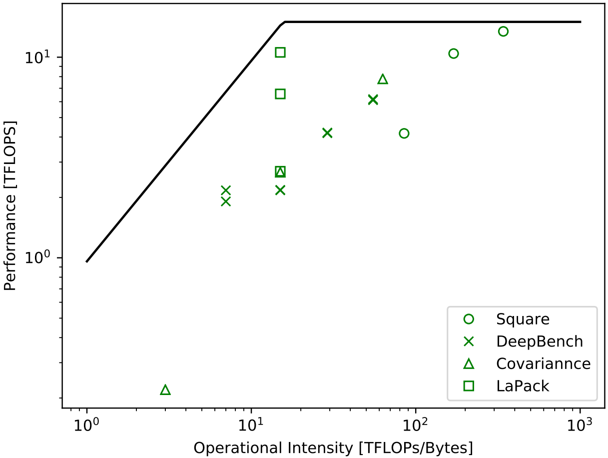
Input-Aware Auto-Tuning of Compute-Bound HPC Kernels

Philippe Tillet, David Cox
Havard University

Motivations

Figure: cuBLAS v9.0 (sGEMM) vs Roofline Model – GV100



Preliminaries

Assume the existence of a kernel generator for GEMM/CONV

$\mathbf{x}_k$: kernel parameters (e.g., tile sizes)

$\mathbf{x}_i$: input parameters (e.g., array shapes, data-type)

$y(\mathbf{x}_i, \mathbf{x}_k)$: Performance of a given kernel on given inputs

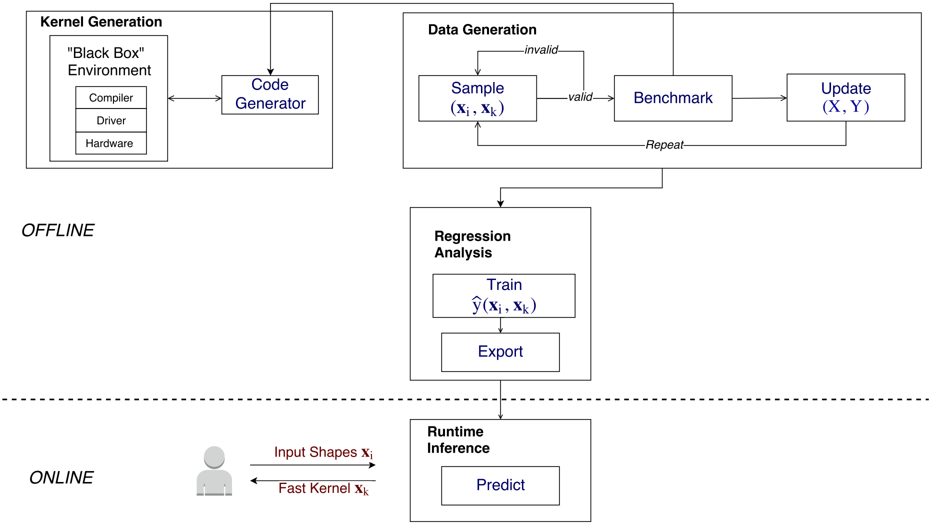
(Input-Sensitive) Auto-Tuning:

- Offline: Choose $\mathbf{x}_i$; find $\arg\max_{\mathbf{x}_k} y(\mathbf{x}_i, \mathbf{x}_k)$.

Input-Aware Auto-Tuning:

- Offline: Build a predictive model $\hat{y}$ for y.
- Online: $\mathbf{x}_i$ is imposed; find $\arg\max_{\mathbf{x}_k} \hat{y}(\mathbf{x}_i, \mathbf{x}_k)$.

Method


Kernel Generation
"Black Box" Environment
Compiler
Driver
Hardware
Code Generator
Data Generation
invalid
Sample $(\mathbf{x}_i, \mathbf{x}_k)$
valid
Benchmark
Update (X, Y)
Repeat
OFFLINE
Regression Analysis
Train $\hat{y}(\mathbf{x}_i, \mathbf{x}_k)$
Export
ONLINE
Input Shapes $\mathbf{x}_i$
Fast Kernel $\mathbf{x}_k$
Runtime Inference
Predict

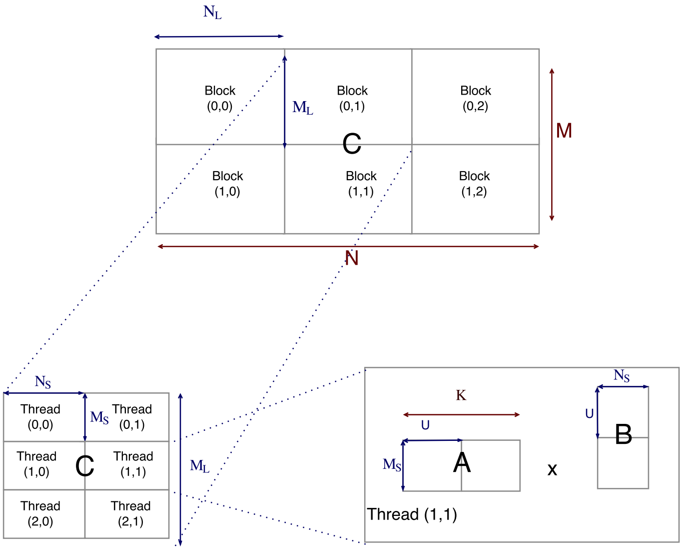
Kernel Generation

Kernel Generation

Goal: Transform a set of parameters $(\mathbf{x}_k, \mathbf{x}_i)$ (e.g., tile sizes, matrix shapes) into functional binaries.

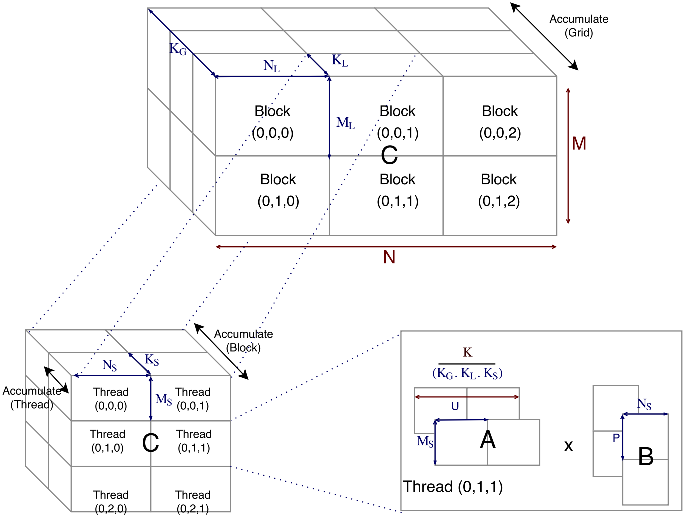
Kernel Generation

$\mathbf{x}_k = (M_L, N_L, M_S, N_S, U)$ $\qquad$ $\mathbf{x}_i = (M, N, K, opA, opB)$



Kernel Generation

$\mathbf{x}_k = (M_L, N_L, M_S, N_S, U, \mathbf{K_G}, \mathbf{K_L}, \mathbf{K_S})$ $\mathbf{x}_i = (M, N, K, opA, opB)$



Kernel Generation

Additional optimizations are necessary for optimal-performance:

- Double-buffering
- Vector loads/stores when possible
- Instructions predications (PTX)
- Explicit rematerialization

CONV is essentially GEMM with **fancy** addressing

Data Generation

Data Generation

Goal: Generate a set of pairs $(\mathbf{x}_n, y_n)$ where $\mathbf{x} = (\mathbf{x}_i, \mathbf{x}_k)$

Method: Sample $\mathbf{x}$ and measure y.

Data Generation

Problem: The space of valid configurations may be **very** (99.9%) sparse

Solution: Two potential solutions:

(a) Rejection sampling: ignore invalid samples.

(b) Generative modeling: build a model for valid kernel configurations

The parameters can for instance be seen as independent categorical variables:

$$P(\mathbf{x} \in \mathbb{X}) = p(\mathbf{x}_0)...p(\mathbf{x}_{I+K})$$

Data Generation

Problem: The auto-tuning procedure is bound by kernels compilation.

Solutions:

(a) Use a low-level language (PTX)

(b) Compile multiple kernels in parallel

Can compile 80,000 kernels per hour!

Regression Analysis

Regression Analysis

Goal: Given X, Y build a predictive model $\hat{y}(\mathbf{x})$

Let's use Deep Learning because ~~HYPE~~:

- Scale well (and we can collect a lot of data!)
- Fast evaluation at runtime

Regression Analysis

Problem: Vanilla Neural Networks are **bad** at handling divisions and multiplications between features:

Even the simplest analytical model will fail:

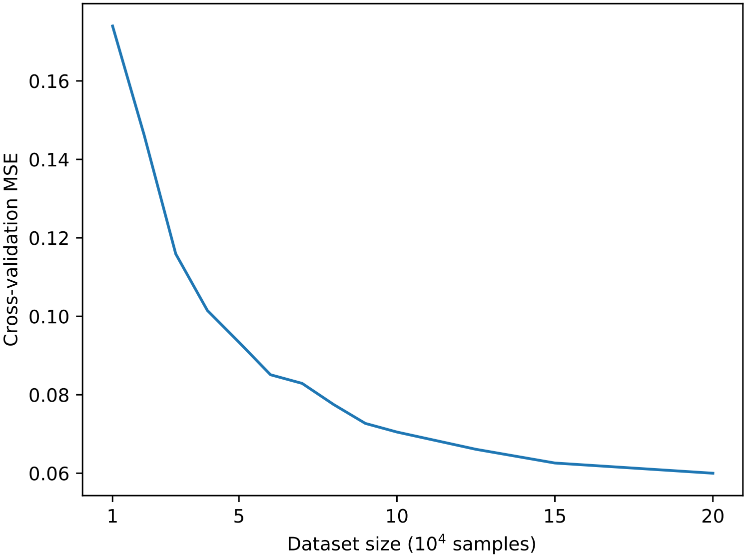
$$t_{\text{arith}} = \max\left(\frac{\text{alu\_latency}}{occupancy}, \text{alu\_throughput}\right)$$

$$t_{\text{mem}} = \max\left(\frac{\text{mem\_latency}}{occupancy}, \text{mem\_throughput}\right)$$

$$t = \max(t_{\text{arith}} i_{\text{arith}}, \quad t_{\text{mem}} i_{\text{mem}})$$

Solution: Learn $\hat{y}(\mathbf{x}' = \log \mathbf{x})$ instead. Multiplications become linear combinations and voila!

Performance vs Dataset Size



Performance vs Network Capacity

Hidden layer sizes	#weights	MSE (no log)
64	1k	0.17 (1.2)
512	10k	0.13 (1.0)
32, 64, 32	5k	0.088 (0.80)
64, 128, 64	17k	0.08 (0.75)
32, 64, 128, 64, 32	21k	0.073 (-)
64, 128, 256, 128, 64	83k	0.067 (-)
64, 128, 192, 256, 192, 128, 64	163k	0.062 (-)

Table: Cross-validation MSE for various MLP architectures

Runtime Inference

Runtime Inference

Goal: Given $\mathbf{x}_i$, find the best possible $\mathbf{x}_k$.

Method: Compute $\arg\max_{\mathbf{x}_k} \hat{y}(\mathbf{x}_i, \mathbf{x}_k)$.

- Exhaustive search: millions of candidates $\mathbf{x}_k$ can be evaluated in one second. Global maximum guaranteed
- Other choices: GA, Simulated Annealing...
- Re-benchmark the ~ 10 best predictions and pick the actual fastest.

Method Summary

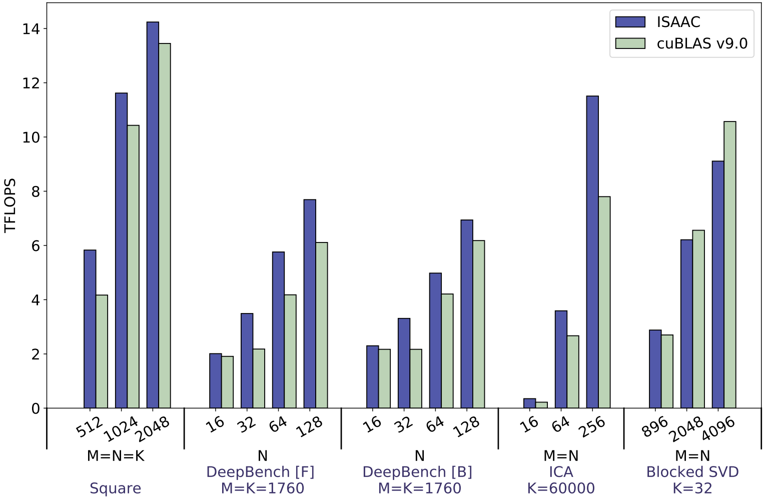
- Build a parameterized code generator for GEMM and CONV
- Benchmark random kernels on random input configurations
- Build a predictive model for the performance of any kernel on any shape
- When shapes are fixed, maximize the model over kernels.

Benchmarks

- Benchmarked the method against the latest cuBLAS and cuDNN...
- ... For GTX980 and GV100 ...
- ... On various HPC tasks (PCA/ICA, SVD)...
- ... And various DL tasks from DeepBench.

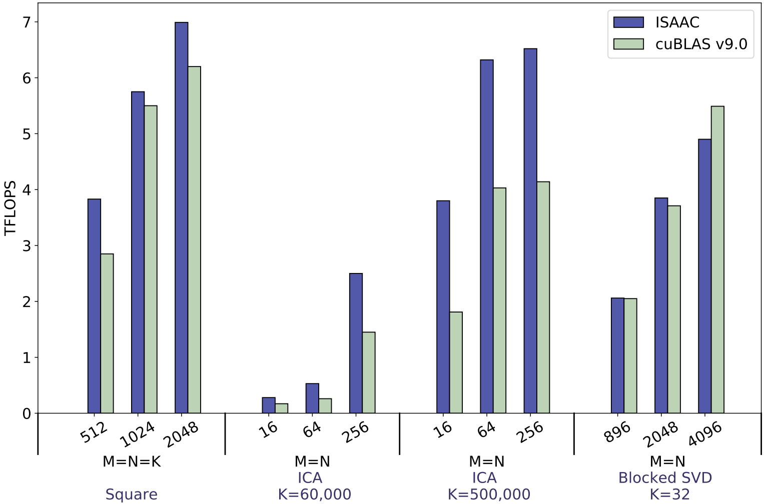
Benchmarks

Figure: SGEMM on GV100

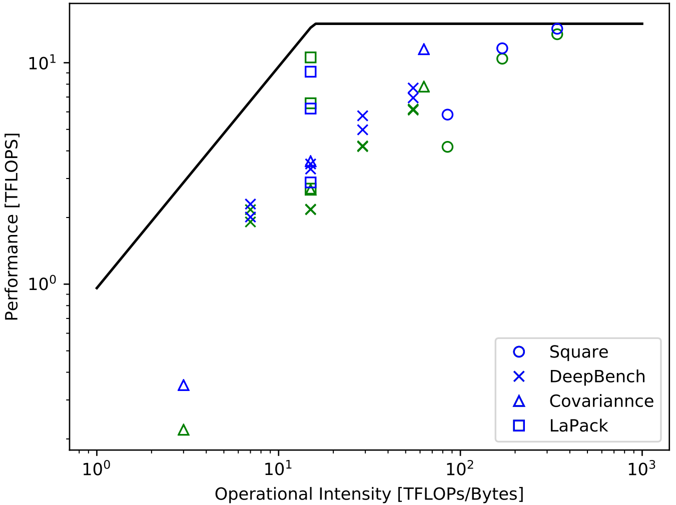


Benchmarks

Figure: DGEMM on GV100

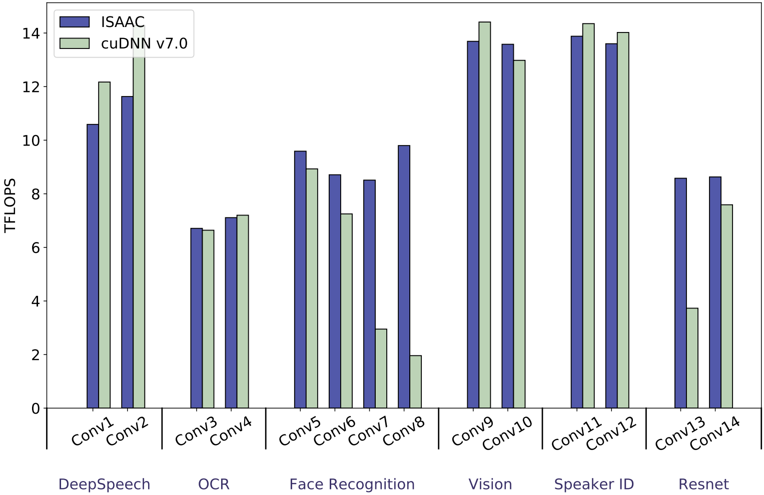


Roofline Model – Revisited



Benchmarks

Figure: SCONV on GV100



Benchmarks

N	P	Q	C	R	S	NPQ	K	CRS	Name
DeepSpeech									
16	79	341	1	5	20	431024	32	100	Conv1
16	38	166	32	5	10	100928	32	1600	Conv2
OCR									
16	24	240	16	3	3	92160	32	144	Conv3
16	12	120	32	3	3	23040	64	288	Conv4
Face Recognition									
8	54	54	64	3	3	23328	64	576	Conv5
8	27	27	128	3	3	5832	128	1152	Conv6
16	14	14	512	5	5	3136	48	12800	Conv7
16	7	7	832	5	5	784	128	20800	Conv8
Vision									
8	112	112	64	3	3	100352	128	576	Conv9
8	56	56	128	3	3	25088	256	1152	Conv10
Speaker ID									
16	39	174	64	5	5	79872	128	1600	Conv11
16	19	87	128	5	5	77824	256	3200	Conv12
ResNET									
16	7	7	512	3	3	784	512	4608	Conv13
16	7	7	1024	1	1	784	2048	1024	Conv14

Analysis

ISAAC learns to make sensible parameterization choices:

(a) Smaller problems require smaller tile sizes

(b) Deep reductions should be mindfully split

(c) Blocking is good for arithmetically intense problems, but adds overhead in IO-bound tasks (SVD)

Conclusions

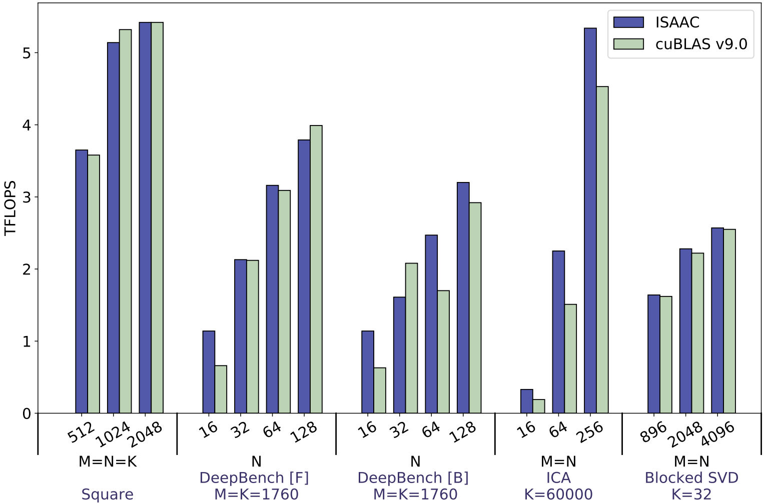
- Presented an input-aware auto-tuning technique for HPC and DL
- Presented design techniques for:
    - High-performance GEMM templates with *reduction-splitting*
    - Efficient parameters sampling with *generative models*
    - Accurate performance modeling with *deep learning*
- Performance often superior to hand-tuned assembly

Thanks for your attention!

`http://github.com/ptillet/isaac/`
The binding of Isaac: BLAS, Tensorflow.

Benchmarks

Figure: SGEMM on GM200



Benchmarks

Figure: SCONV on GM200