A tool for Bottleneck analysis and Performance Prediction for GPU-accelerated Applications

S. Madougou, A. Varbanescu, C. de Laat and R. van Nieuwpoort

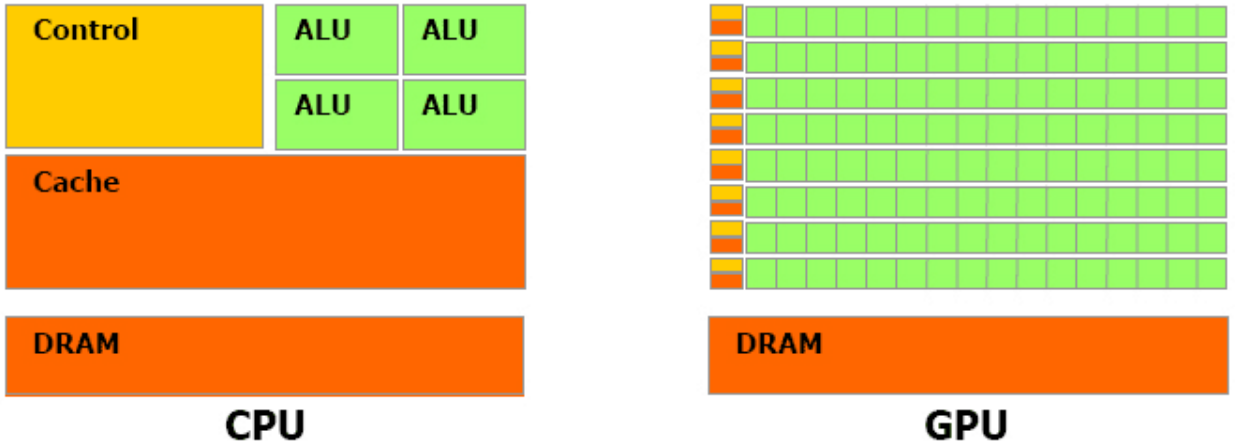
Universiteit van Amsterdam, NL

May 23, 2016

Motivation

- Heterogeneous computing emerging as a way to computing efficiency
  - parallel design and programming are the trends
- Hard to get optimal performance on heterogeneous architectures
- Need for tools for understanding performance on heterogeneous architectures
  - Different approaches: profilers, simulators, performance models

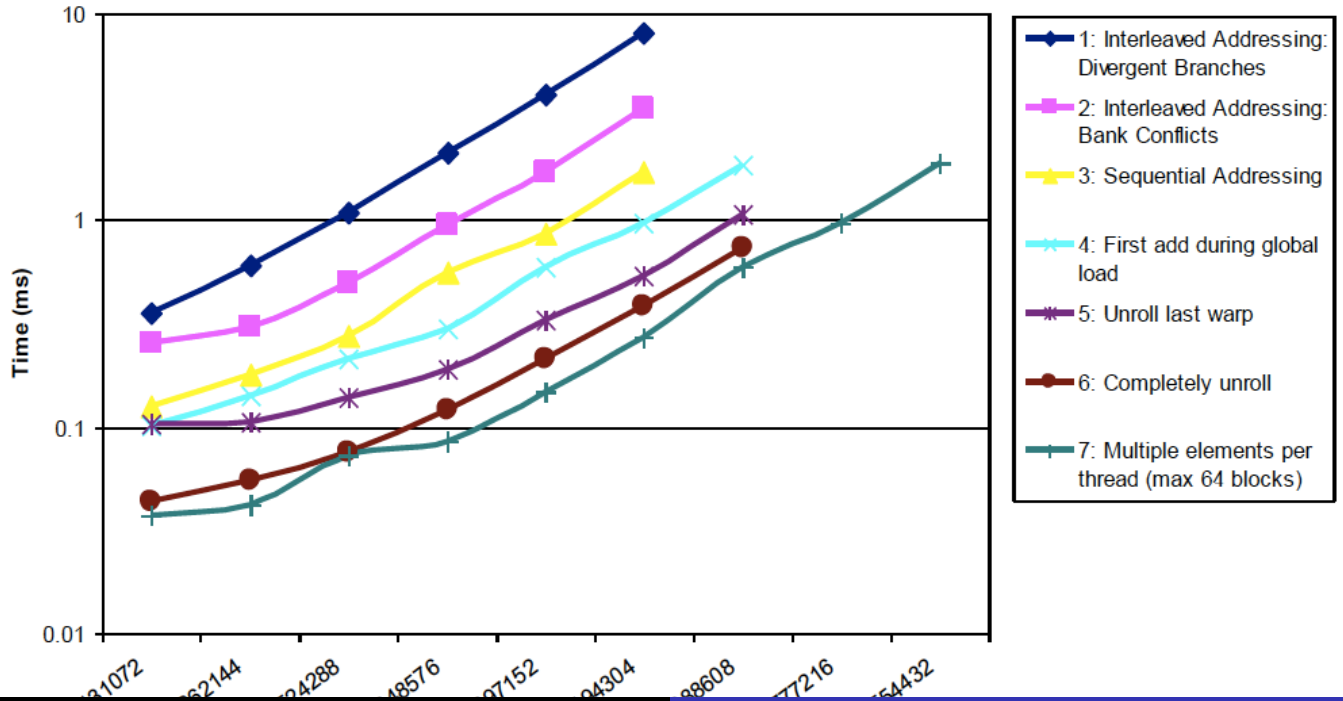
Why GPUs?



- For their popularity
  - Higher pure computing horse-power than CPUs
  - Performance enhancement for more and more applications
- For the challenge of getting performance on GPUs
  - Fitness to data parallel and specific programing models
  - Exploration of a large optimization space (via tuning, etc)

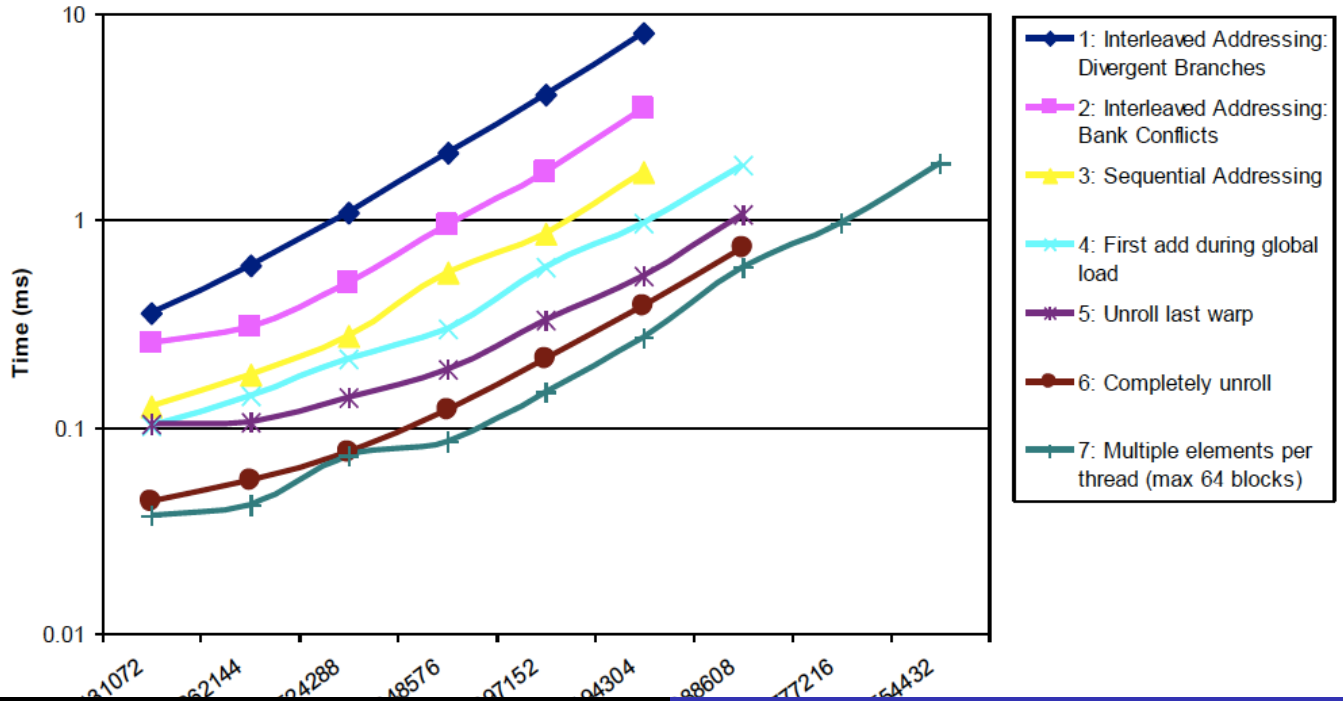
Modelling performance, why?

- Scaling behavior through application parameter space
- Scaling behavior through hardware parameter space
- Performance bottlenecks
- Performance limiting factors



Modelling performance, why?

- Scaling behavior through application parameter space
- Scaling behavior through hardware parameter space
- **Performance bottlenecks**
- **Performance limiting factors**



Performance modelling (PM)

Not the first, certainly not the last.

Many different approaches:

- Simulation
- Analytical
- Statistical/ML
- Measurements

Current approaches present many shortcomings[1]:

[1] Madougou et al., An empirical evaluation of GPGPU performance models, Hetero-Par 2014.

Main PM Obstacles

- Complexity
- Requirement for detailed hardware knowledge
- Dependence on hardware or application
- Requiring user intervention
- Simulation/benchmarking is time consuming

Machine Learning Trade-Offs

Pros:

- Doesn't require hardware understanding
- Doesn't require software understanding
- Sparse set of measurements is sufficient
- Easily publishable buzzword!

Cons:

- Don't know **what** is learned
- Hard to know where bottlenecks are
- Prone to overfitting

Some Observations

All platforms expose hardware performance counters (PCs)

Performance data is *easy* to extract but *hard* to interpret

PC Measurements and Metrics

- PC: special-purpose register built into a processor to store the count of an hardware event
- PCs allow to establish correlation between application code and its mapping to the hardware
- Choice of tool for PC counting and derived metrics
  - Low level: PAPI, vendor-specific, high level: TAU, HPCToolkit, Score-P, etc
  - LIKWID (CPU), nvprof (GPU) used currently

Some PCs and Metrics for CPU (Intel Nehalem)

metric	meaning	group
inst_per_br	*instructions per branch*	BRANCH
br_rate	*branch rate*	BRANCH
mem_data_vol	*volume of data read/write in GByte*	MEM
SPFlops	*single precision arithmetic performance*	FLOPS_SP
SPMUOPS	*single precision vectorization performance*	FLOPS_SP
PMUOPS	*vectorization performance*	FLOPS_SP
L1_miss_ratio	*L1 data cache miss ratio*	CACHE
dcache_miss_rate	*L1 data cache miss rate*	CACHE
L3_data_vol	*data volume between L2 and L3*	L3
L2S_ratio	*loads to stores ratio*	DATA
L1DTLB_miss_rate	*L1 data TLB miss rate*	TLB
cpi	*cycles per instruction*	Always
br_mispred_rate	*branch misprediction rate*	BRANCH

Some PCs and Metrics for GPU (CUDA CC 2.0)

counter	meaning
shared_replay_overhead	*average number of replays due to shared memory conflicts for each instruction executed*
shared_load|store	*number of executed shared load (store) instructions, increments per warp on a multiprocessor*
inst_replay_overhead	*average number of replays for each instruction executed*
l1_global_load_hit	*number of cache lines that hit in L1 for global memory load accesses*
l1_global_load_miss	*number of cache lines that miss in L1 for global memory load accesses*
gld_request	*number of executed global load instructions increments per warp on a multiprocessor*
gst_request	*similar to gld_request for store instructions*
global_store_transaction	*number of global store transactions increments per transaction which can be 32,64,96 or 128 bytes*
gld_requested_throughput	*requested global memory load throughput*
achieved_occupancy	*ratio of average active warps per active cycle to the maximum number of warps per SM*
l2_read_throughput	*memory read throughput at L2 cache*
l2_write_transactions	*memory write transactions at L2 cache*
ipc	*number of instructions executed per cycle*

Counter Behavior vs Performance - CPU[2]

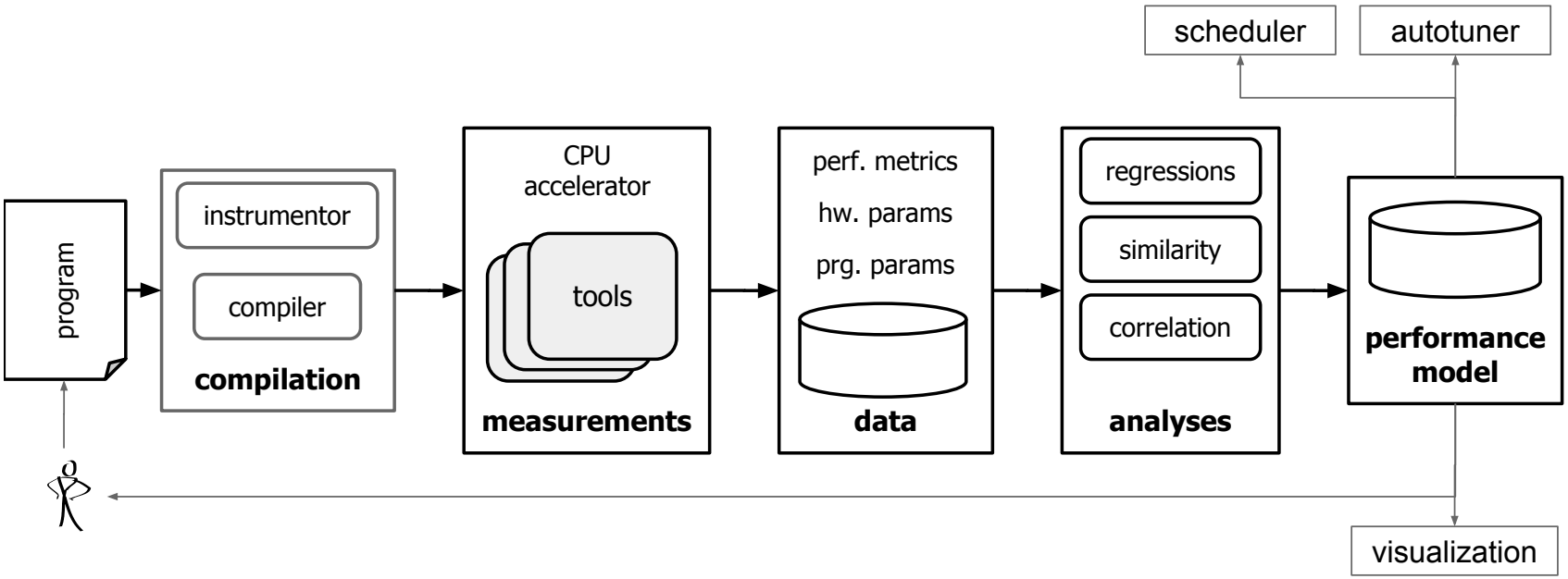
pattern	**signature**	
	performance behavior	**HPM** (group)
load imbalance	saturating speedup	different counts of instructions retired or FP operations among cores (FLOPS_DP,FLOPS_SP)
memory BW saturation	saturating speedup across cores sharing a memory interface	memory BW comparable to peak memory BW (MEM)
strided memory access	large discrepancy between between simple BW-based model and actual performance	low BW utilization despite LD/ST domination, low cache hit ratios, frequent evicts/replacements (CACHE,DATA,MEM)
bad instruction mix	performance insensitive to problem sizes fitting into different cache levels	large ratio of inst. retired to FP inst. if FP, many cycles per inst. if long-latency arithmetic, scalar instructions dominating in data-parallel loops (FLOPS_DP,FLOPS_SP,CPI)
limited instruction throughput	large discrepancy between actual performance and simple predictions based on max FLOP/s or LD/ST throughput	low CPI near theoretical limit if instruction throughput is the problem, static code analysis predicting large pressure on single execution port (FLOPS_DP,FLOPS_SP,CPI)
synchronization overhead	speedup going down as more cores are added, no speedup with small problem sizes, core busy but low FP	large non-FP instruction count (growing with number of cores used), low CPI (FLOPS_DP,FLOPS_SP,CPI)
false cache line sharing	very low speedup or slowdown even with small core counts	frequent (remote) evicts (CACHE)

[2] J. Treibig et al., Best practices for HPM-assisted performance engineering on modern multicore processors, CoRR, 2012

Counter Behavior vs Performance - GPU

performance issue	**counter set**	**values and trends**	**message**
scattered access pattern	gld_request ,l1_global_load_miss l1_global_load_hit,gst_request l1_global_store_transaction gld|gst_transactions_per_request	memory instruction count $\ll$ memory transaction count kernel throughput $\ll$ hardware throughput	coalesce access addresses, non-caching loads or textures
insufficient mem. concurrency	gld_throughput,gst_throughput achieved_occupancy	effective $\ll$ theory low	increase occupancy many elements / thread
instruction serialization	inst_executed,inst_issued	executions $\ll$ issues	see next 2 items
shared bank conflicts	l1_shared_bank_conflict shared_load,shared_store	conflicts $>$ loads+stores	use padding
warp divergence	divergent_branch,branch or branch_efficiency	divergent branches $\approx$ branches	data or thread index rearrangement
limited inst. throughput	ipc	low compared to theory	use intrinsics
insufficient parallelism	achieved_occupancy	low	adjust exec. config.
synchronization overhead	stall_synch	high	code rearrangement
latency	gld|gst_throughput,ipc	both mem. and inst. throughput $\ll$ theory	msgs for insuf. mem. and inst. throughput
register spilling	l1_local_load_miss,local_load local_store,gld_request gst_request,inst_issued	compare to total instructions compare to global memory instructions	increase register limit per thread, increase L1

BlackForest[3] Architecture



BlackForest (BF) Approach

Goal: *explain* performance behavior and *predict* performance

Main approach: regression by random forest[4]

- black-box approach
- predictive power and high accuracy of the predictions
- variable importance feature

Model simplification: model important variables in terms of problem/hardware parameters: (g)lm, MARS

Additional techniques for model improvement and ease of interpretation: PCA, clustering

[4] L. Breiman, Random forests, *Machine Learning*, 2001

Random Forest Model Construction

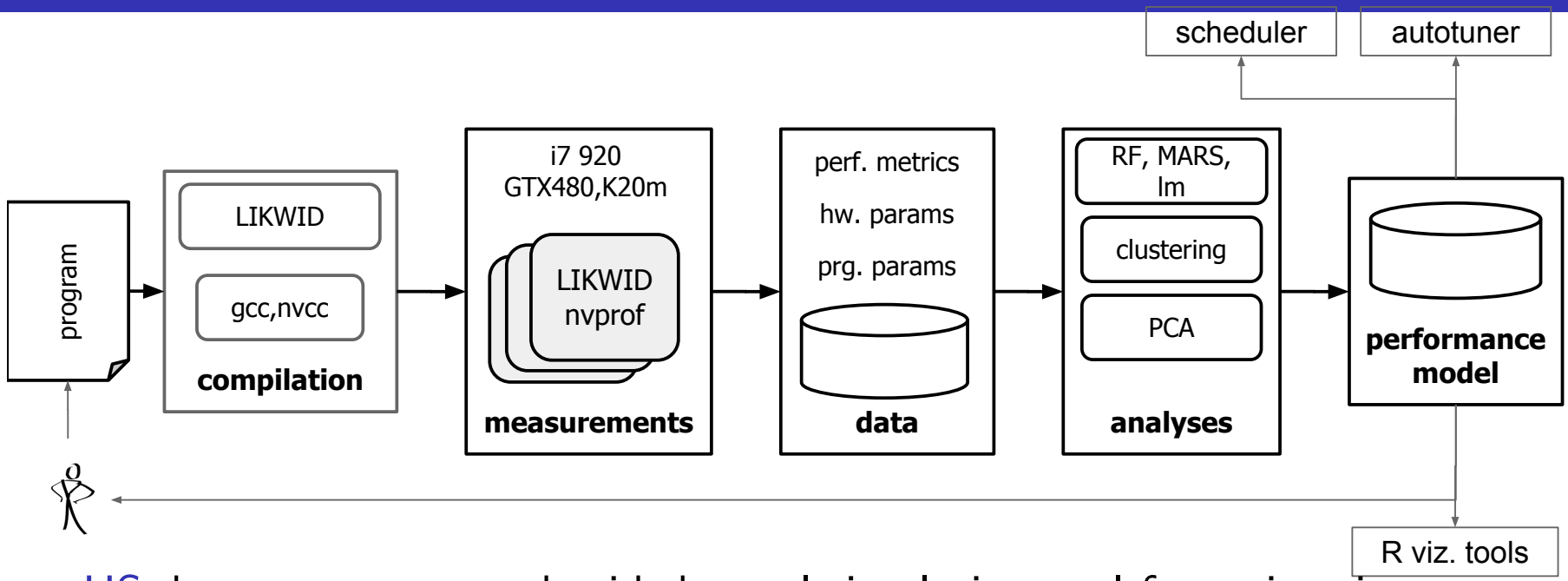
Steps:

1. Select random sample from training set (Bagging)
2. Select random sample from PCs
3. Construct regression tree to fit data
4. Repeat to build forest of trees
5. Average predictions of all trees together

Remarks:

- Randomness reduces overfitting
- Identifies important performance counters!

BlackForest Measurements



HS hotspot, structured grid thermal simulation tool for estimating processor temperature, memory intensive, latency limited

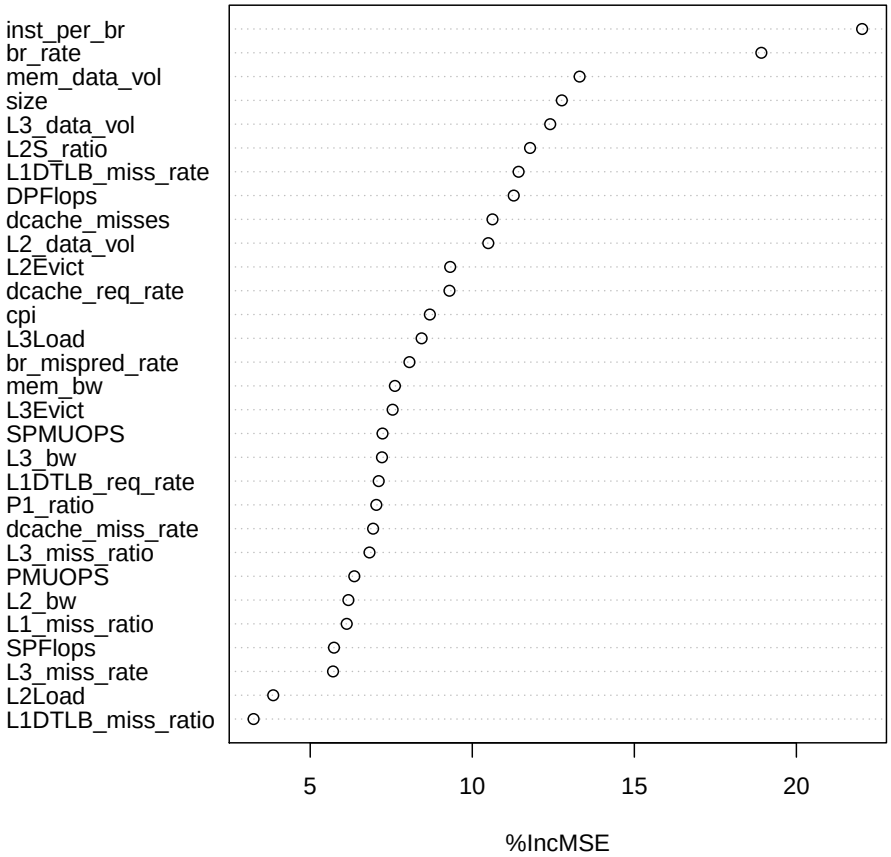
NW Needleman-Wunsch, nonlinear global optimization method for DNA sequence alignment, memory intensive, bandwidth limited

MM Matrix Multiply, linear algebra primitive used in many numerical algorithms, memory intensive, bandwidth limited

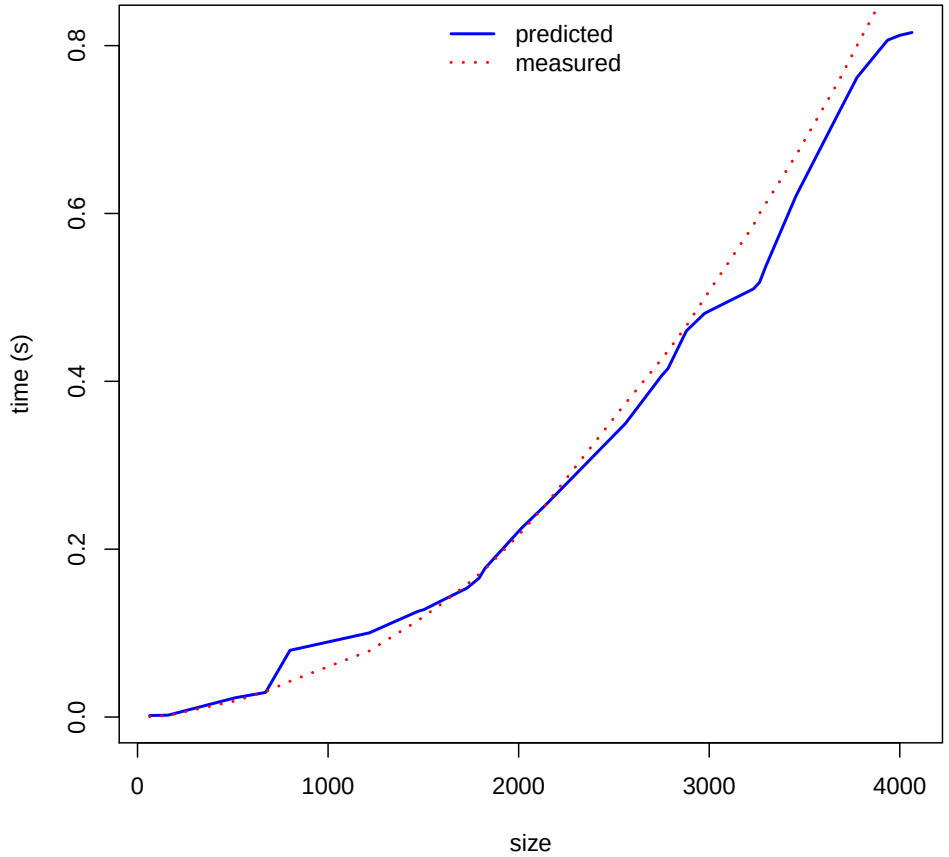
Experimental Setup

- Experimental data collection: several application runs with different problem sizes
- Response and predictors specification, model building and training
  - Sampling UAR of 20% data for test
- Use variable importance to simplify the model if possible
- Otherwise, PCA and/or clustering to try to simplify
- Control goodness-of-fit by R-squared (>95%)

HS Problem Scaling on CPU

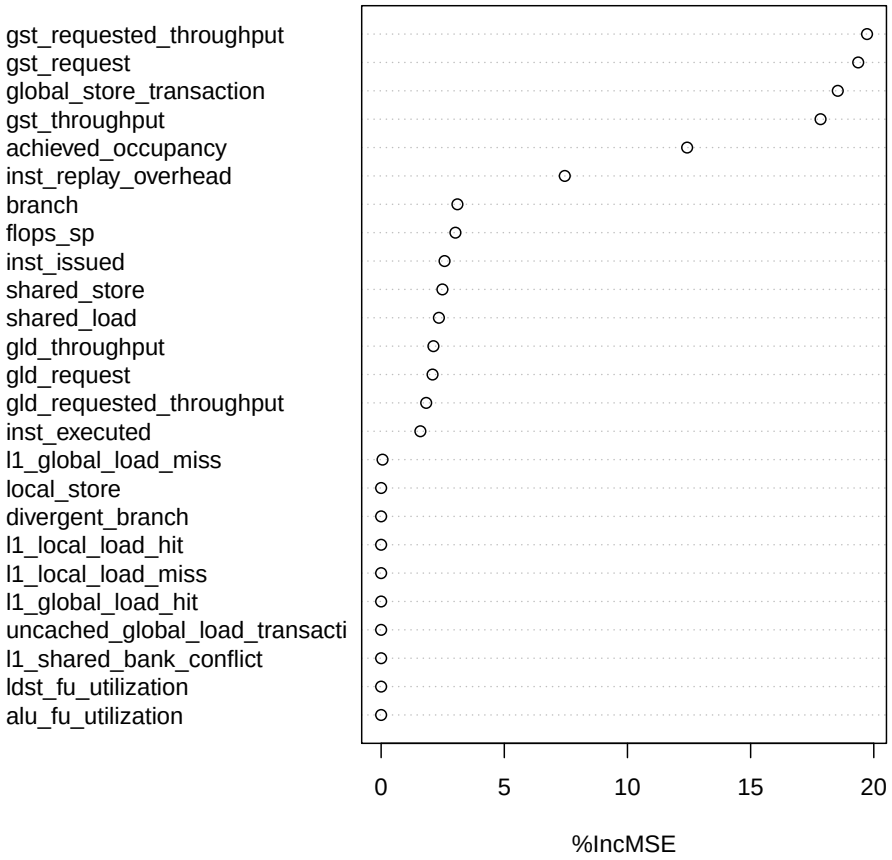


variable importance

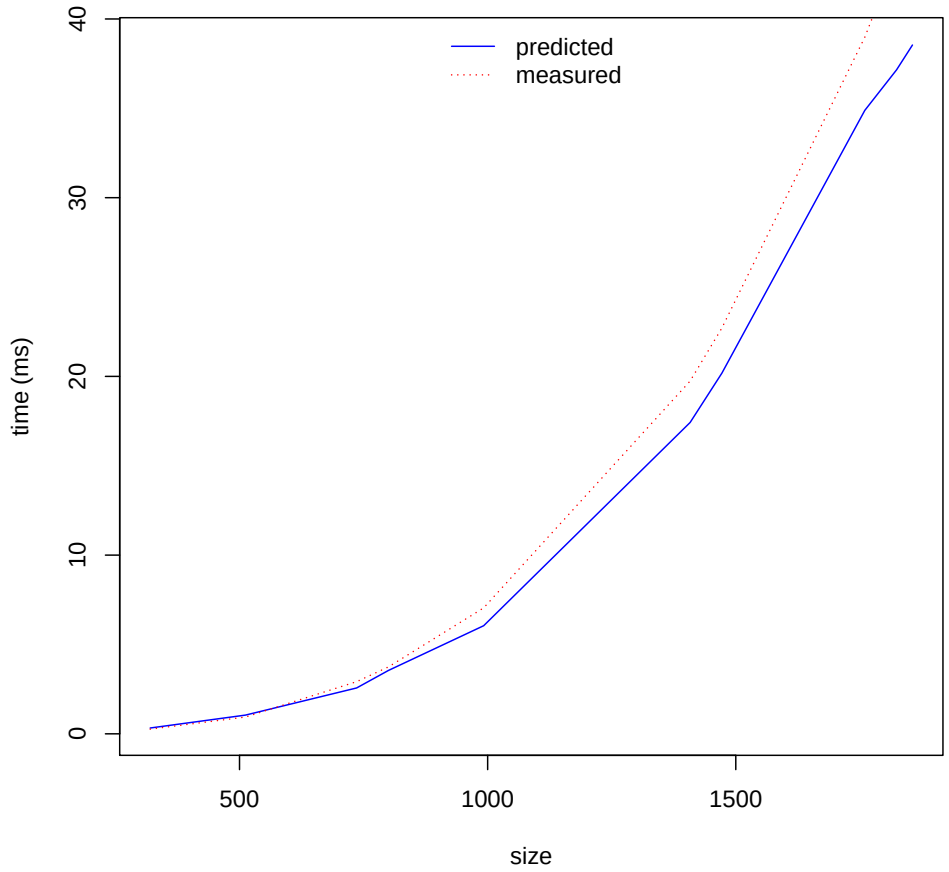


Prediction of unseen grid sizes

MM Problem Scaling on GPU

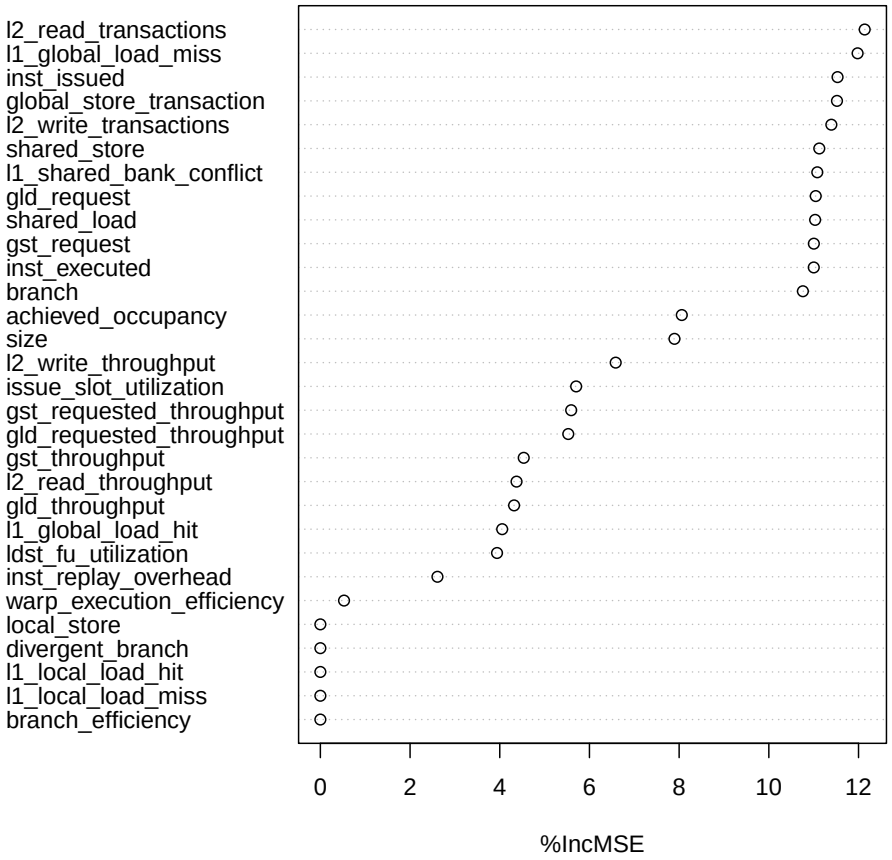


variable importance

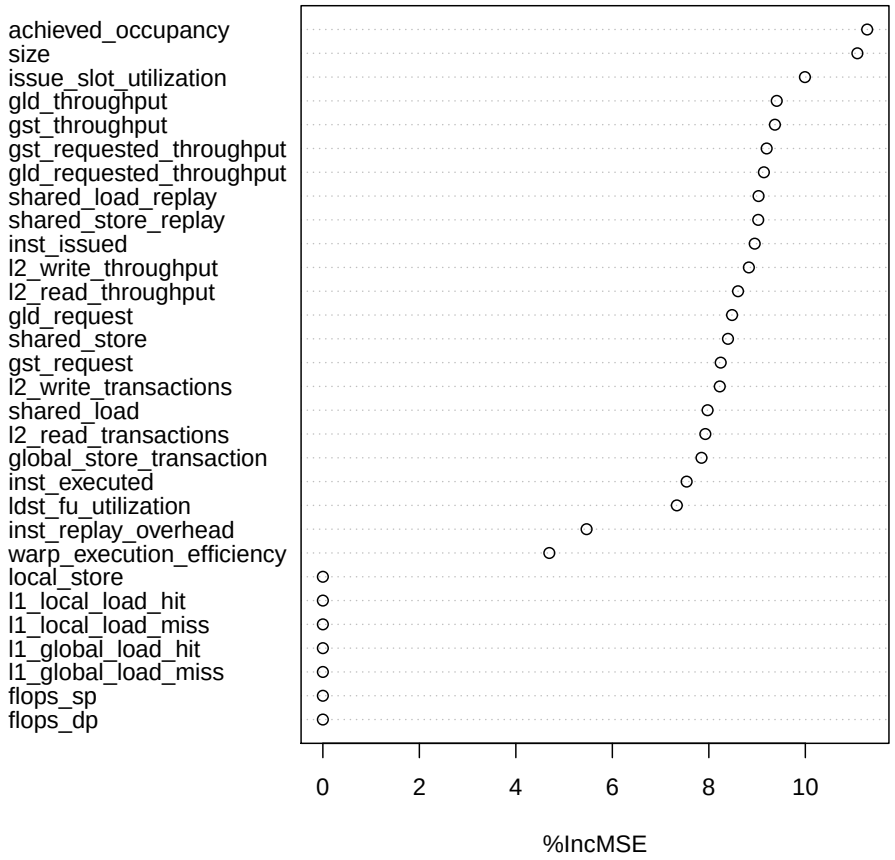


predicting unseen matrix sizes

NW Hardware Scaling on GPUs (1/2)

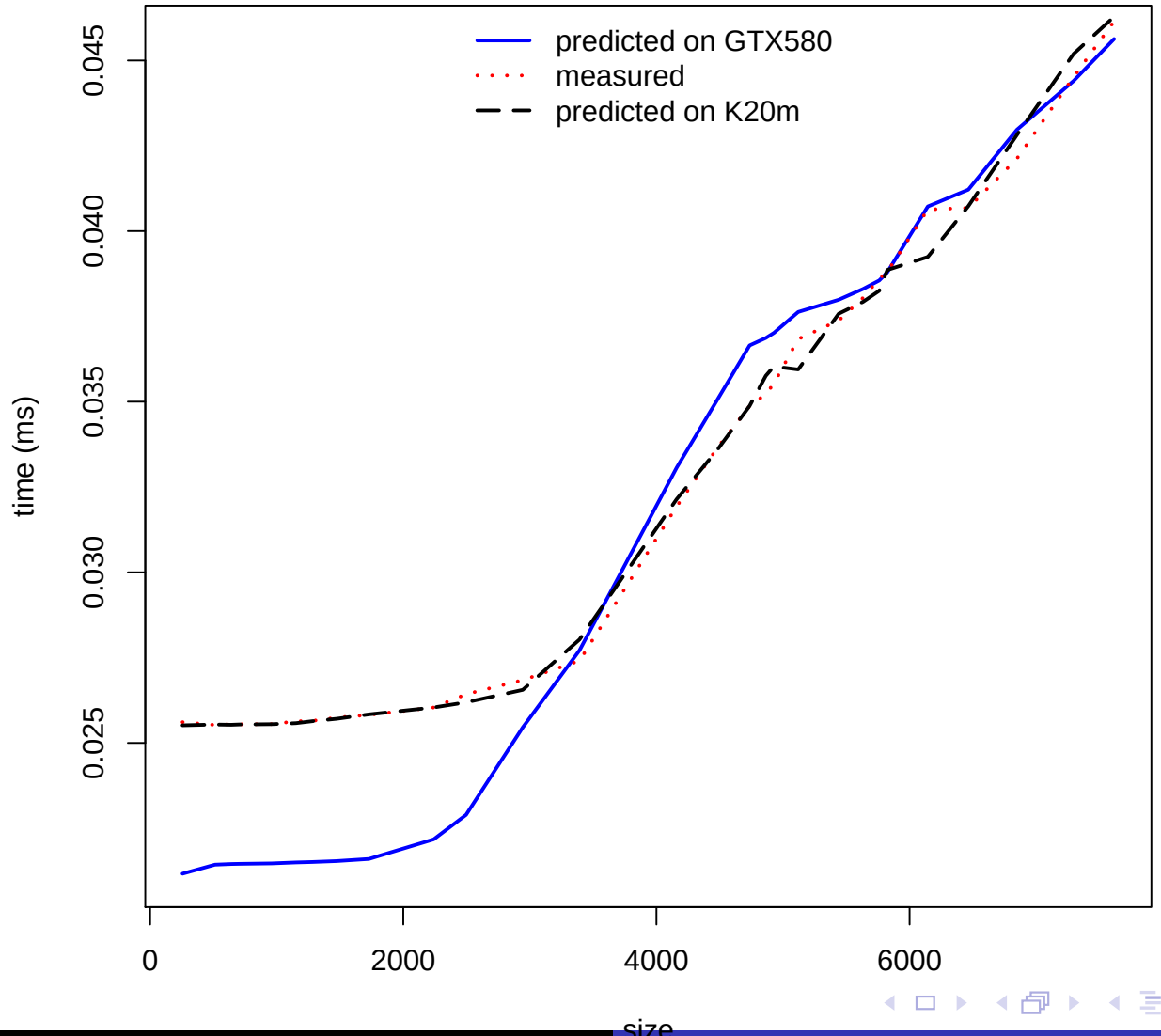


variable importance on GTX480



variable importance on K20m

NW Hardware Scaling on GPUs (2/2)



Conclusion & Outlook

Results:

- BF is a step towards an easy-to-use and insightful PM framework
- Accuracy, quasi automation, application and architecture agnostic

Future directions:

- Automation
- Improve accuracy for irregular applications
- Build higher level metrics on top of PC
- Address counter hardware specificity to improve portability (PAPI?)
- Implement correlation between counter behavior vs performance issue

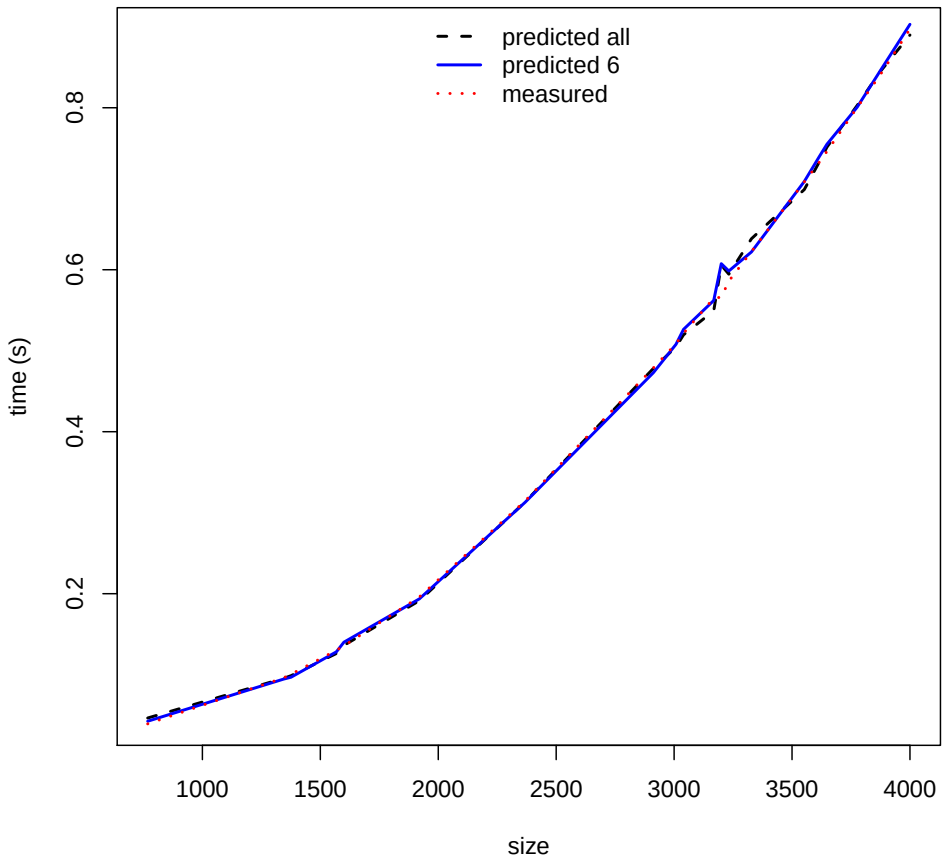
Questions?

Questions?

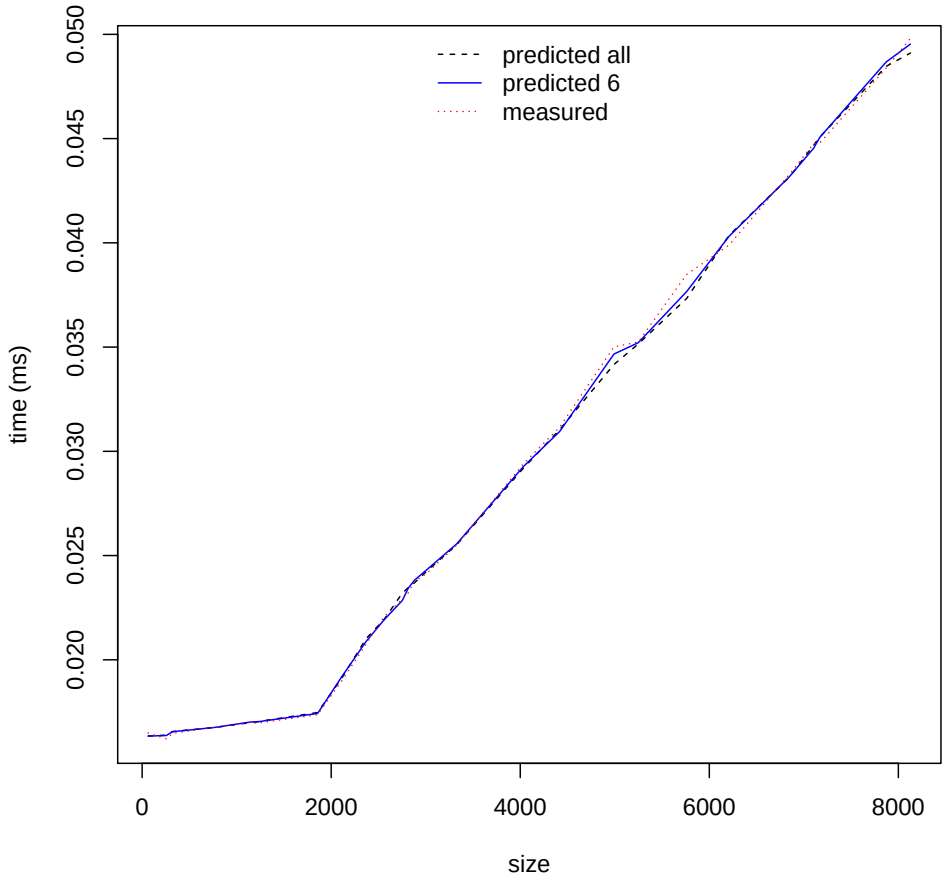
Source: `https://bitbucket.org/smadougou/rfpm`
(**Warning:** Pre-alpha software)

Email:{s.madougou,a.l.varbanescu}@uva.nl

Simplification validation using VI

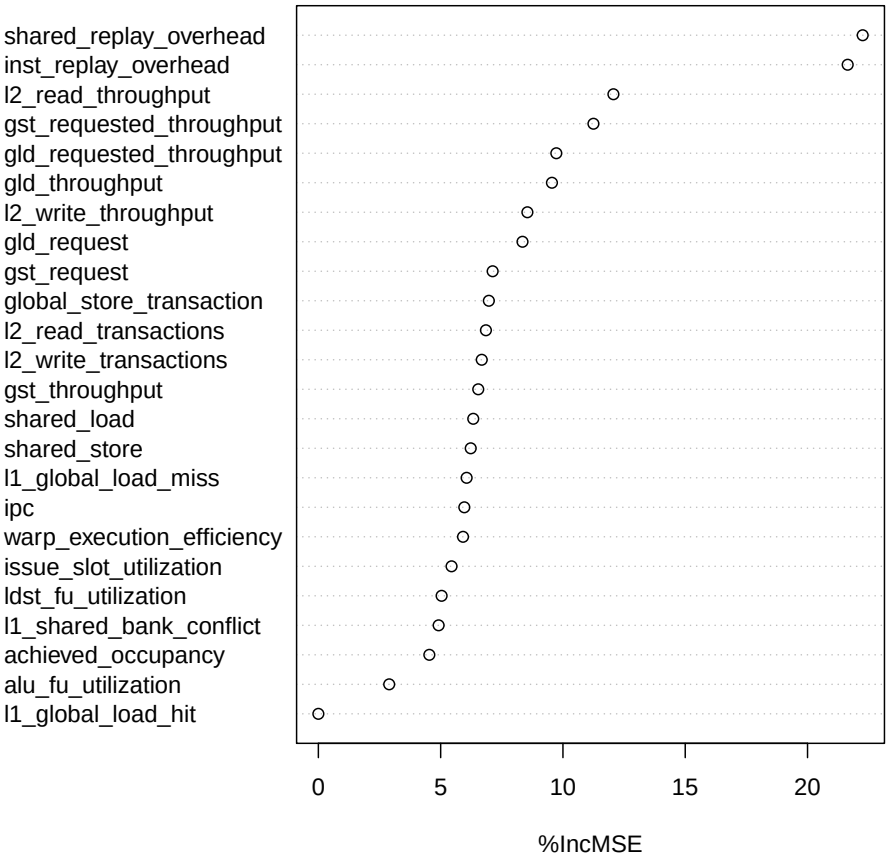


HS on CPU

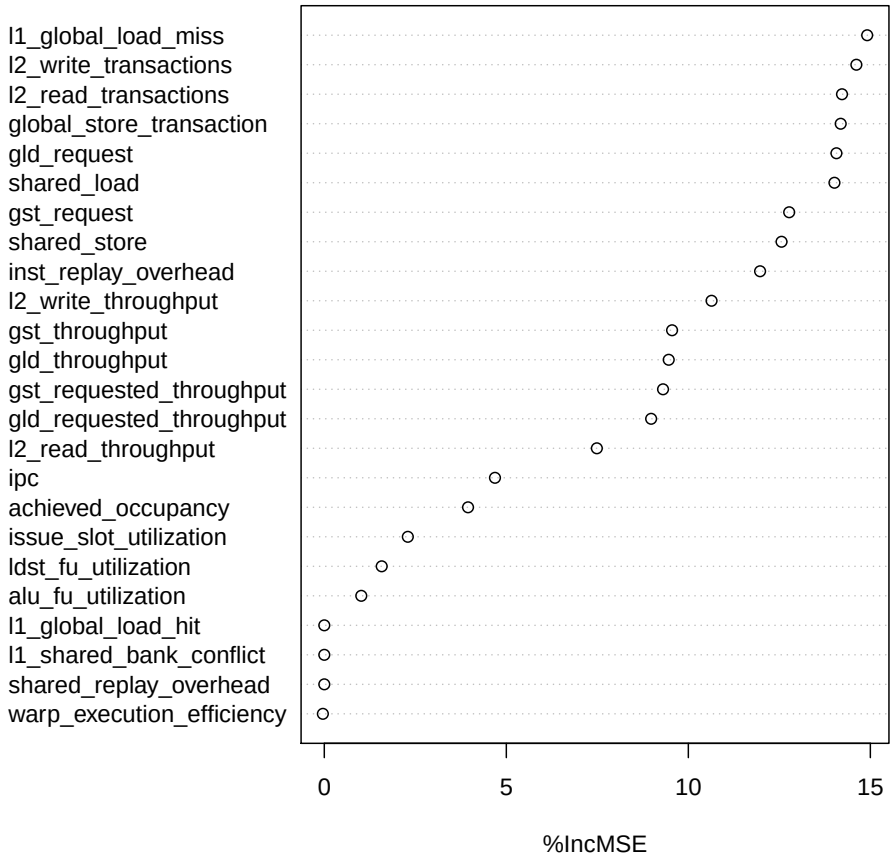


NW on GPU

Bottleneck analysis using VI

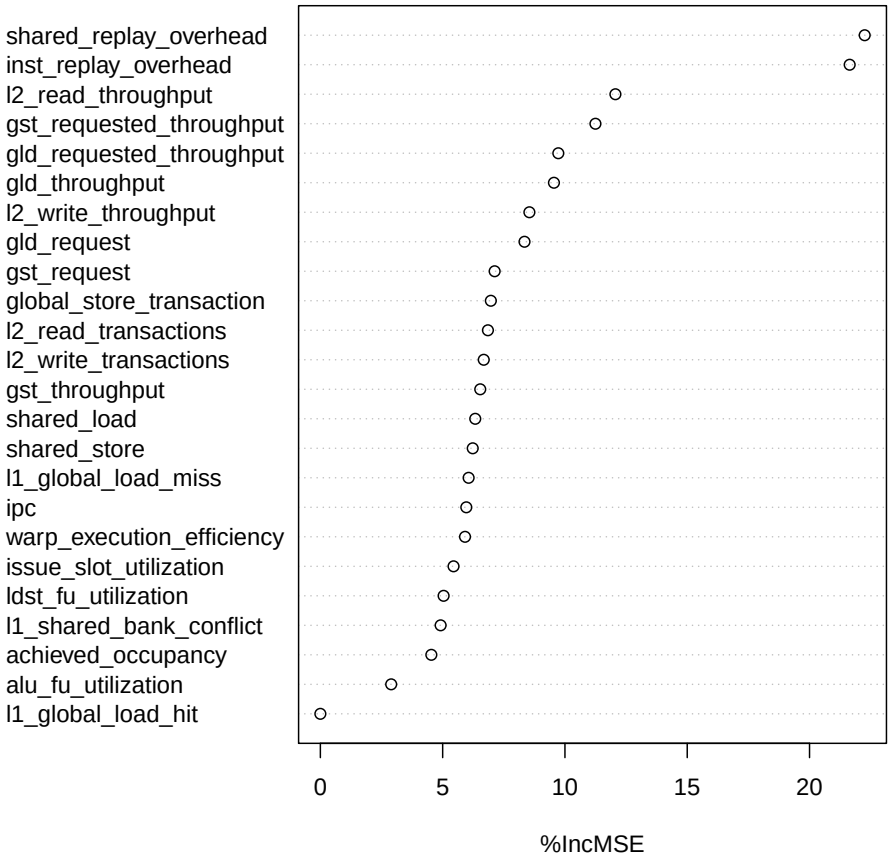


reduce1 VI on GPU

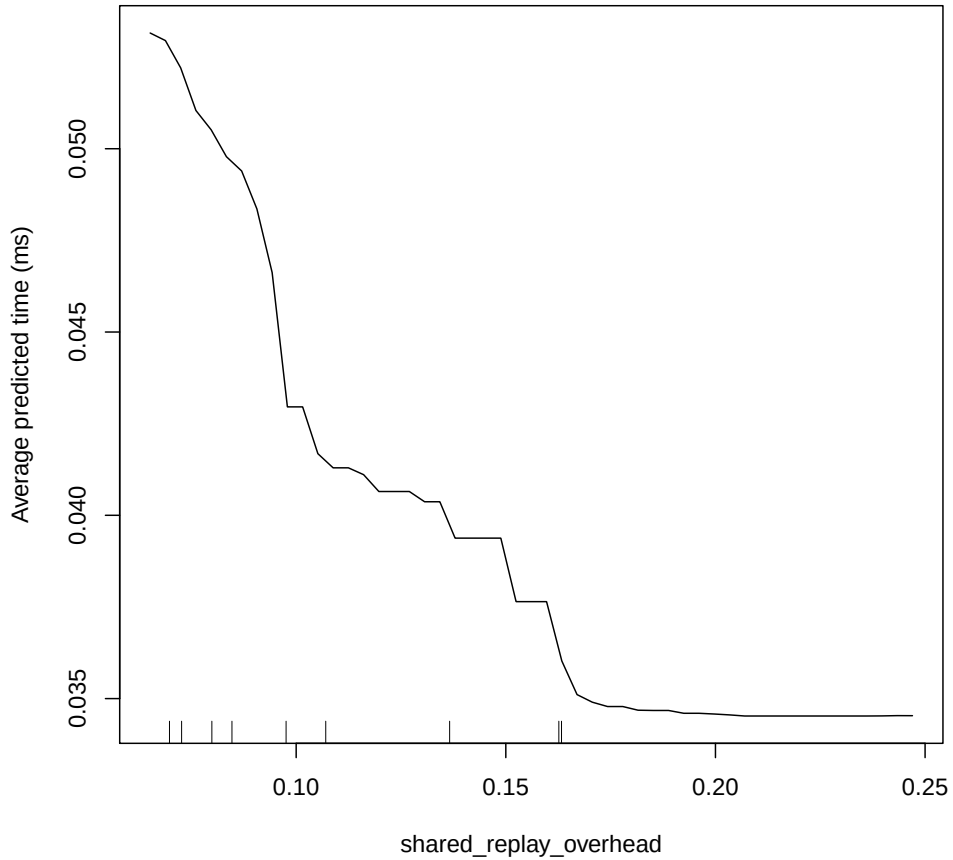


reduce2 VI on GPU

Predictor-Response Association (1/2)

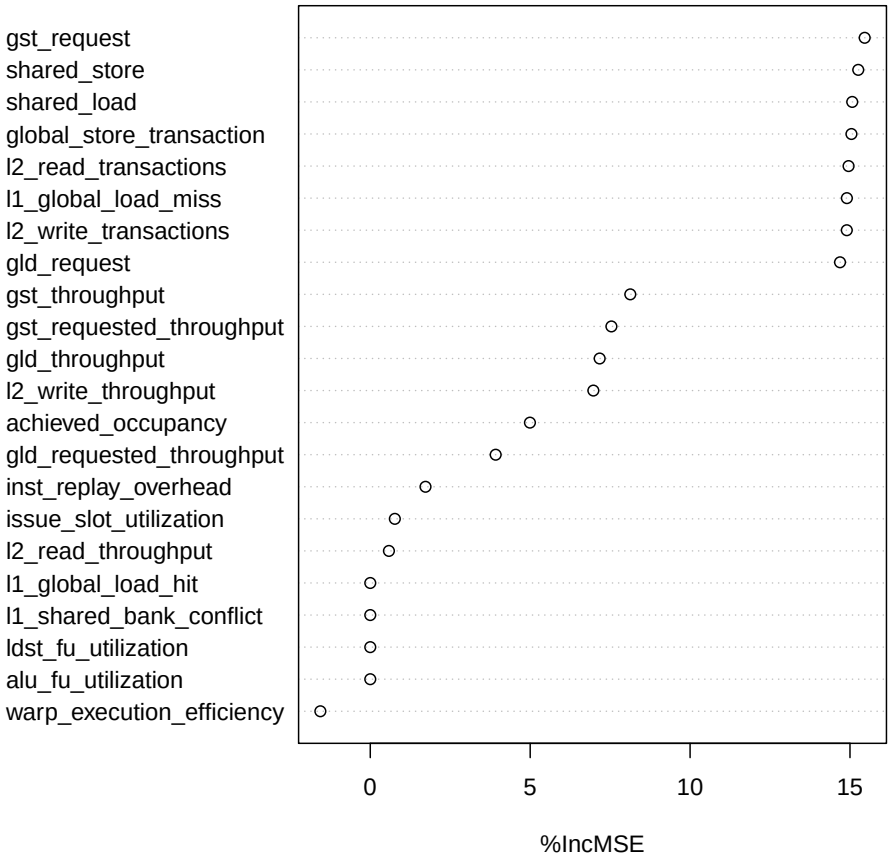


reduce1 VI on GPU

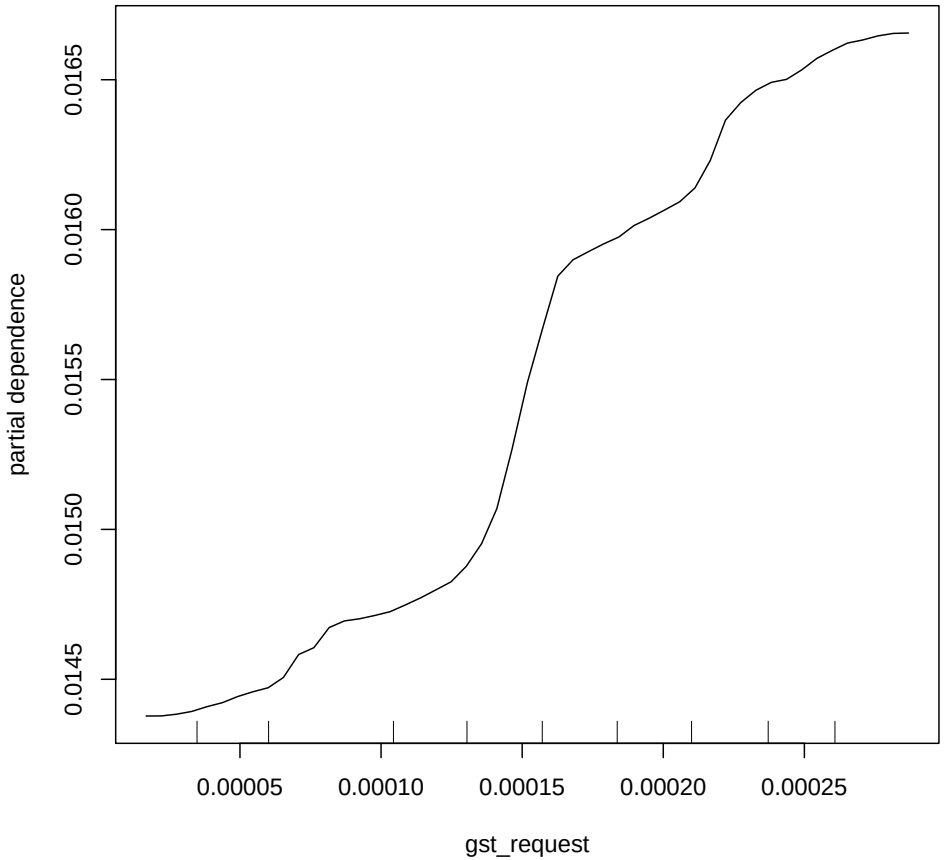


Partial Dependence Plot

Predictor-Response Association (2/2)

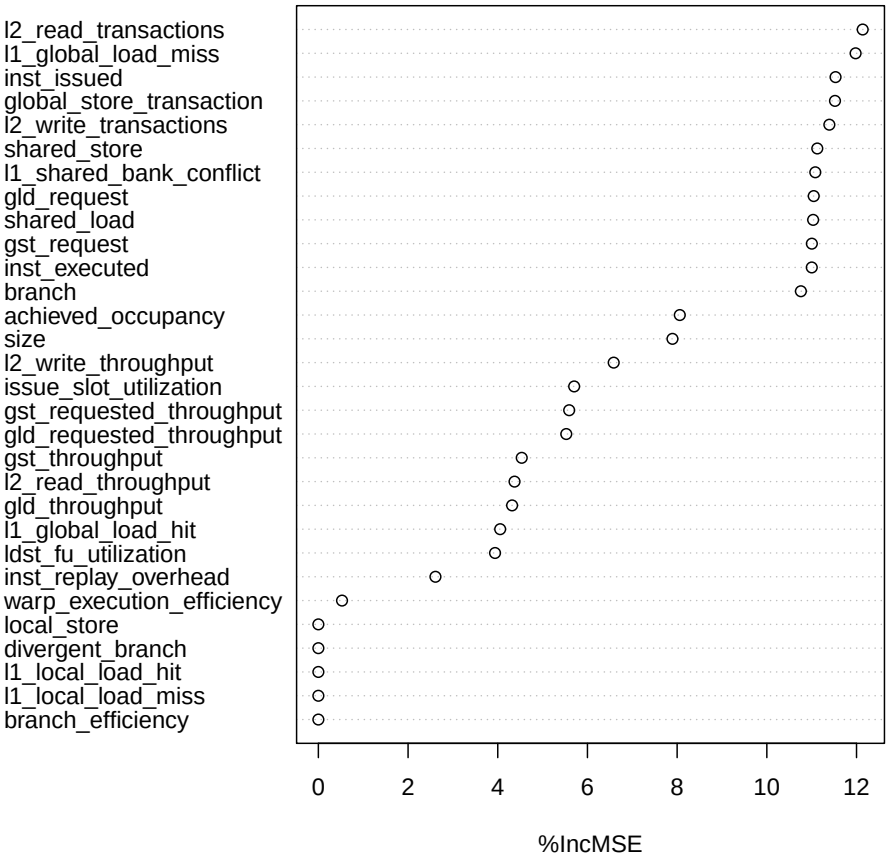


reduce6 VI on GPU

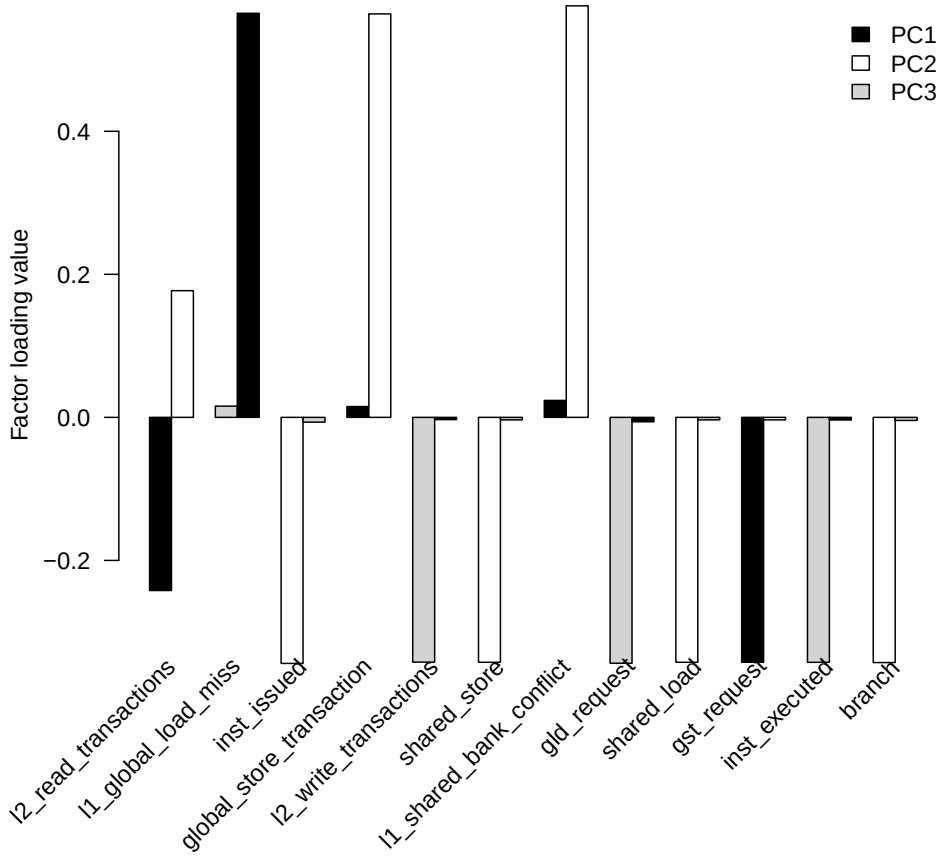


Partial Dependence Plot

Redundant predictors removal using PCA



variable importance on GTX480



PCA involving 9 most VI

MM hardware scaling on GPUs