Performance Prediction Modeling of GPU Applications: Analytical Modeling and Machine Learning

Marcos Amarís González*
Advisor: Alfredo Goldman*
co-advisor: Raphael Y. de Camargo†

University of São Paulo*
Federal University of ABC†
São Paulo, Brasil

SC17 - November 2017

Timeline

Introduction

The prediction of application execution times over GPUs is a great challenge and is essential for efficient Job Managment Systems.

I am going to present two works

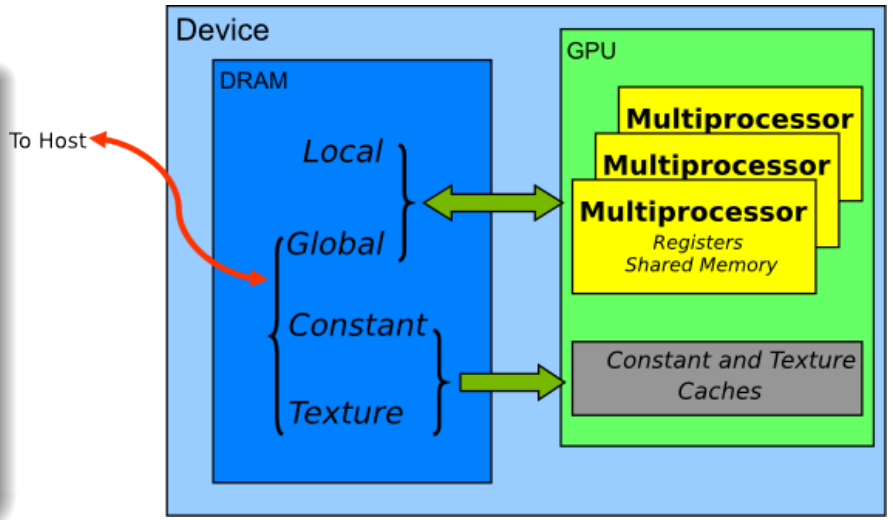
1. A Comparison of GPU Execution Time Prediction using Machine Learning and Analytical Modeling with regular applications
2. Feature Extraction and Machine Learning Techniques to Predict Execution Times of Irregular GPU Applications

3 Machine Learning Techniques

- **Linear Regression (LR)**
- **Support Vector Machines (SVM)**
- **Random Forest (RF)**

CUDA, GPUs and Memory spaces

A GPU has many processors $\mathbf{P}$, all processors have the same clock rate $\mathbf{R}$ and they are organized in Multi-processors.
A CUDA **Kernel** can be composed of thousands or millions of threads $\mathbf{t}$.



Type	On Chip	Cacheable	Instructions	Visibility	$\mathbf{g}$ Latency
Registers	Yes	No	Load/Store	*Thread*	1 cycle
Shared-L1	Yes	No	Load/Store	Block	5 cycles
Cache L2	No	Yes	Load/Store	*Kernel*	250 cycles
Global	No	Yes	Load/Store	*Kernel*	500 cycles

Table: Memory types in GPUs supported by CUDA

Bulk Synchronous Parallel Model

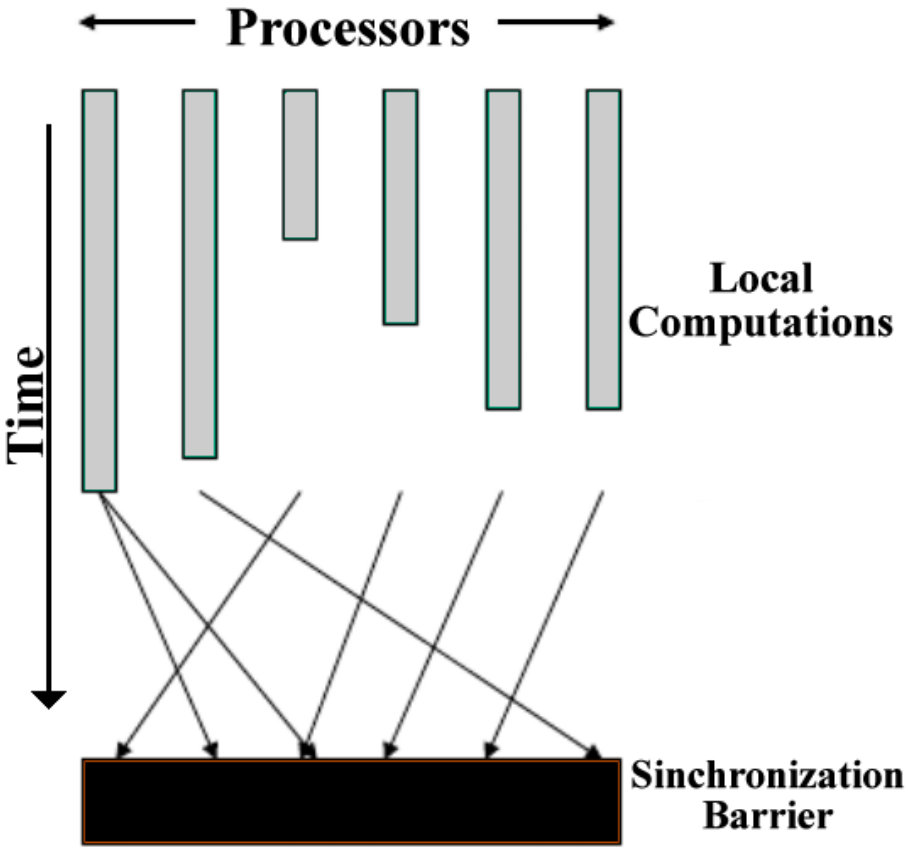


Figure: Super-step in the BSP model

The cost to execute the i-th super-step is then given by:

$$w_i + gh_i + L \tag{1}$$

The total execution time of the application is given by:

$$T = W + gH + LS \tag{2}$$

Proposed Analytical Model

$$\mathbf{T_k} = \frac{\mathbf{t} \cdot (\mathbf{Comp} + \mathbf{Comm_{SM}} + \mathbf{Comm_{GM}})}{\mathbf{R} \cdot \mathbf{P} \cdot \lambda} \tag{3}$$

$$\mathbf{Comm_{GM}} = (\mathbf{ld_1} + \mathbf{st_1}) \cdot \mathbf{g_{GM}} \tag{4}$$

$$\mathbf{Comm_{SM}} = (\mathbf{ld_0} + \mathbf{st_0}) \cdot \mathbf{g_{SM}} \tag{5}$$

$\mathbf{comp}$, $\mathbf{ld_0}$, $\mathbf{st_0}$, $\mathbf{ld_1}$ and $\mathbf{st_1}$ are obtained from the source code or profiling process.

Divergence, optimizations in the communication and differences between architecture or GPUs are adjusted by one parameter, λ.

[1] M. Amaris, D. Cordeiro, A. Goldman, and R. Y. Camargo, "A simple bsp-based model to predict execution time in gpu applications," in *High Performance Computing (HiPC), 2015 IEEE 22nd Int'l Conference on*, December 2015, pp. 285–294.

GPUs of the Testbed

Model	C.C.	Memory	Bus	Bandwidth	L2	Cores/SM	Clock
GTX-680	3.0	2 GB	256-bit	192.2 GB/s	0.5 M	1536/8	1058 Mhz
Tesla-K40	3.5	12 GB	384-bit	276.5 GB/s	1.5 MB	2880/15	745 Mhz
Tesla-K20	3.5	4 GB	320-bit	200 GB/s	1 MB	2496/31	706 MHz
Titan	3.5	6 GB	384-bit	288.4 GB/s	1.5 MB	2688/14	876 Mhz
Quadro K5200	3.5	8 GB	256-bit	192.2 Gb/s	1 MB	2304/12	771 Mhz
Titan X	5.2	12 GB	384-bit	336.5 GB/s	3 MB	3072/24	1076 Mhz
GTX-970	5.2	4 GB	256-bit	224.3 GB/s	1.75 MB	1664/13	1279 Mhz
GTX-980	5.2	4 GB	256-bit	224.3 GB/s	2 MB	2048/16	1216 Mhz
Pascal-P100	6.0	12 GB	4096	224.3 GB/s	1.75 MB	1664/13	1279 Mhz

Table: Hardware specifications of the GPUs in the testbed

Algorithm Testbed

9 different regular CUDA kernels

- Matrix Multiplications in 4 different optimizations:
  - Global Memory - MMGU
  - Global Memory with coalesced accesses - MMGC
  - Global and Shared Memory - MMSU
  - Global and shared Memory with coalesced accesses - MMSC
- Matrix Addition in 2 different optimizations:
  - Global Memory - MAU
  - Global Memory with coalesced accesses - MAC
- Dot Product - dotP
- Vector Addition - vAdd
- Maximum Subarray Problem - MSA

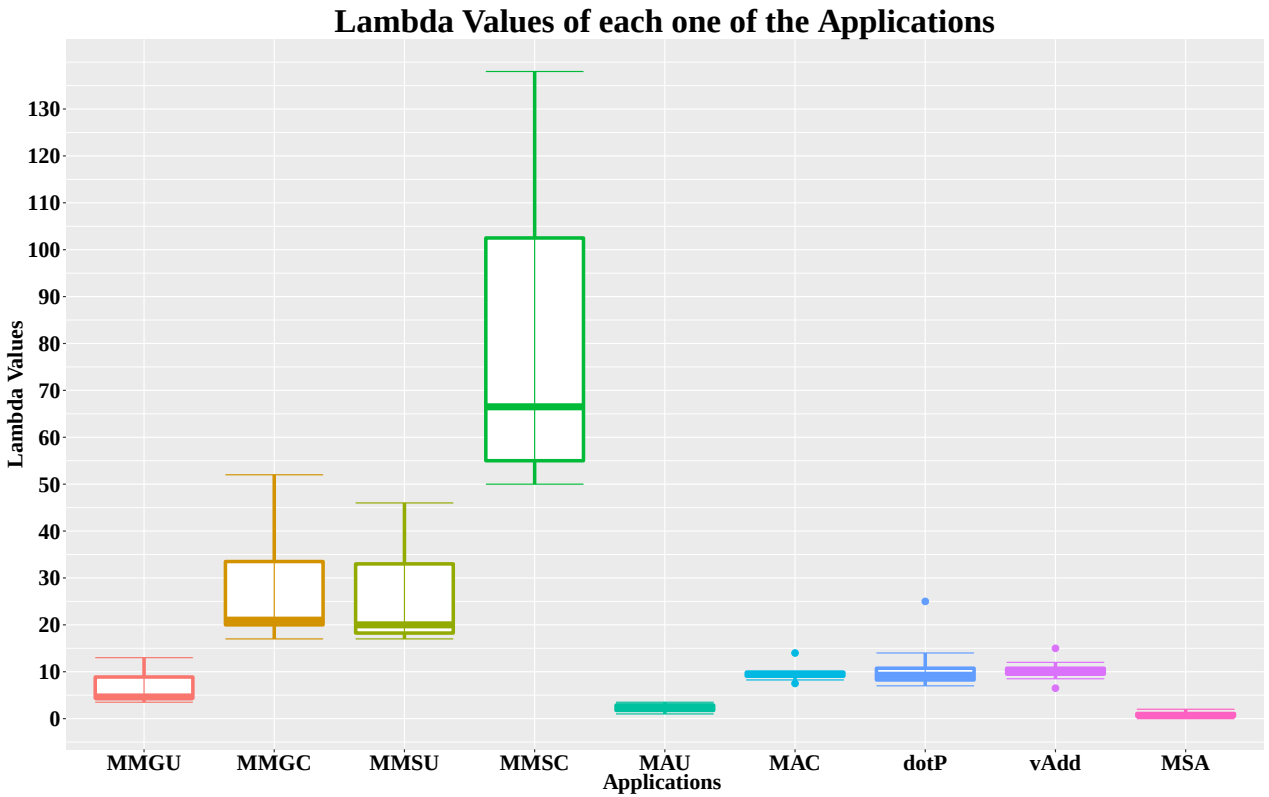
Features of the Machine Learning Techniques

13 features were used to feed the Machine learning Techniques.

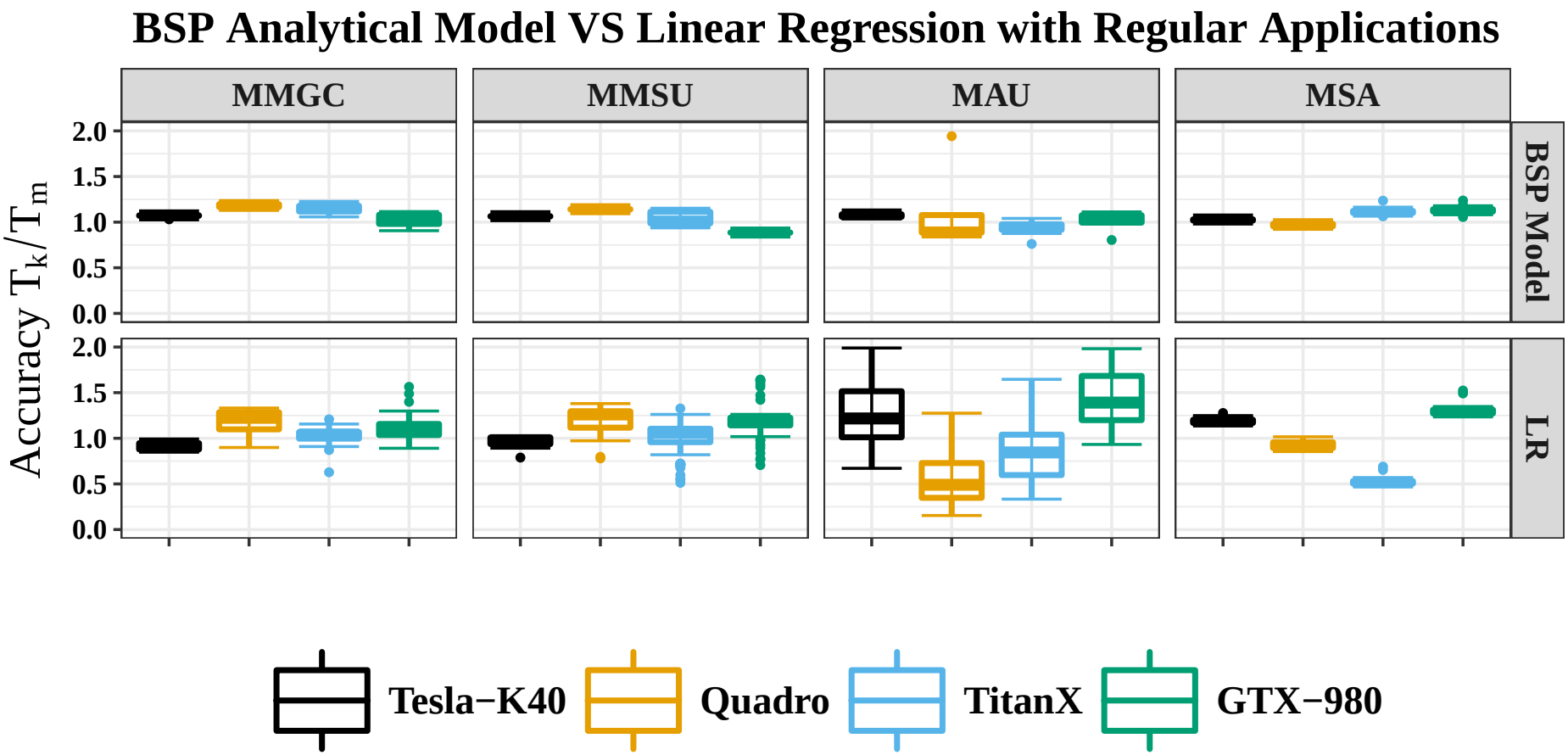
Feature	Description
num_of_cores	Number of cores per GPU
max_clock_rate	GPU Max Clock rate
Bandwidth	Theoretical Bandwidth
Input Size	Size of the problem
totalLoadGM	Load transaction in Global Memory
totalStoreGM	Store transaction in Global Memory
TotalLoadSM	Load transaction in Shared Memory
TotalStoreSM	Store transaction in Global Memory
FLOPS SP	Floating operation in Single Precision
BlockSize	Number of threads per blocks
GridSize	Number of blocks in the kernel
No. threads	Number of threads in the applications
Achieved Occupancy	Ratio of the average active warps per active cycle to the maximum number of warps ed on a multiprocessor.

Use Cases of the Analytical Model

Par.	Matrix Multiplication				Matrix Addition		vAdd	dotP	MSA
	MMGU	MMGC	MMSU	MMSC	MAU	MAC			
comp	$N\cdot$ FMA				$1 \cdot 24$			$1 \cdot 96$	$(N/t) \cdot 100$
ld_1	$2 \cdot N$				2			2	N/t
st_1	1				1			1	5
ld_0	0		$2 \cdot N$		0			0	N/t
st_0	0		1		0			$1 + log(t)$	5



Results Machine Learning VS Analytical Model



Algorithm Testbed

- Correlation analysis
- Hierarchical clustering analysis.

Each cleaned sample of the dataset resulted in 85 kernel features plus 11 GPU architecture features.

Application	Berkeley Dwarf	Param.	Kernels	Samples
Back Propagation (BCK)	Unstructured Grid	1 - [57]	layerforward adjust-weights	57
Gaussian Elimination (GAU)	Dense Linear Algebra	1 - [32]	Fan1 Fan2	34800
Heart Wall (HWL)	Structured Grid	1 - [84]	heartWall	5270
Hot Spot (HOT)	Structured Grid	2 - [5,4]	calculate-temp	396288
Hot Spot 3D (H3D)	Structured Grid	2 - [3,10]	hotspotOpt1	150150
LU Decomposition (LUD)	Dense Linear Algebra	1 - [32]	diagonal perimeter internal	8448 8416 8416

Table: Rodinia applications used in the experiments

[1] elapsed_cycles_sm
[2] gld_inst_32bit
[3] gst_inst_32bit
[4] inst_executed
[5] inst_issued1
[6] active_cycles
[7] multiprocessor_activity
[8] global_load_transactions
[9] global_store_transactions
[10] device_memory_read_transactions
[11] l2_read_transactions
[12] l2_write_transactions
[13] issued_control.flow_instructions
[14] executed_control.flow_instructions
[15] issued_load.store_instructions
[16] executed_load.store_instructions
[17] issue_slots
[18] control.flow_instructions
[19] load.store_instructions
[20] misc_instructions

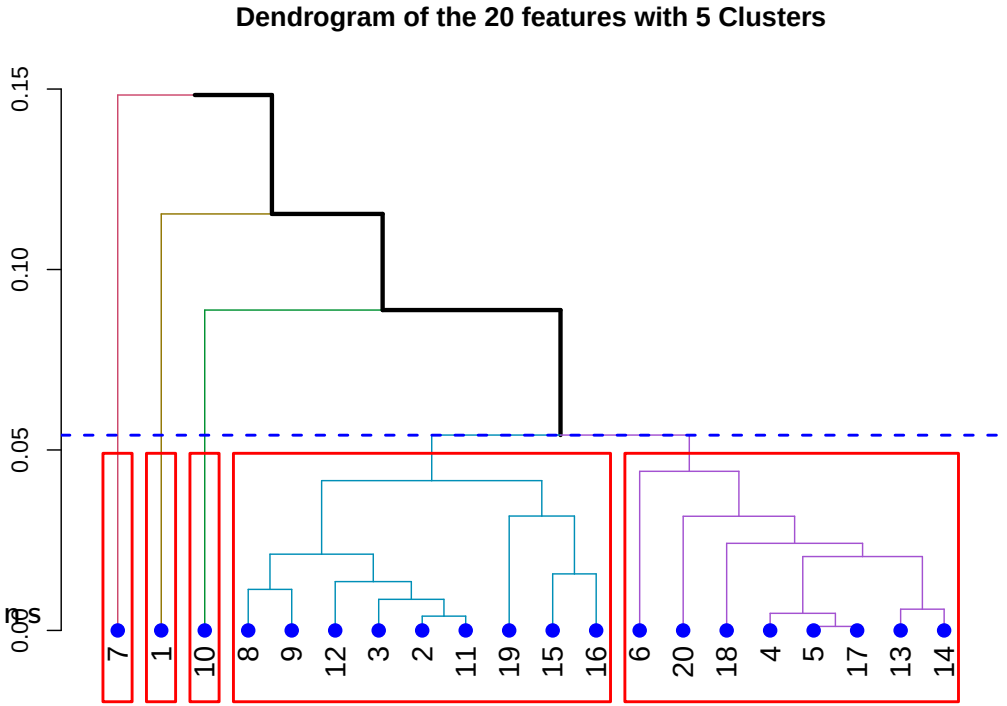


Figure: Dendrogram with 5 clusters to select 1 features from each one

2 Contexts were tested with each ML

1. GPUs: we evaluated if the ML algorithm could predict the execution time over a previously unseen GPU
2. Kernels: in this context we predict the execution time of a previously unseen CUDA kernel.

Results

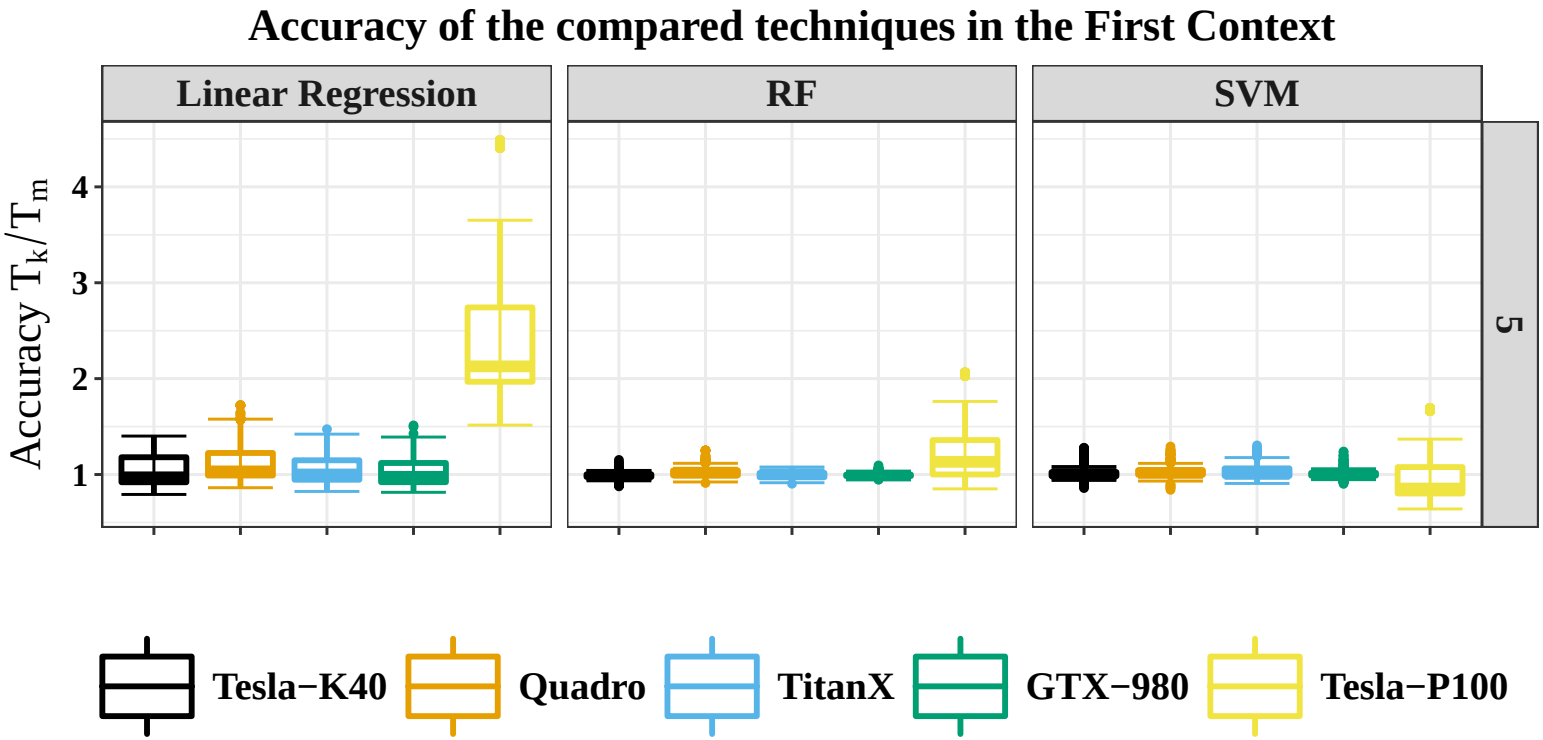


Figure: Accuracy of the first context with all the machine learning techniques used over 5 target GPUs

Results

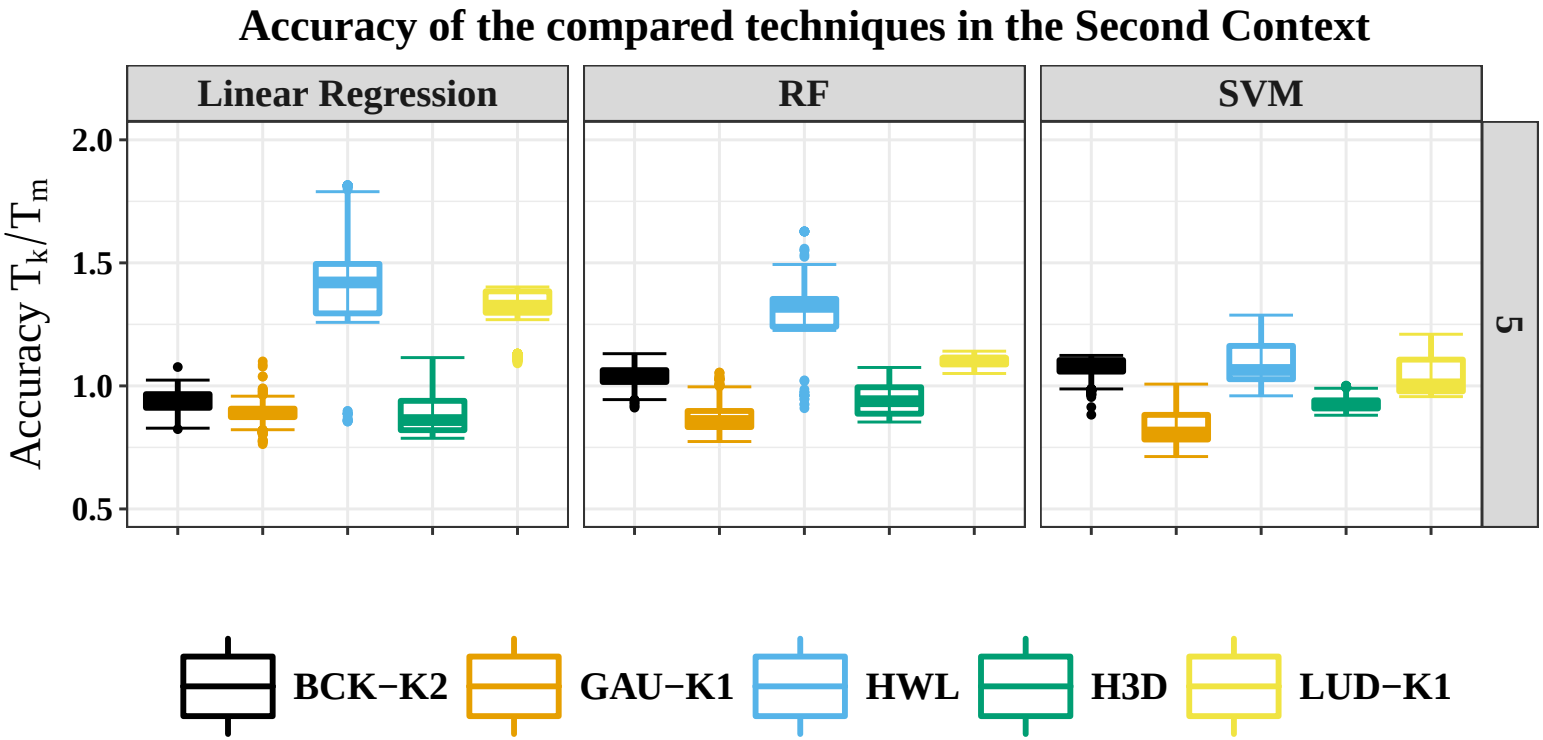


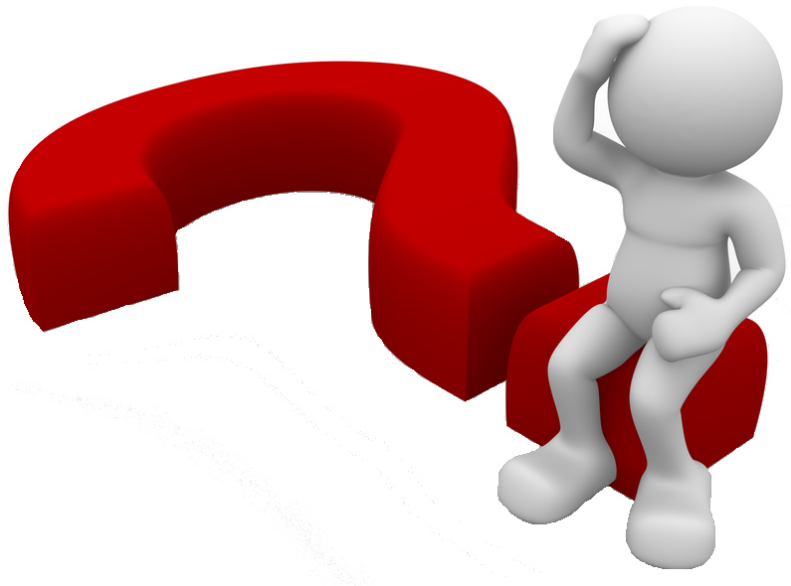
Figure: Accuracy of the second context with all the machine learning techniques used over 5 target irregular kernels

Conclusions

- Performance of regular GPU applications can be reasoneably predicted with simple analytical models.
- Few parameters about communication and/or computation of the application are enough for these applications.
- Machine learning techniques worked better for Irregular Applications.
- Linear models are enough if the features are linearized.
- Predictions with high accuracy are possible with an optimum statistical process of feature extraction.

FAPESP
SÃO PAULO RESEARCH FOUNDATION

Thanks for your attention



Links of the work: `https://github.com/marcosamaris/`