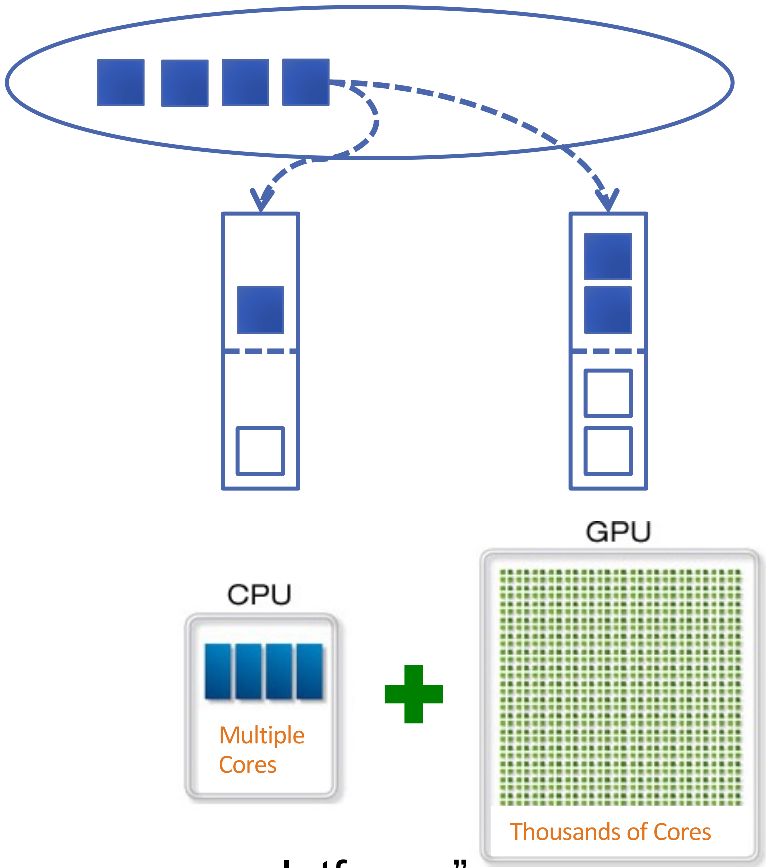
Example 3: matrix multiply



Determining the partition

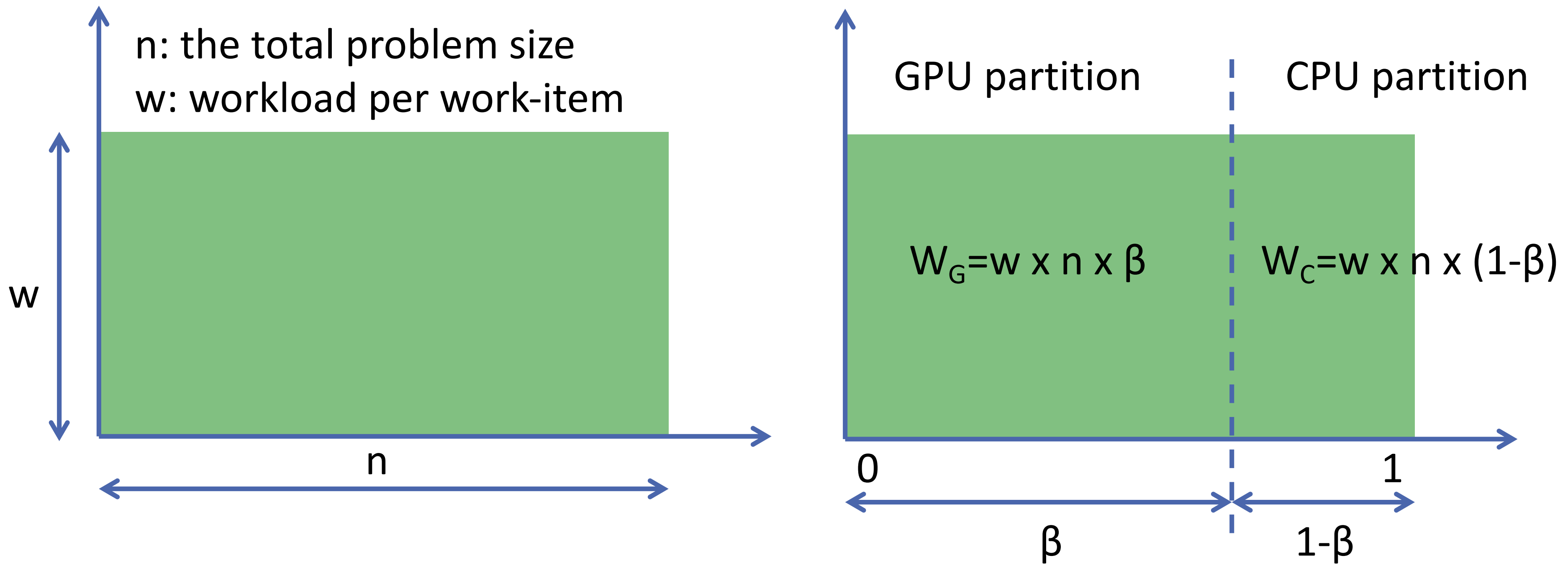
- Static partitioning (SP) vs. Dynamic partitioning (DP)





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

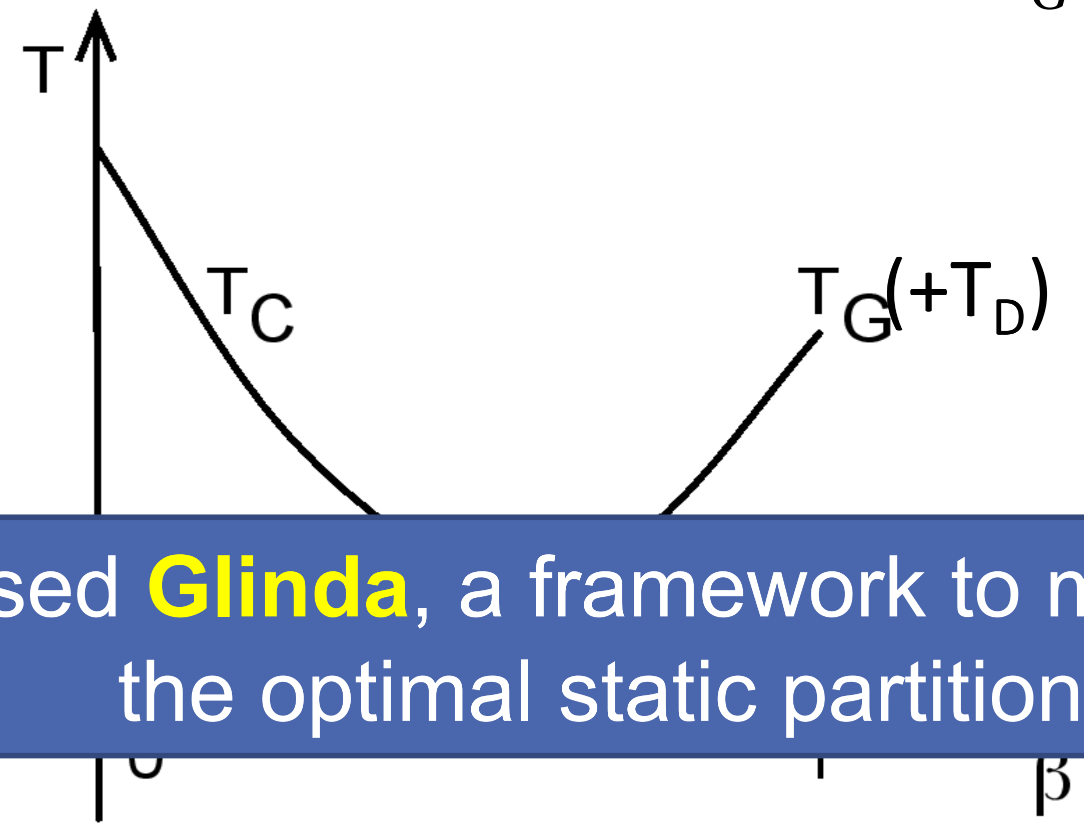
Application workload



W (total workload size) quantifies how much work has to be done in a partition

Optimal partitioning

$$T_G(\beta) + T_D(\beta) = T_C(\beta)$$



We proposed **Glinda**, a framework to model and predict the optimal static partitioning.

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

Predicting the optimal partitioning

- Solving β from the equation

Total workload size

HW capability ratios

Data transfer size

$$\frac{W_G}{W_C} = \frac{\beta}{1-\beta} = \frac{P_G}{P_C} \times \frac{1}{1 + \frac{P_G}{Q} \times \frac{O}{W_G}}$$

β predictor

- There are three β predictors (by data transfer type)

$$\beta = \frac{R_{GC}}{1 + R_{GC}}$$

No data transfer

$$\beta = \frac{R_{GC}}{1 + \frac{v}{w} \times R_{GD} + R_{GC}}$$

Partial data transfer

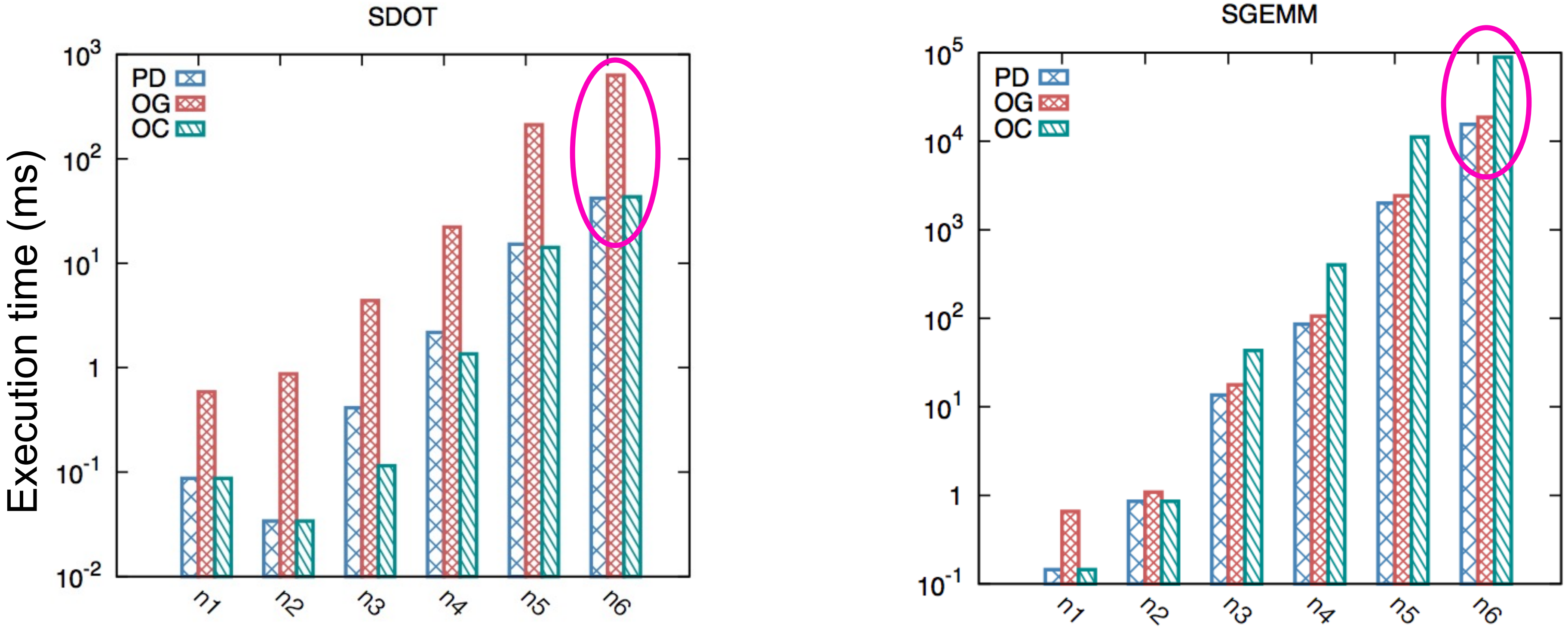
$$\beta = \frac{R_{GC} - \frac{v}{w} \times R_{GD}}{1 + R_{GC}}$$

Full data transfer

TitanX : ~280 Watts
CPU: ~80 Watts

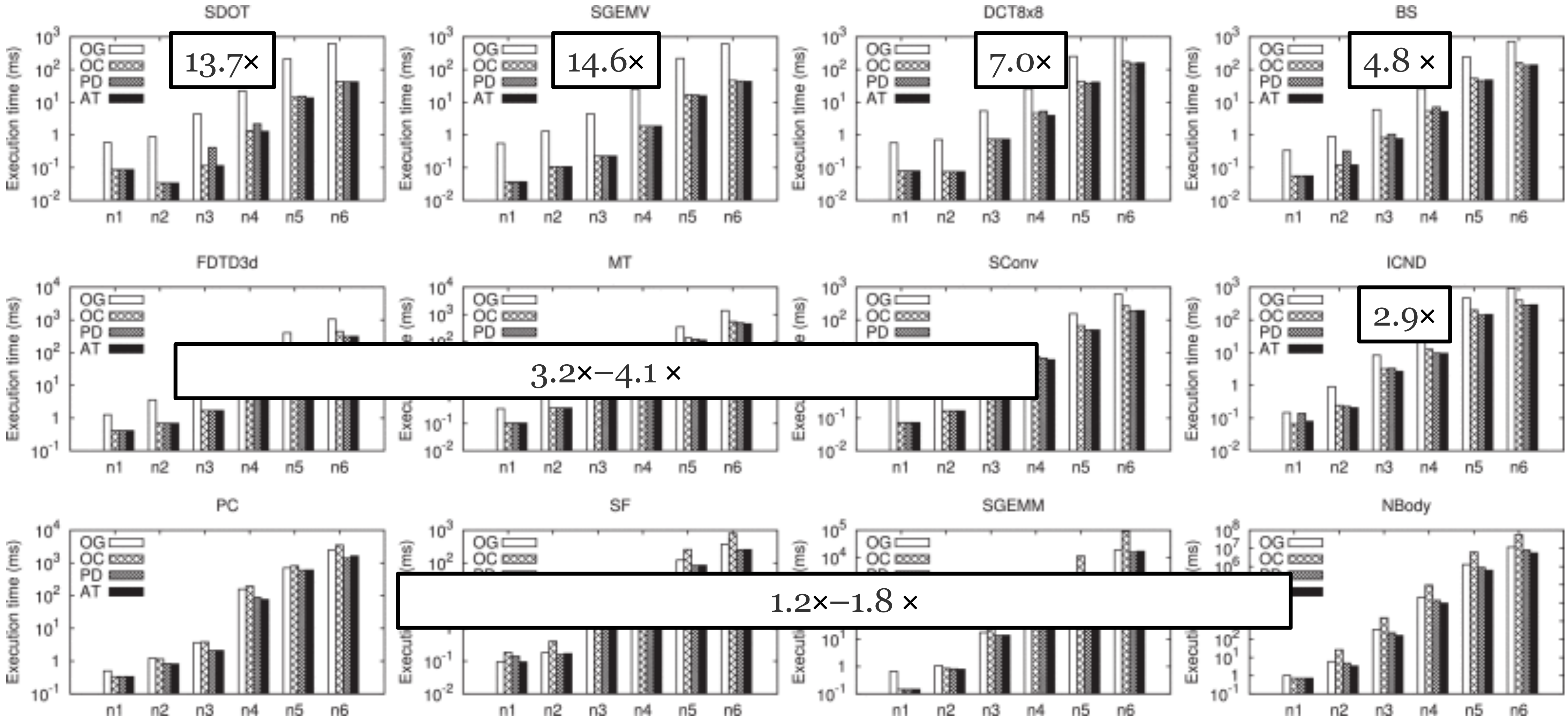
Evaluation

- Effectiveness (compared to Only-CPU/Only-GPU)
  - Up to 12.6x/6.6x speedup
- Wasted [%]: $T/T_{PD} * (P_{CPU} + P_{GPU}) = S * (P_{CPU} + P_{GPU})$
- Wasted : $(T-T_{PD}) * (P_{CPU} + P_{GPU})$



TitanX : ~280 Watts
CPU: ~80 Watts

Waste for GPU-only



Contributions & Lessons learned [1]

- 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

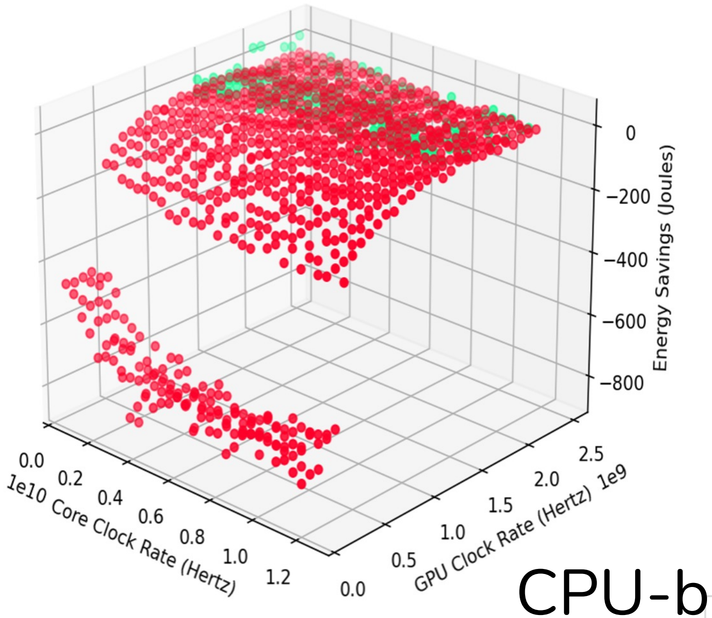
Case-study #2: Energy harvesting

Energy improvements

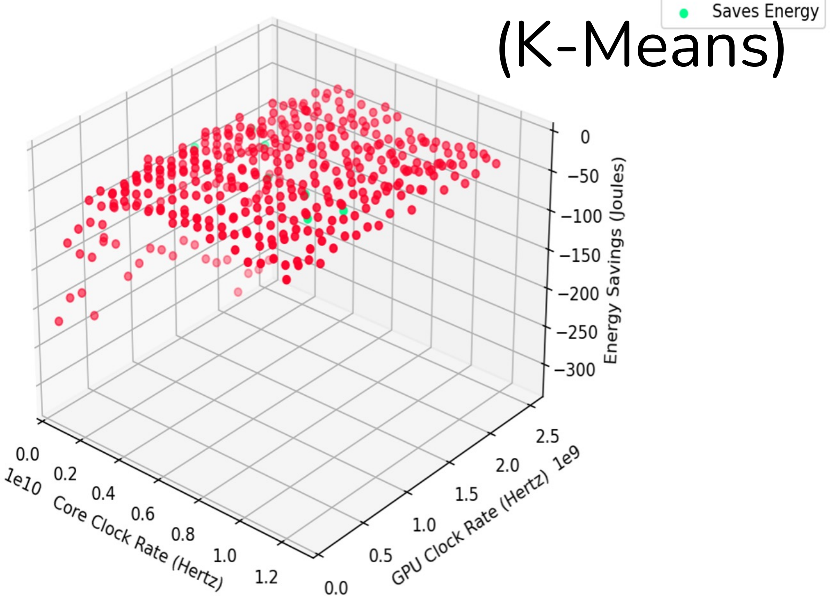
- Basic assumptions
  - Tasks run on different processors
  - Idle processors waste energy
  - Higher/lower operating frequencies
    - => more/less power respectively
    - => reduce or increase runtime respectively

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

GPU-bound
(Matrix Multiply)

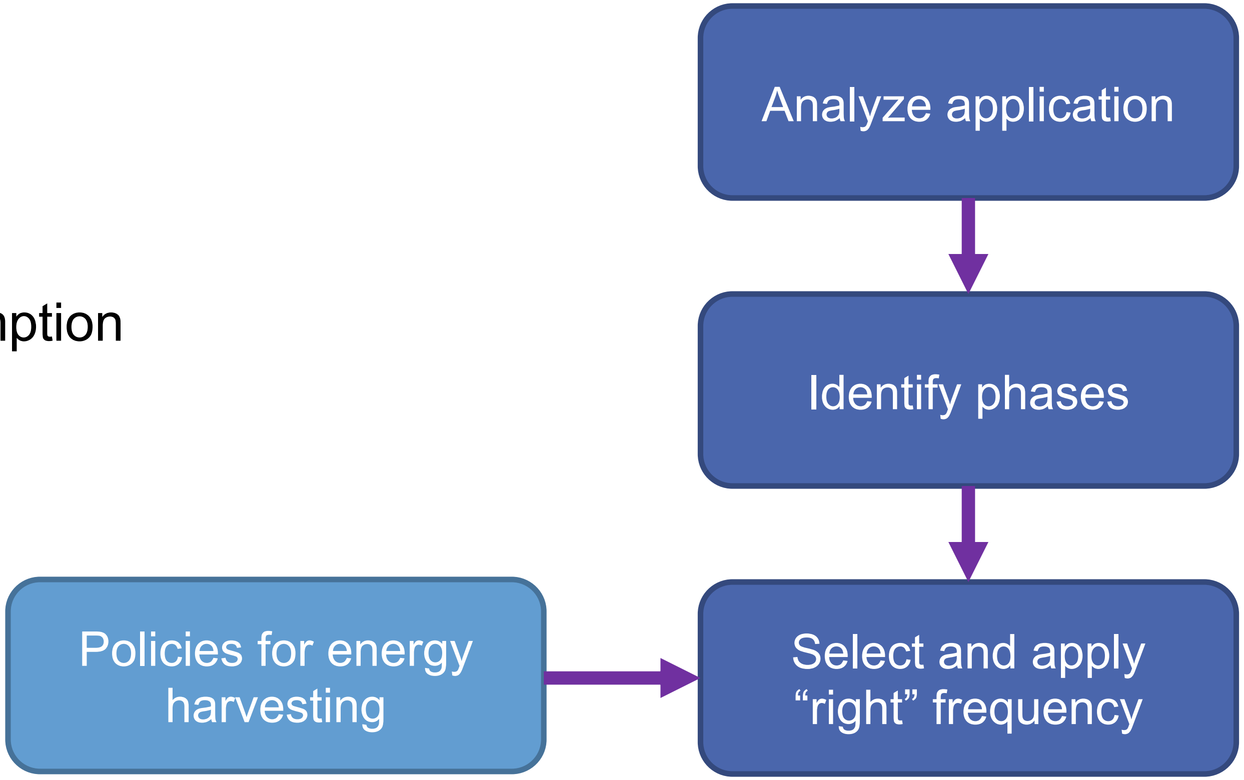


CPU-bound
(K-Means)



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



State detection

- **Monitoring framework**
  - Records performance variables: e.g., utilization rate, clock rate, ...

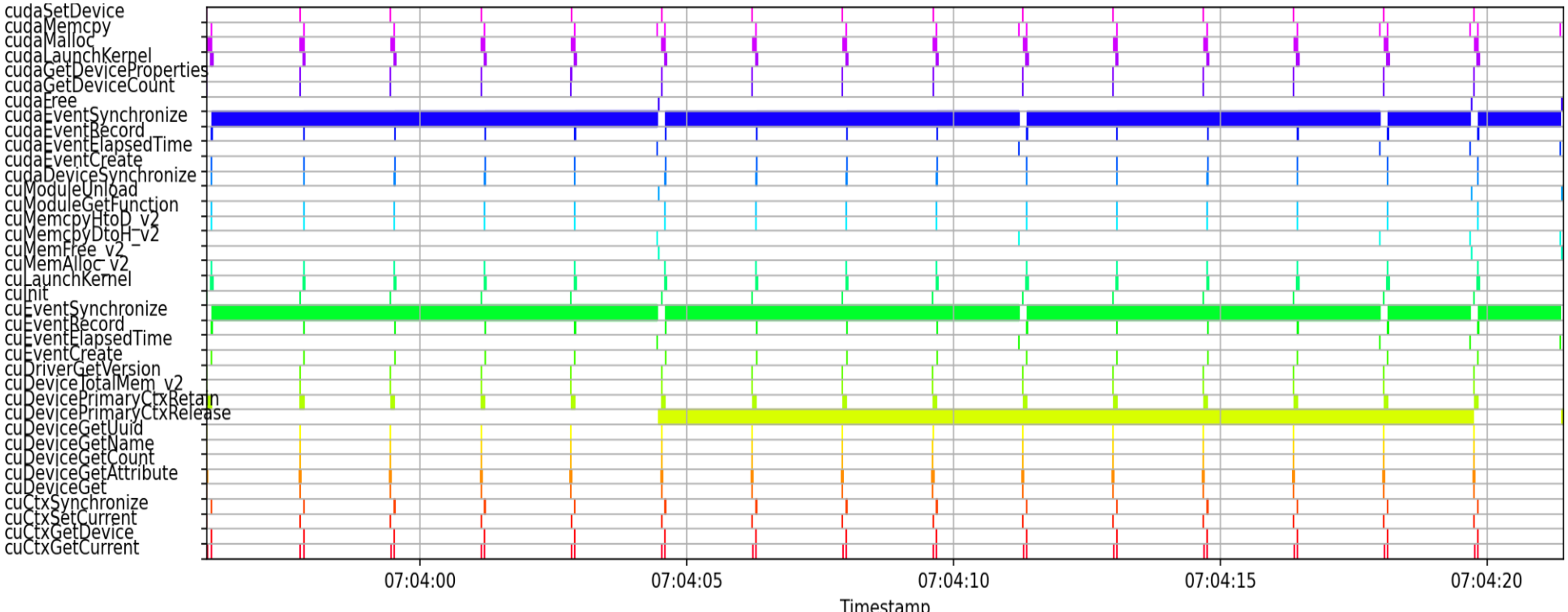
- **Application state detection** based on **processor utilization** and **application events**

- **5 states** of interest
  - CPU/GPU/BOTH IDLE
  - ALL BUSY
  - CPU BUSY WAIT

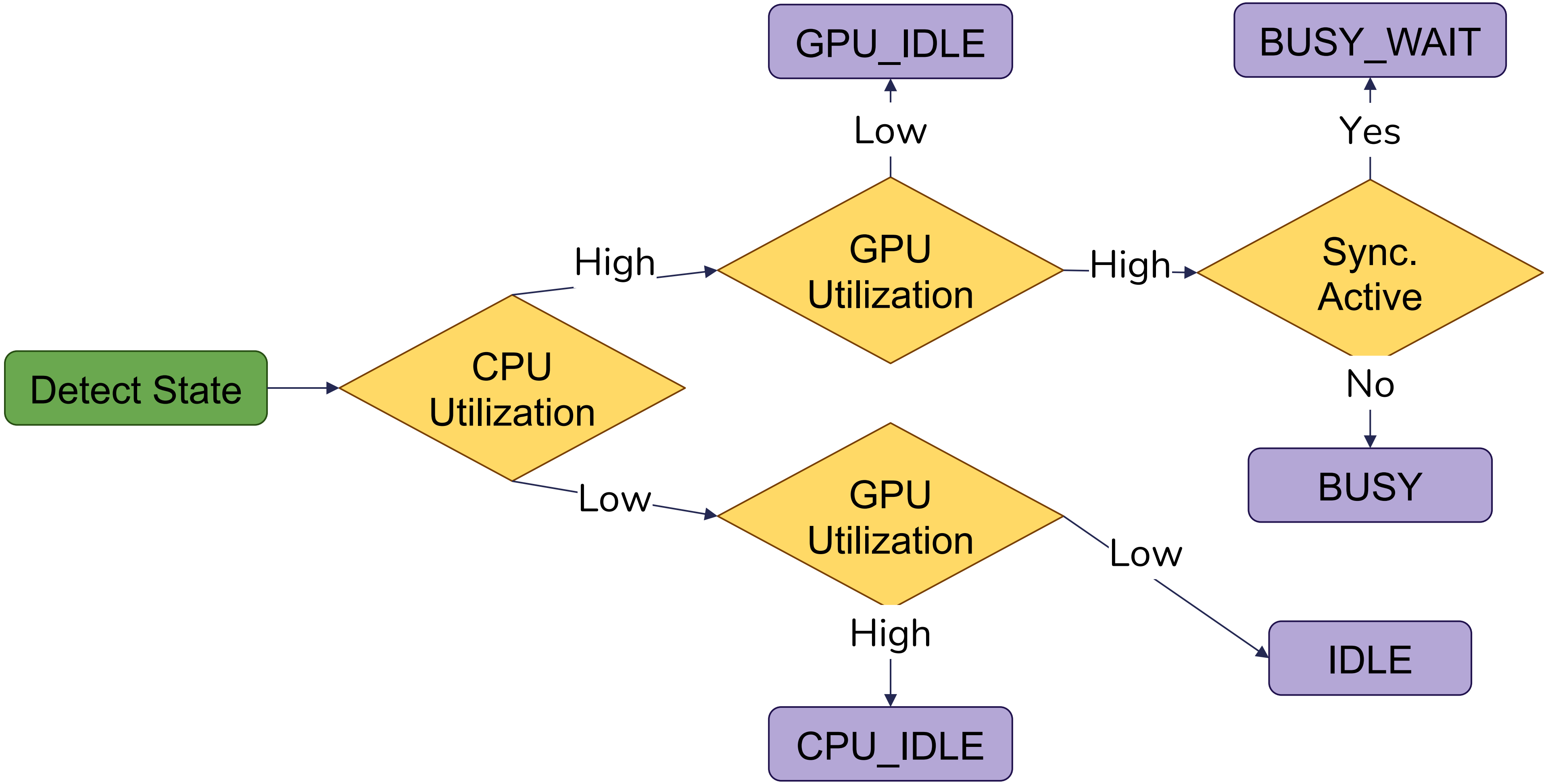
- State detection library
  - Detects all 5 different states every **10ms**

State detection

- **Monitoring framework** records performance variables: e.g., utilization rate, clock rate, ...
- **Application state detection** based on **processor utilization** and **application events**



States of interest

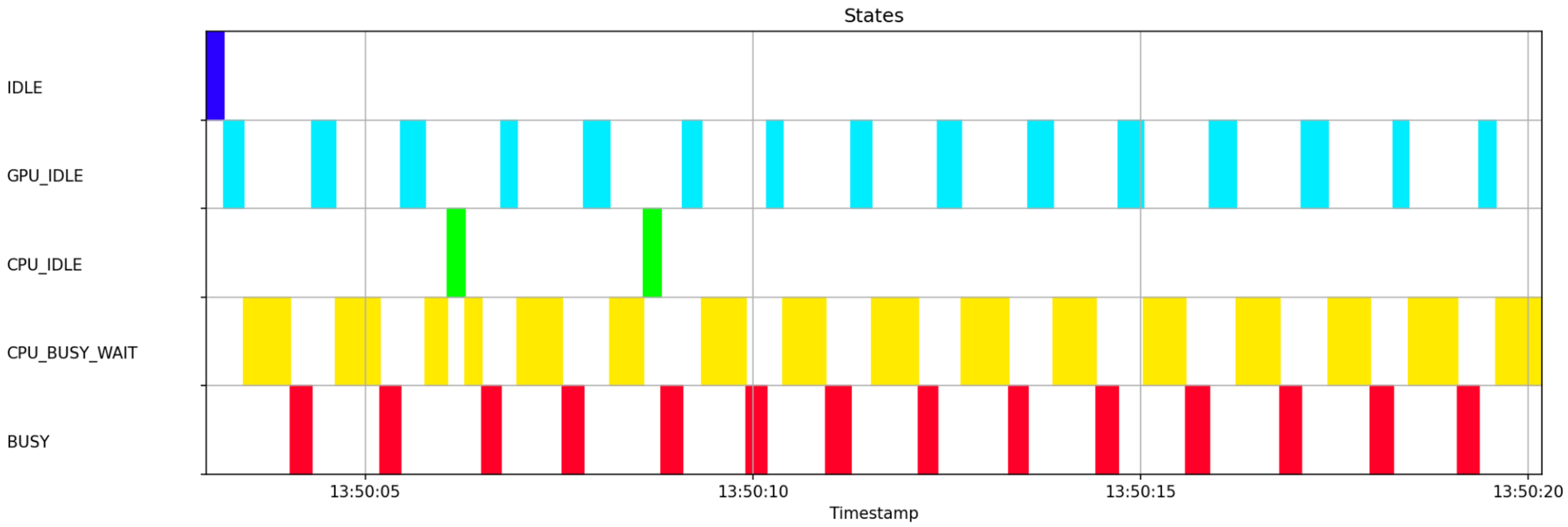


From States to Actions

- Policies define specific actions to take for each state
  - Mechanism = dynamic frequency scaling
  - Specific actions = how to scale the frequency (up/down, and by how much)
  - Ideal operating frequencies prevent “idling”
- Two different policies were defined
  - “MinMax” policy: Frequencies minimized or maximized in IDLE and BUSY states
  - “System” policy: Frequencies selected by powersave governor in IDLE and BUSY states

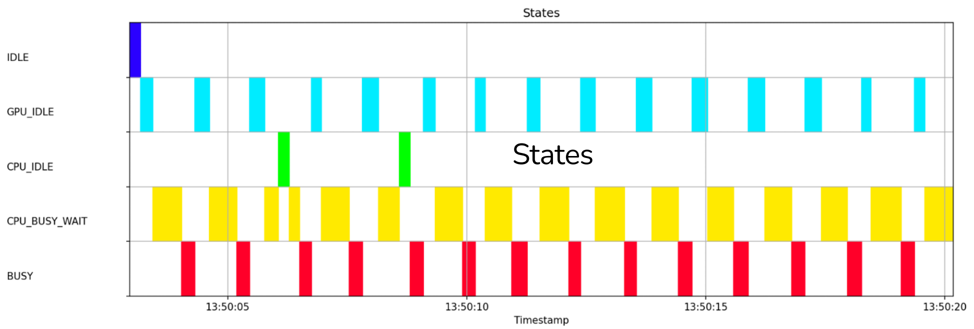
From States to Actions

- Detected states are used to trigger energy harvesting actions
  - Different states trigger different actions
    - E.g., CPU_IDLE triggers the “lower CPU frequency” action

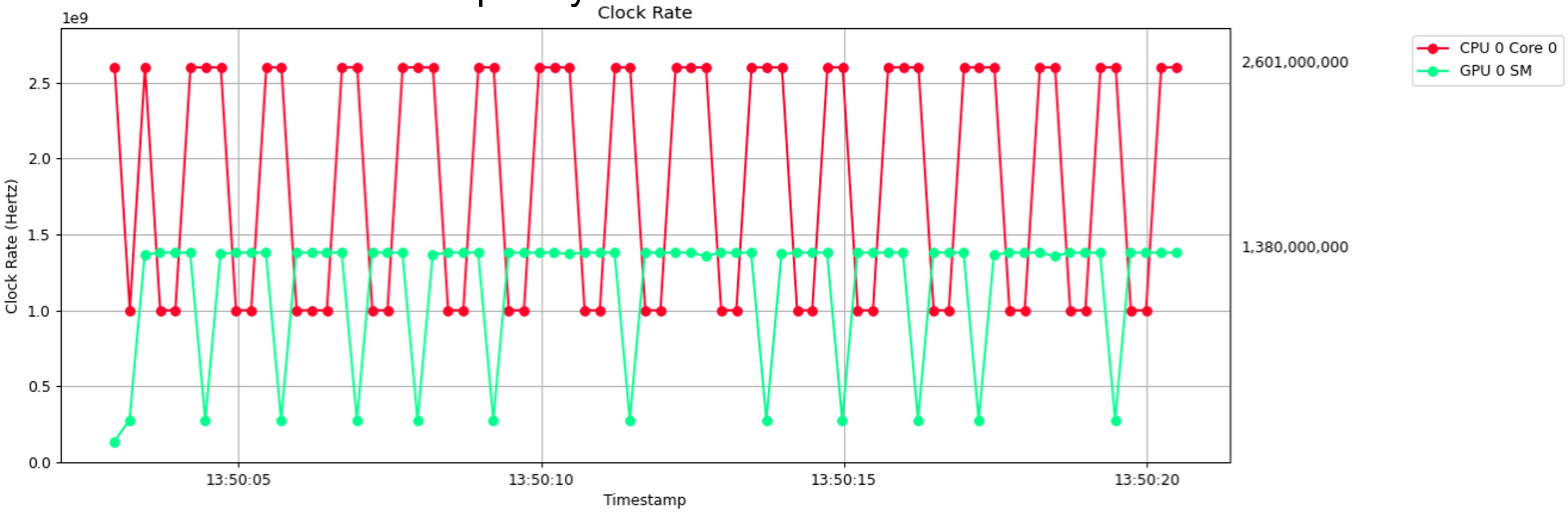


* Graph shows one execution of Matrix Multiply sourced from the NVIDIA CUDA Toolkit v10.2

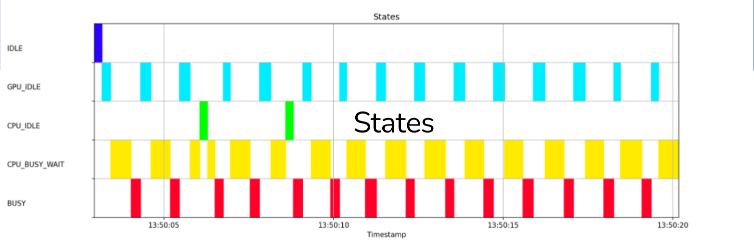
From States to Actions

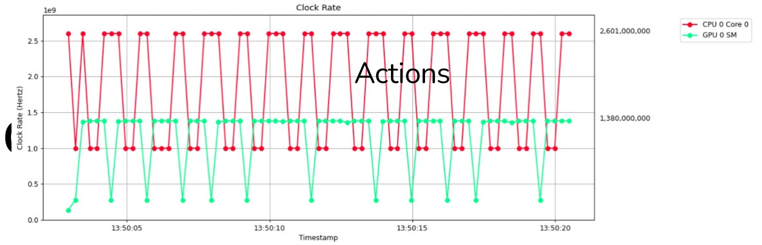


- Energy harvesting actions change the operating frequencies based on the current state
  - Busy states => increase the frequency
  - Idle states => decrease the frequency

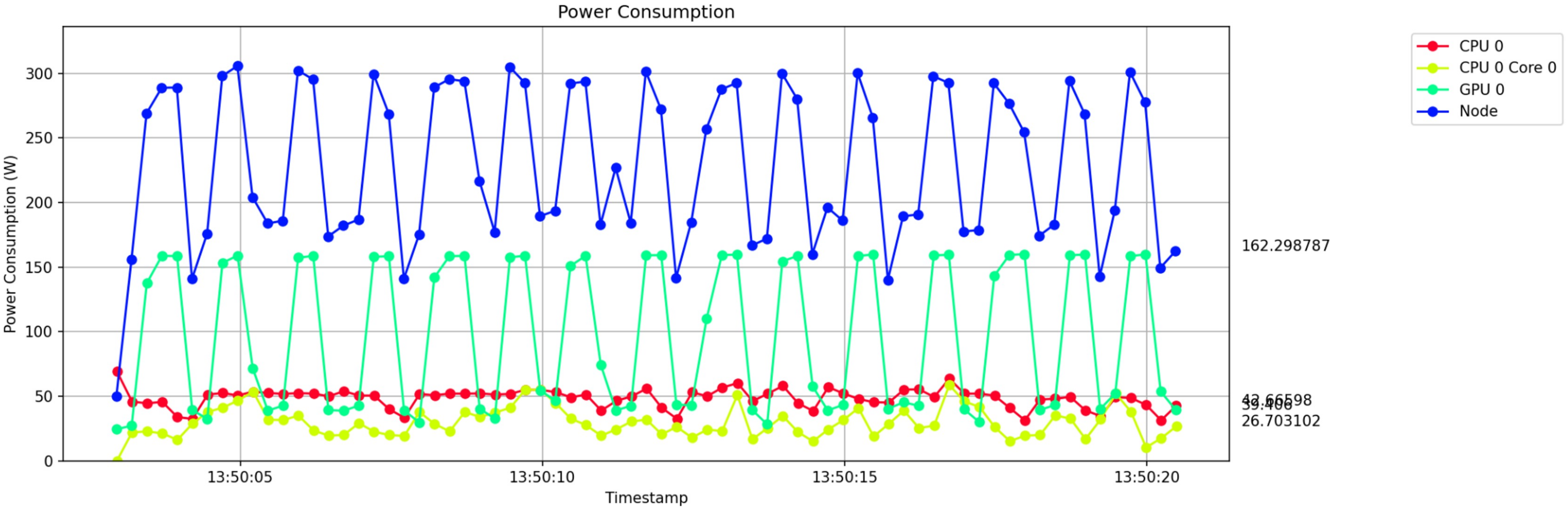


From States to Actions





- Changing operating frequencies affects power consumpti
  - Lower frequencies reduce power consumption



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	Policy											
	No Action		MinMax		System		Maximum frequency		Ranked MinMax		Scaled MinMax	
	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%

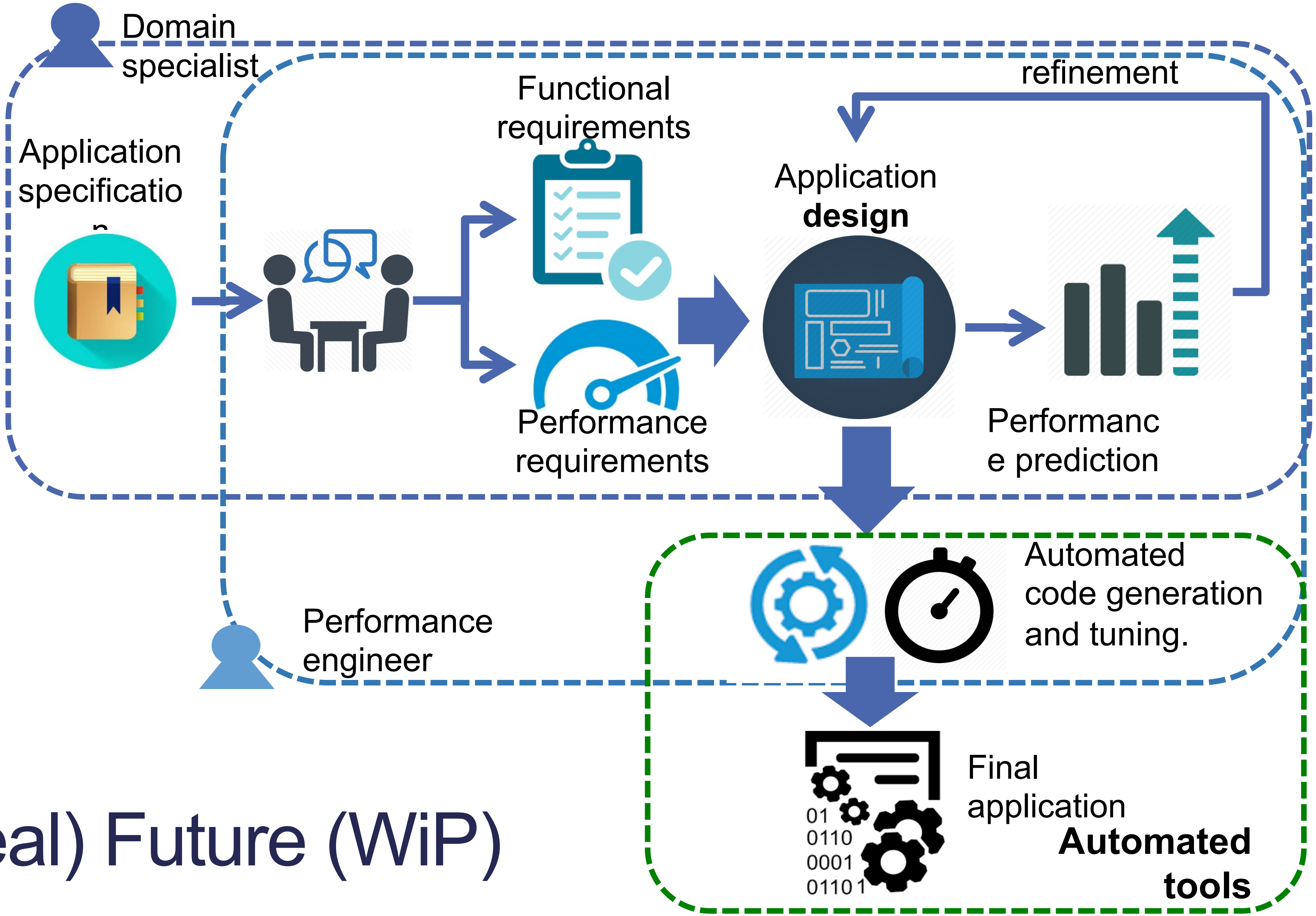
Contributions & Lessons [2]

- Heterogeneous computing => high performance, high energy consumption
- Energy harvesting can work
  - Depends a lot on implementation
- More interesting question: Can we (/should 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

Improving systems and applications

~~Case-study #3~~: Co-designing systems and applications



(An ideal) Future (WiP)

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

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, …