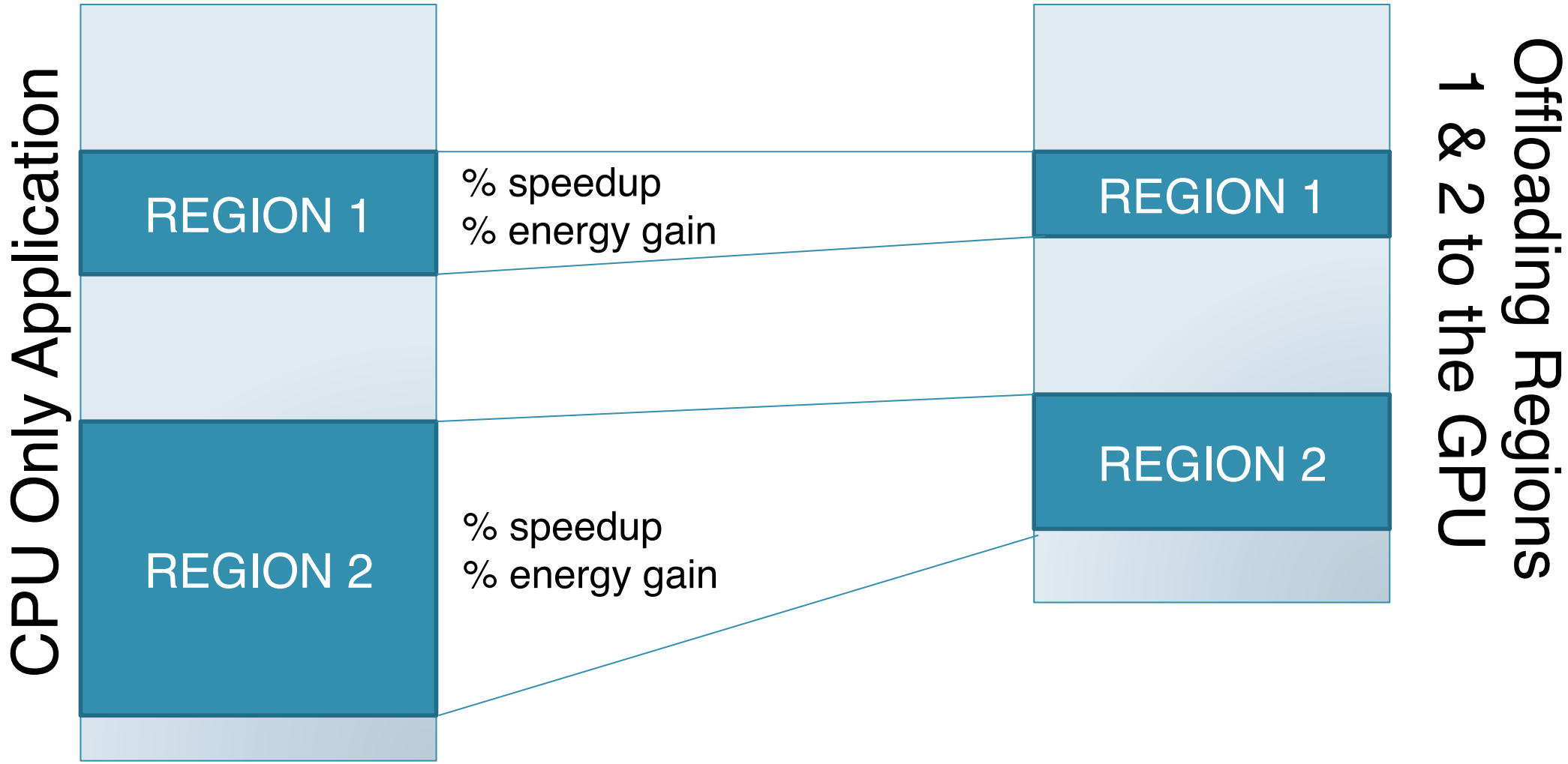
Offloading to the GPU: An Objective Approach

Ajaykumar Kannan, Mario Badr,
Parisa Khadem Hamedani, Natalie Enright Jerger
{kannanaj, badrmari, parisa, enright}@ece.utoronto.ca
University of Toronto

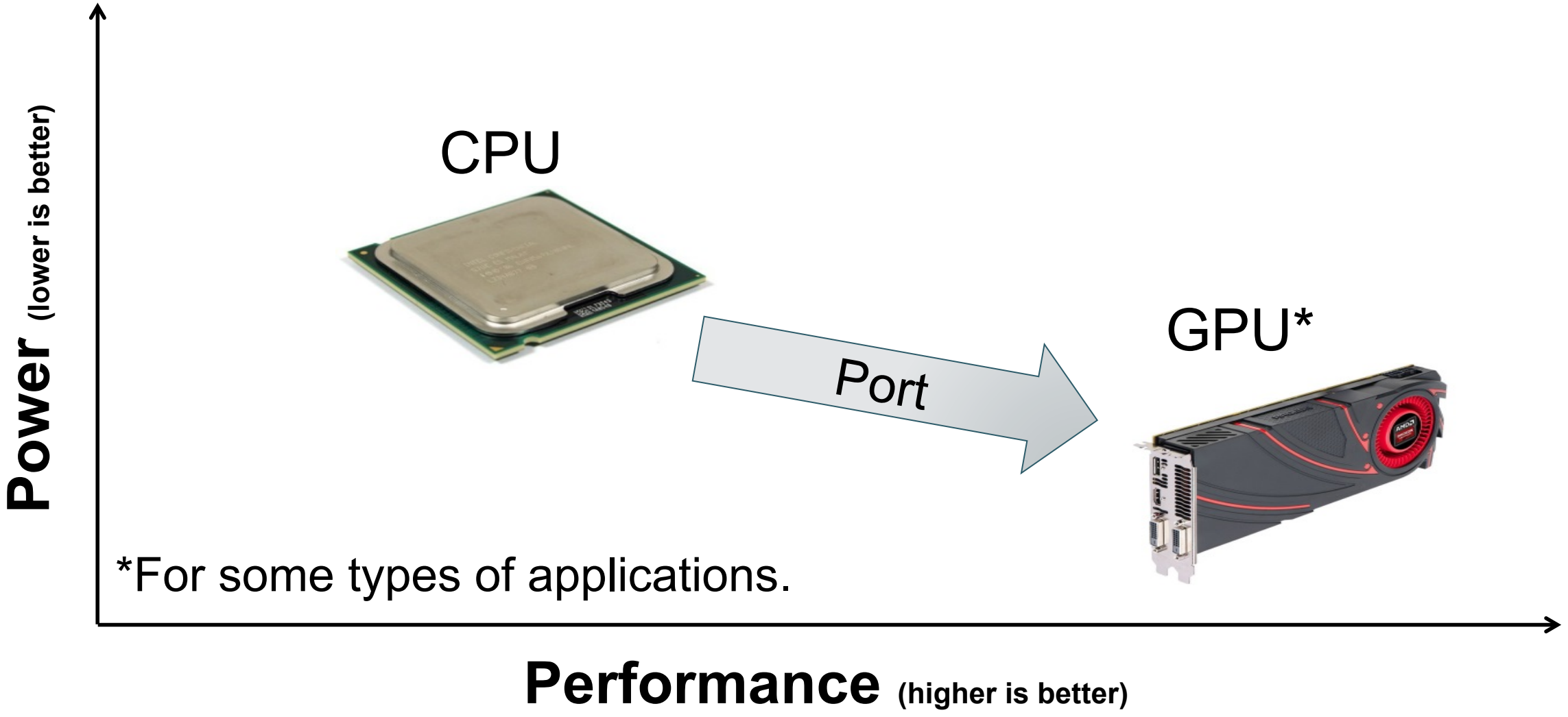
Overview


CPU Only Application
REGION 1
REGION 2
% speedup
% energy gain
% speedup
% energy gain
REGION 1
REGION 2
Offloading Regions
1 & 2 to the GPU

Outline

- Motivation
- Overview
- Methodology
- Modeling the devices
- Profiling the Application
- Evaluation of Sample Applications

Motivation: Improving Battery Life And The UX



Is It Worth It?

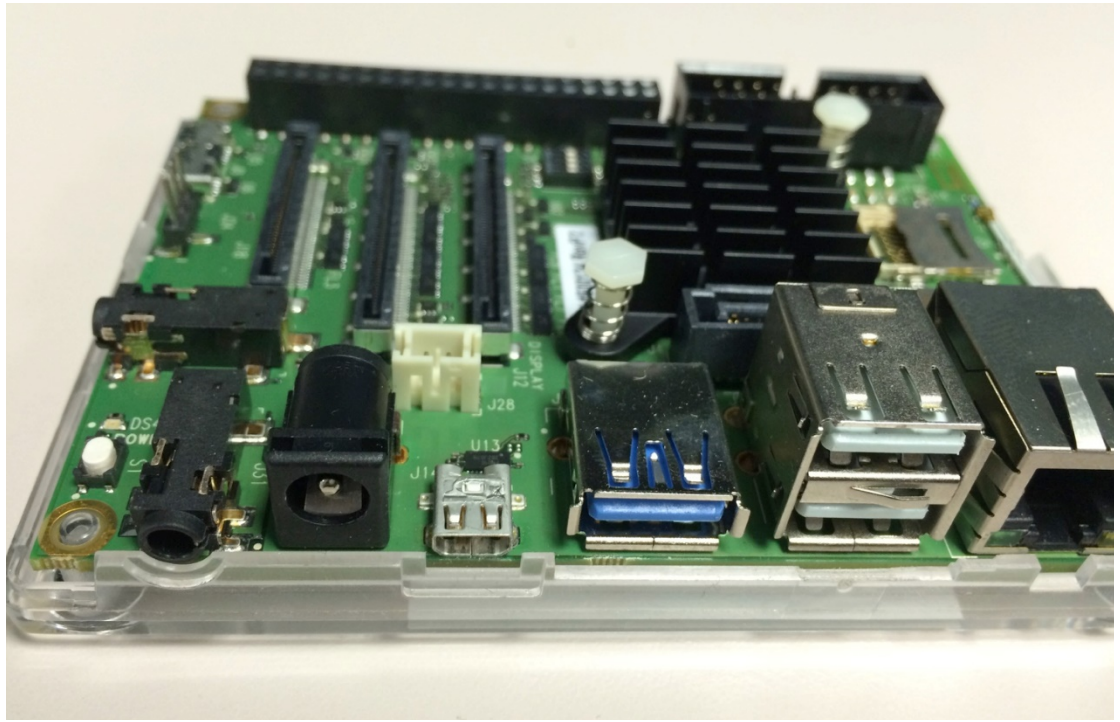
Qualitative	Quantitative
• Performs Better	• Estimate run-time (T)
• Saves Energy	• Estimate energy (E)

Goal

Given an estimated T and E for a port of the application, managers can objectively decide to commit developers to the project.

Methodology – The Setup

- Intrinsyc Snapdragon 805 Development Board
  - CPU:
    - Quad-Core Krait 450
  - GPU:
    - Adreno 420



Methodology – The Setup

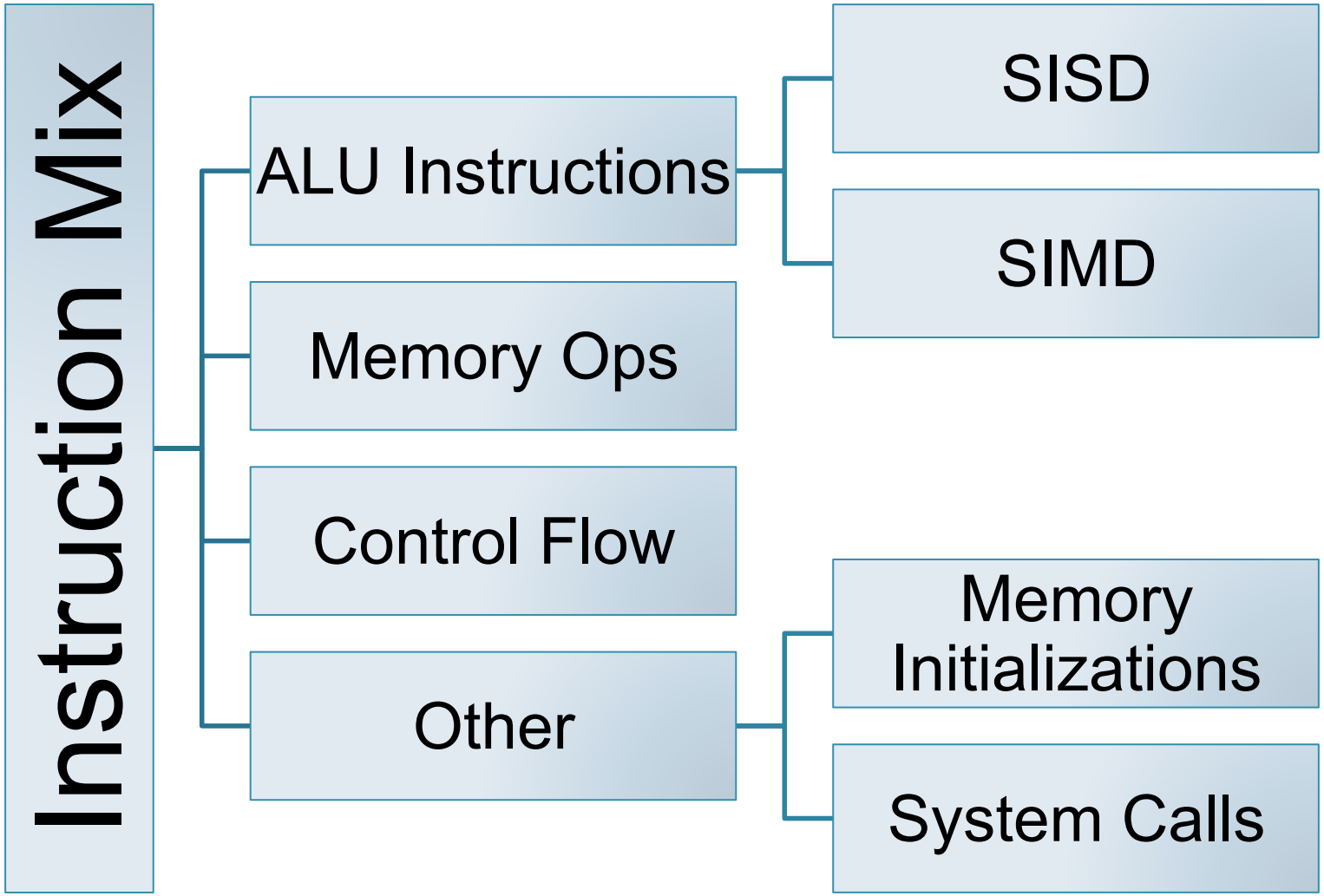
Power Measurement

- Total power measured
- ACS712 Current Sensor with 10x amplifier circuit
- 10-bit ADC to Linux PC

Timing Measurement

- Android OS – UNIX time
- Synchronization mechanism with Linux PC over adb

Application Breakdown



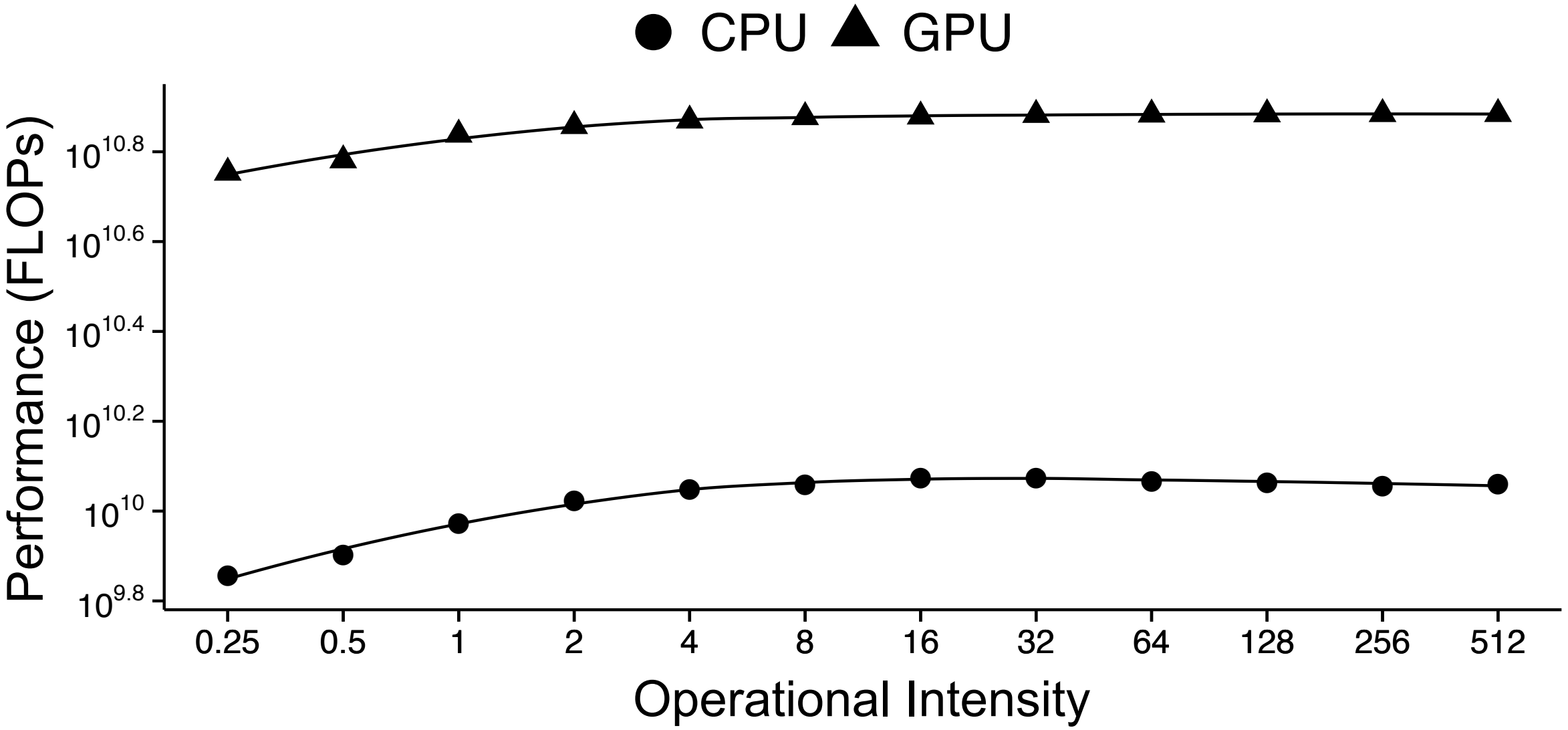
Operational Intensity & The Roofline Model [4]

- **Operational Intensity**:
  - Number of operations performed for each byte of data transferred
  - Relatively static for a given application

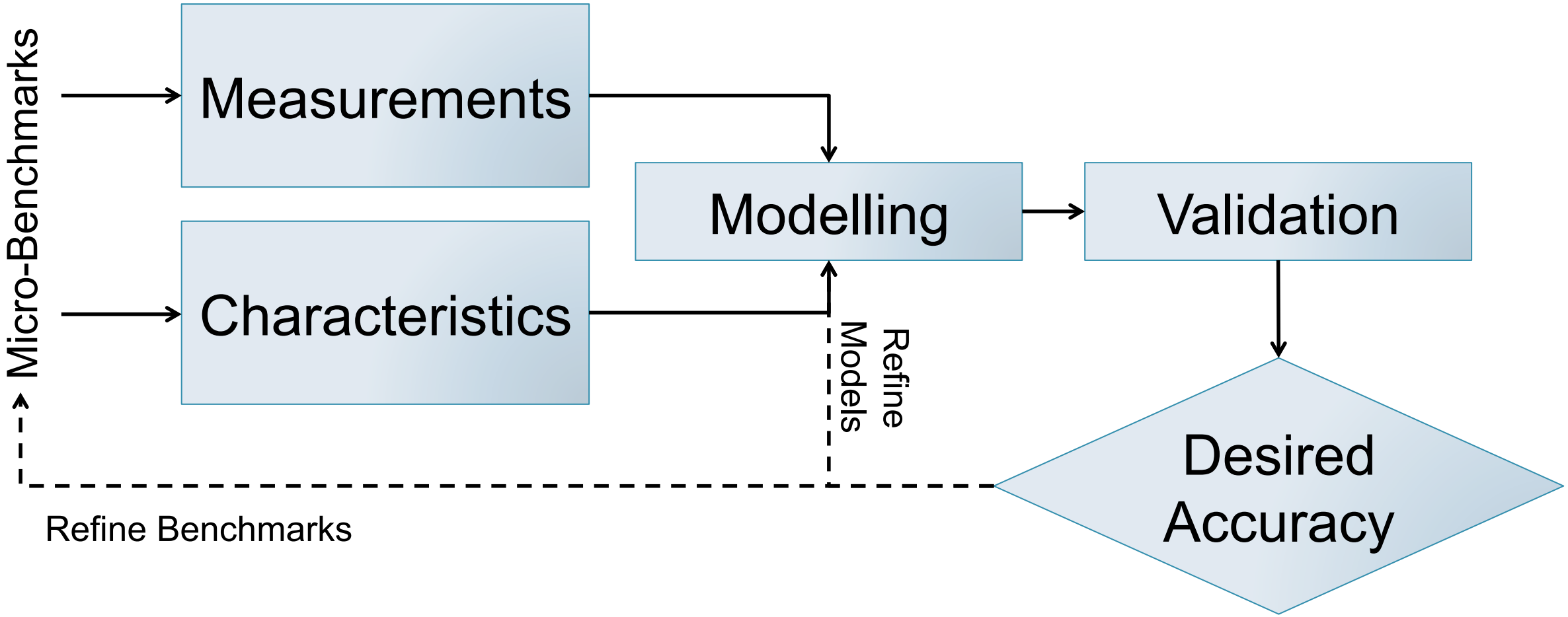
- **Roofline Model**

4. S. Williams, A. Waterman, and D. Patterson, “Roofline: An insightful visual performance model for multicore architectures,” Commun. ACM, vol. 52, no. 4, pp. 65–76, Apr. 2009.

Operational Intensity & The Roofline Model [4]



High Level Overview



Estimating From The Bottom Up

- Composable Model [1]
  - Power and Run-Time can be broken down

$$\begin{bmatrix} \textbf{Application} \\ \text{Characteristic}_1 \\ \text{Characteristic}_2 \\ \text{Characteristic}_3 \\ \dots \\ \text{Characteristic}_n \end{bmatrix} X \begin{bmatrix} \textbf{Coefficients} \\ P_1 \\ P_2 \\ P_3 \\ \dots \\ P_n \end{bmatrix} = T^*$$

*Similar for Energy (E)

1. Diop, Tahir, Natalie Enright Jerger, and Jason Anderson. "Power modeling for heterogeneous processors." Proceedings of Workshop on General Purpose Processing Using GPUs. ACM, 2014.

Behind the (Look-Up) Table

1. Create *Parametrized* Micro-Benchmarks
   - Number of operations
   - Number of bytes
2. Measure Power and Run Time
3. Solve System of Equations [1-3]

1. Diop, Tahir, Natalie Enright Jerger, and Jason Anderson. "Power modeling for heterogeneous processors." Proceedings of Workshop on General Purpose Processing Using GPUs. ACM, 2014.
2. Choi, Jee Whan, et al. "A roofline model of energy." Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on. IEEE, 2013.
3. Choi, Jee, et al. "Algorithmic time, energy, and power on candidate HPC compute building blocks." Parallel and Distributed Processing Symposium, 2014 IEEE 28th International. IEEE, 2014.

Methodology – *Parametrized* Micro-Benchmarks

- Data type
- Operation
- Vector
- Input Data Size
- Intensity

```
for each benchmark {
        for i = 1 to 3 {
        run kernel;
        sleep 2 seconds;
        }
}
```

Methodology – Example Micro-Benchmarks

```
int id = get_global_id(0);
float reg1 = input[id];
float reg2 = 0.0f;
reg2 = reg2 + reg1;
reg2 = reg2 + reg1;
reg2 = reg2 + reg1;
// ..................................
// Repeat for operations number of times
// ..................................
output[id] = reg2;
```

Linear Regression and the Timing Model

- Linear Regression performed
  - for each operation
  - at each operational intensity
  - for each device
- Output – LUT

- Time Estimate dependent on:
  - Number of operations
  - Number of bytes
  - Current availability of data

$$T = T_{compute} + T_{transfer}$$

$$T_{compute} = \sum_{ops} W_{op} \times \tau_{op}$$

$$T_{transfer} = B_{to} \times Q_{there} + B_{from} \times Q_{back}$$

Timing Model for the CPU & GPU*

Data	Operation	τ_{CPU} (ps)	τ_{GPU} (ps)
all	all	260	14
float	add	400	13
float4	add	91	13
float	fma	510	14
float4	fma	110	14
float4	multiply	140	14
float4	divide	2000	14
float4	Random access	710	150

- GPU
  - Constant for different operations
- CPU
  - Variance for different operations
  - Float4 faster than float

* Does not include transfer time

Power Model

- Similar Methodology as timing
- Key factors:
  - Energy per operation
  - Static power
  - Time taken by application

$$\frac{E}{W} = \epsilon_{op} + \pi_0 \times \frac{T}{W}.$$

Data	Operation	$\varepsilon_{CPU,op}$ (pJ)	$\varepsilon_{GPU,op}$ (pJ)	$\pi_{CPU,op}$(J)	$\pi_{GPU,op}$ (J)
all	all	360	740	25.4	27.2
float	add	1600	28	14.8	14.2
float4	add	360	260	18.0	2.6
float	fma	1100	39	21.2	26.0
float4	fma	300	37	20.1	18.6
float4	multiply	-590	67	28.1	18.7
float4	divide	-12000	20	37.5	32.1
float4	Random access	4000	2900	25.6	25.3

What's the problem with this model?

Problem:

- Which portion of the code do you apply this to?

Solution:

- Apply at a basic block granularity

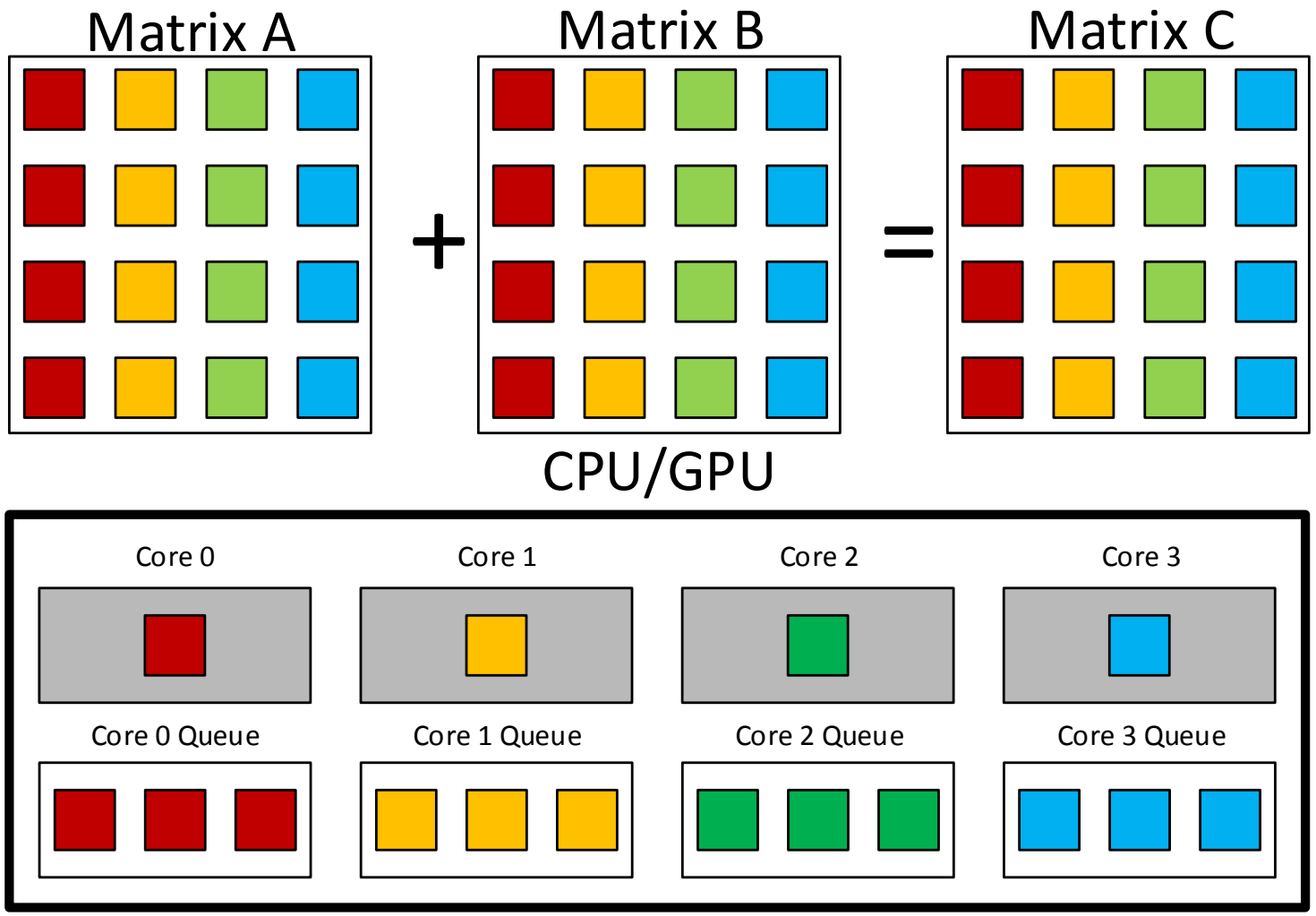
- Basic block:
  - Portion of code in a program
  - Single point of entry & single point of exit
- No control flow within basic block

Profiling the Application: Basic Block Granularity

- **Instrument** BBL & Count:
  - # runs of BBL
  - Unique bytes accessed by BBL
  - # useful operations (ALU & Memory)
- Obtain:
  - Operational intensity (ι)
  - Working set size (σ)
  - Instruction mix (for each op: v_{op})

$$T_{BBL} = max(\sum_{op} \nu_{op} \times \tau_{op}, \sigma \times \tau_{transfer})$$

Example Application: Matrix Addition

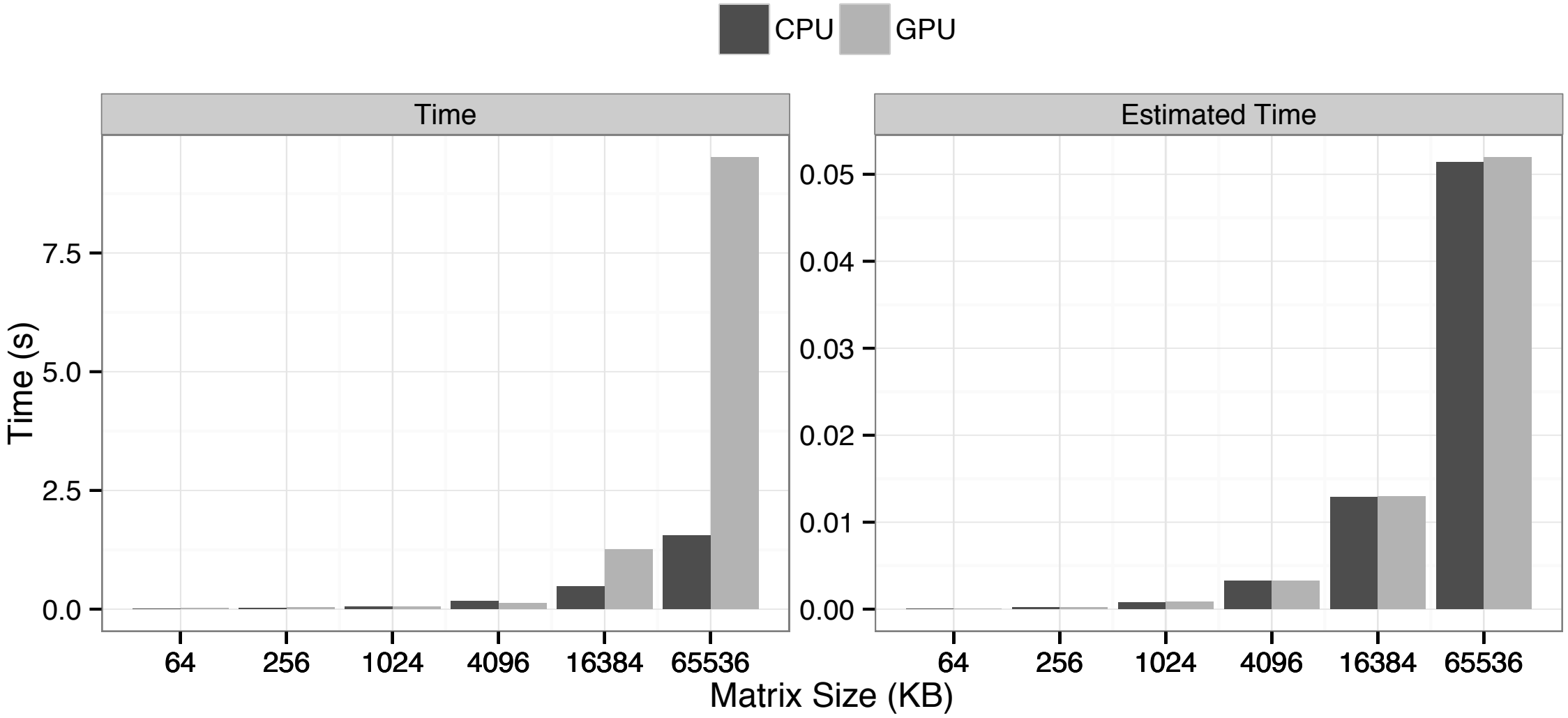


Example Application: Matrix Addition

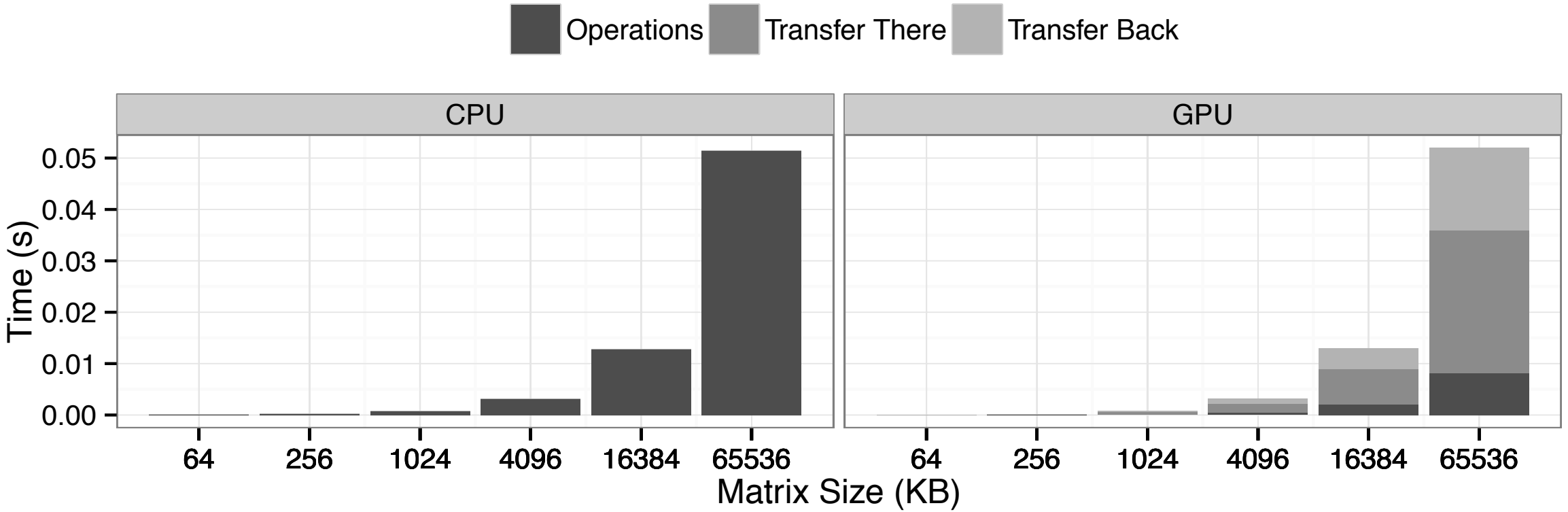
Kernel Instruction Mix

Data	*Operation*	*Count*	*Actual Operation*
float	add	1	int add
float	add	1	float add
float4	multiply	1	int multiply
float4	random access	3	float random access

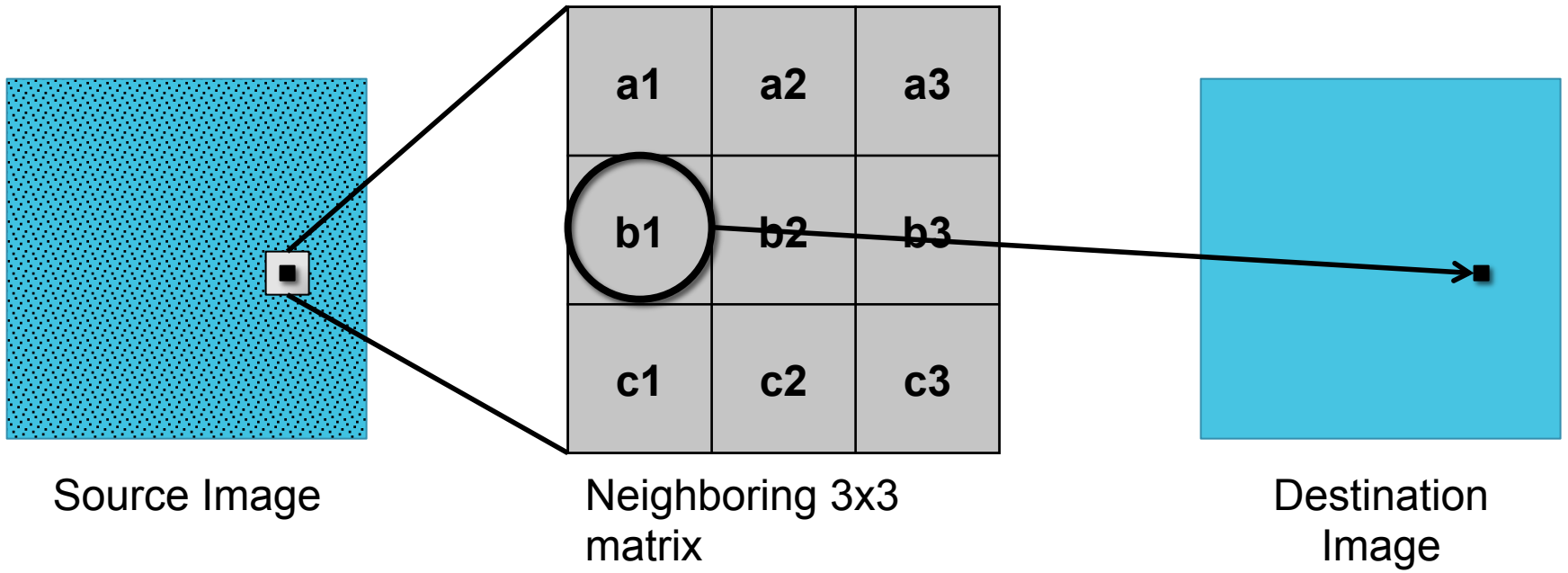
Example Application: Matrix Addition



Example Application: Matrix Addition



Sample Application: Median Filter

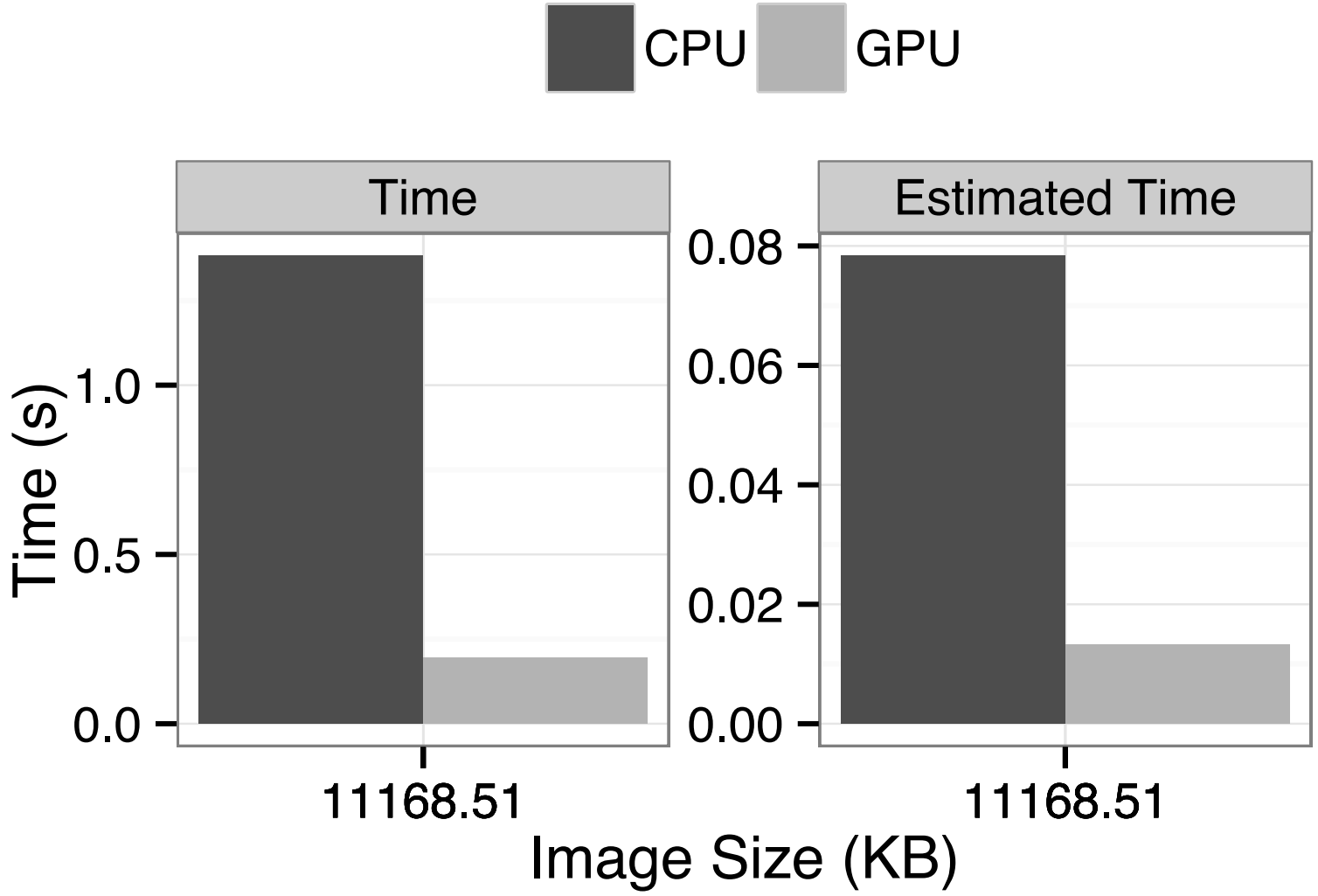


Sample Application: Median Filter

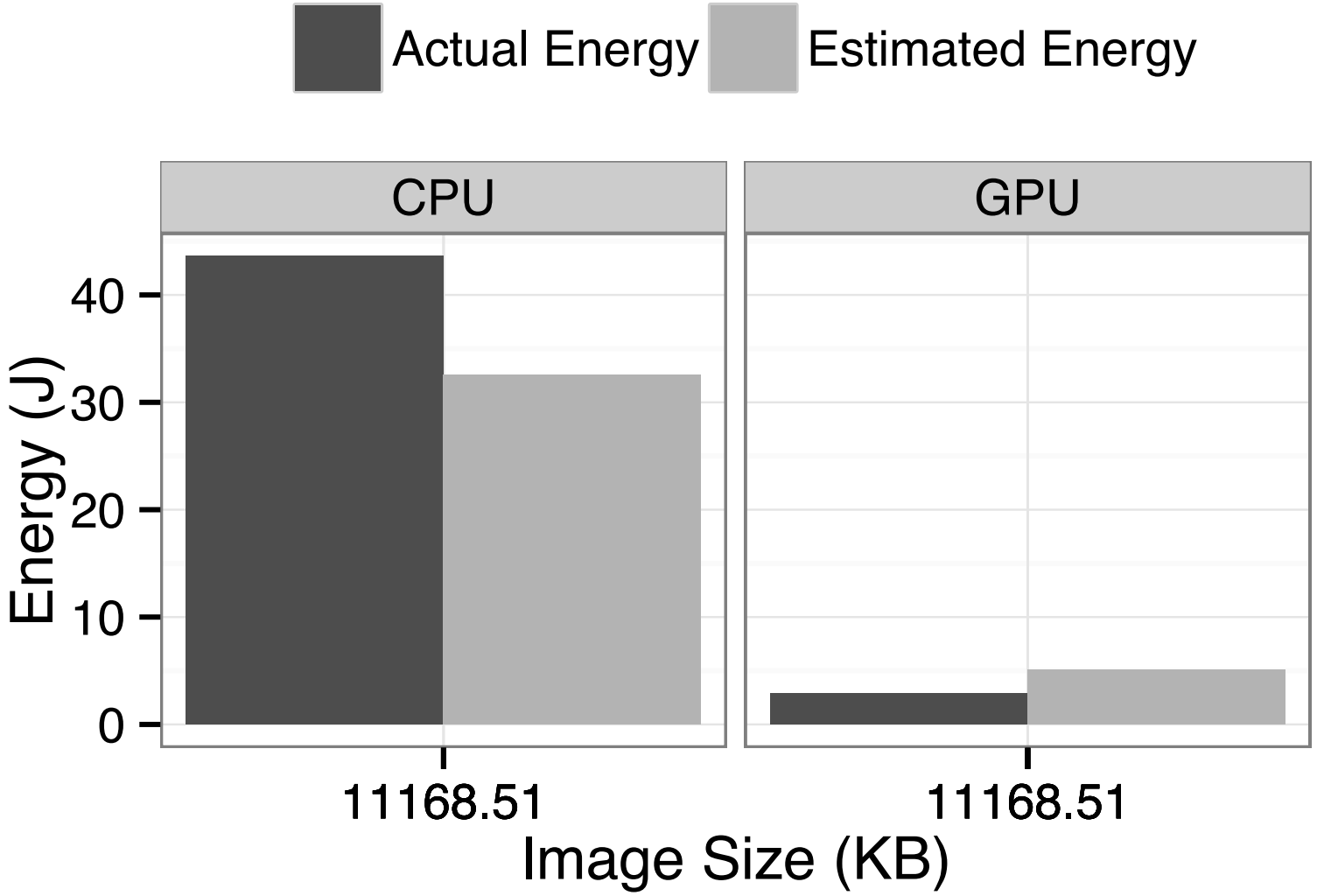
Kernel Instruction Mix

Data	*Operation*	*Count*	*Actual Operation*
float	fma	20	fmin
float	fma	20	fmax
float4	random access	10	float random access

Estimated Time



Estimated Energy



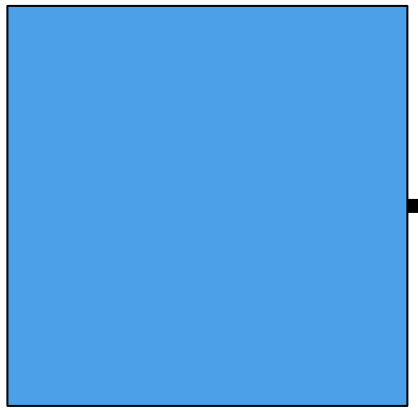
Sample Application: Sobel Image Detection

-1	0	1
-2	0	2
-1	0	1

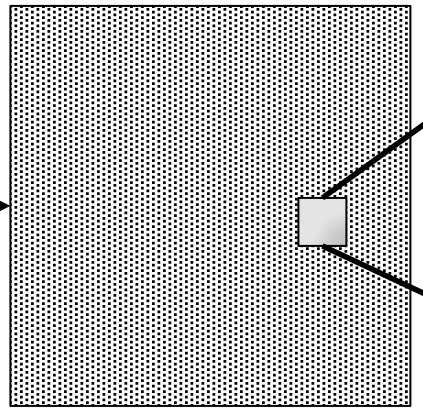
SobelX

-1	-2	-1
0	0	0
1	2	1

SobelY



Original Color Input Image



Gray Input Image

a1	a2	a3
b1	b2	b3
c1	c2	c3

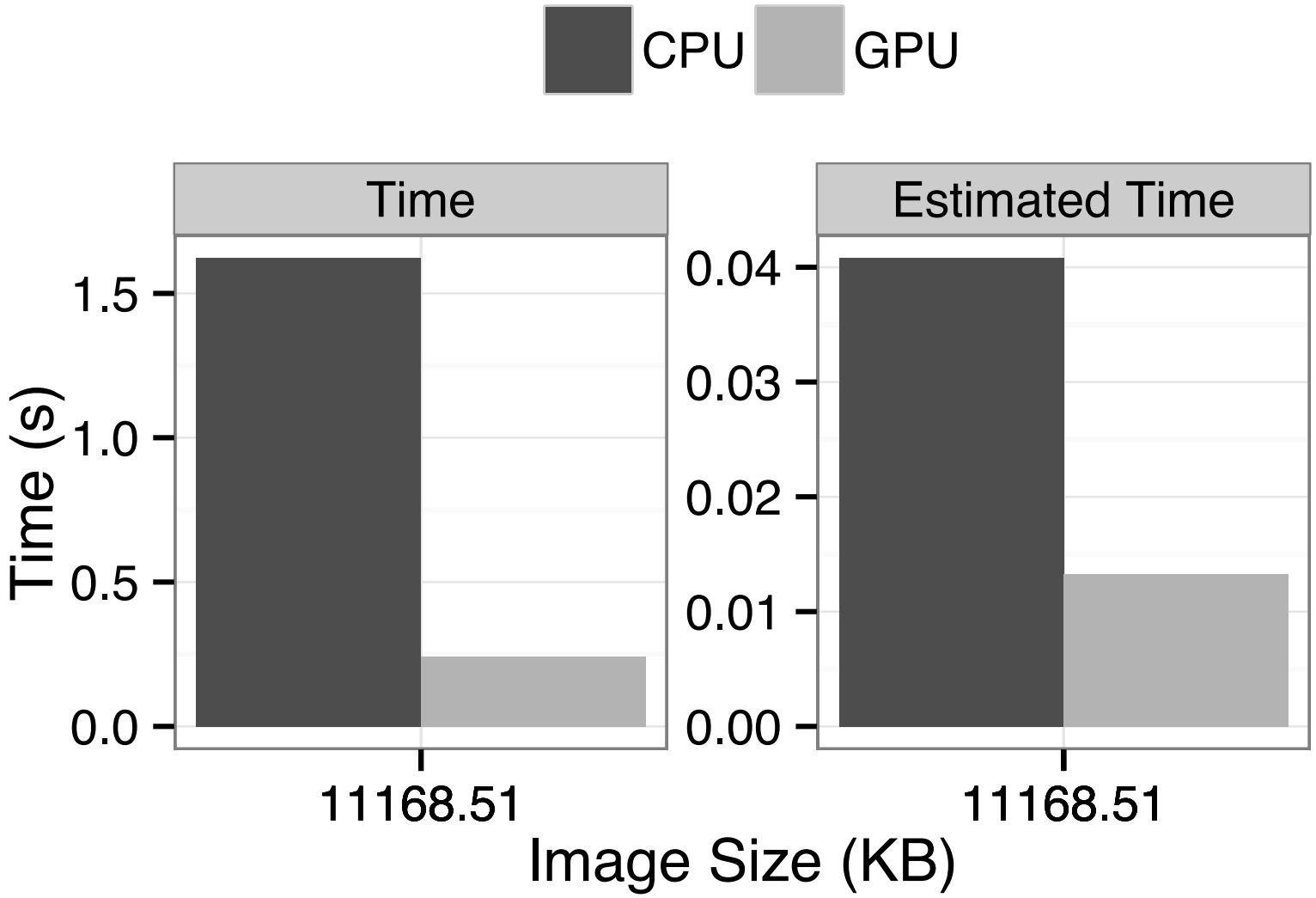
subMatA
Neighboring 3x3 matrix

Sample Application: Sobel Image Detection

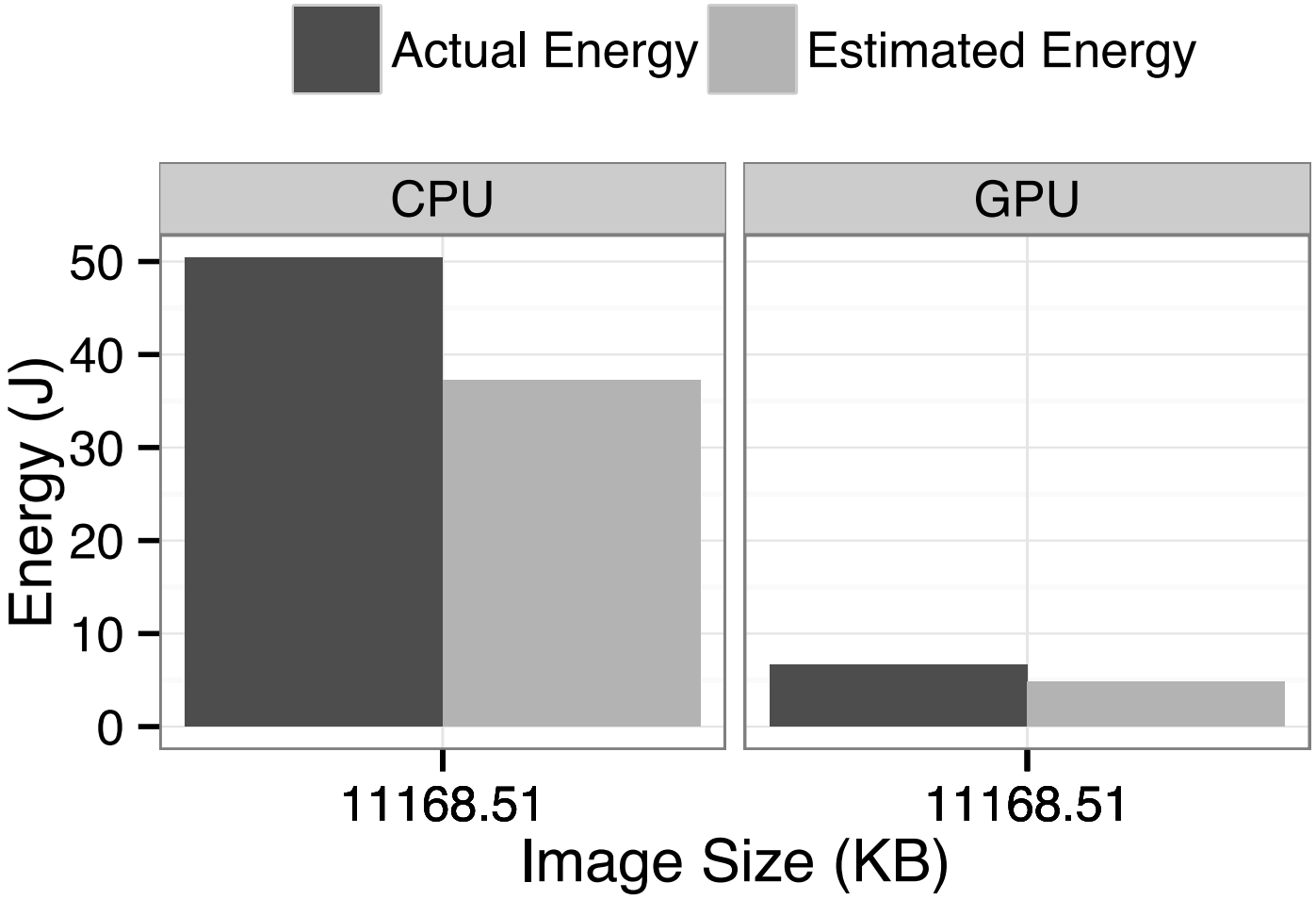
Kernel Instruction Mix

Data	*Operation*	*Count*	*Actual Operation*
float	add	12	float add
float4	multiply	7	float multiply
float4	random access	12	float random access

Estimated Time



Estimated Energy



Conclusions

- Devices have to be characterized at different operational intensities
- Generate coefficients for each instruction at different operational intensities
- Instrument and analyze applications at basic block granularity
  - Currently Working on This
- Apply generated coefficients to data obtained

Questions?

Thank you!