Portable, Customizable, Black-Box GPU Performance Modeling

James Stevens

October 12, 2018

Acknowledgements

- Andreas Klöckner
- Matt Wala
- Kaushik Kulkarni

Big Picture

- Everyone wants fast and easy solutions to PDEs

PDE → Solver → Code

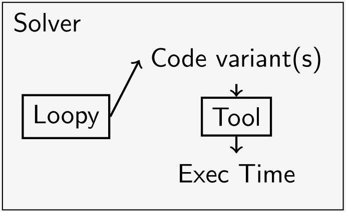
- fast = high performance code
- easy = minimal input from user

High Performance Code

- Different code variants perform better on different machines
- Solver must produce, select these with minimal user effort

Code Variants

- Loopy provides code transformation



- Need tool to choose high performing transformation set from available options

What should this tool look like?

- Analytical model determining which variant to produce

$$T_{\text{wall}}(\mathbf{n}) \approx g\left(\text{feat}_0^{\text{in}}(\mathbf{n}), \ldots, \text{feat}_j^{\text{in}}(\mathbf{n}), p_0, \ldots, p_k\right)$$

- e.g., $t = \text{madds}(\mathbf{n}) \cdot p_{\text{madd}}$
- *Feature*: quantitative kernel characteristic
- *Parameter*: hardware-dependent value relating features to exec time
- How do we determine g? $\leftarrow$ Key question!

What should this tool look like?

- Analytical model determining which variant to produce

$$T_{\text{wall}}(\mathbf{n}) \approx g\left(\text{feat}_0^{\text{in}}(\mathbf{n}), \ldots, \text{feat}_j^{\text{in}}(\mathbf{n}), p_0, \ldots, p_k\right)$$

- e.g., $t = \text{madds}(\mathbf{n}) \cdot p_{\text{madd}}$
- *Feature*: quantitative kernel characteristic
- *Parameter*: hardware-dependent value relating features to exec time
- How do we determine g? $\leftarrow$ Key question!
  (topic of this presentation)

How to Determine Model Expression

- Determine kernel features a priori
- Require minimal hardware info from user; GPU = black box
- Find parameters $p_0, \ldots, p_k$ by gathering feature values (including exec times) from set of *measurement computations* and fitting model to data
- Model expression g
  - Can we create broadly applicable g?

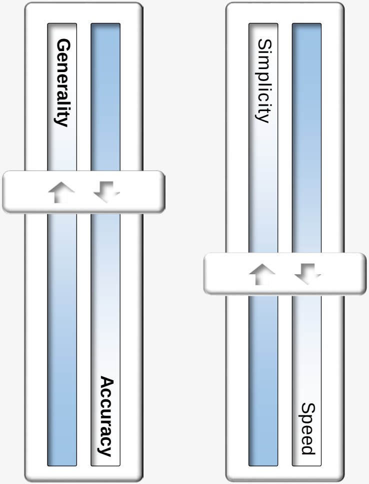
Memory access pattern variants

```
for (int k_outer = 0; k_outer <= int_floor_div_pos_b(-16 + n, 16); ++k_outer)
  ...
  a_fetch[...] = a[n * (16 * gid(1) + lid(1)) + 16 * k_outer + lid(0)];
  b_fetch[...] = b[n * (16 * k_outer + lid(1)) + 16 * gid(0) + lid(0)];
  ...
```

- Fetching **b** takes 5x longer
- *Access patterns* for memory access features have numerous characteristics that individually may affect execution time
  - Multiple thead index strides (any int)
  - loop stride (any int)
  - access to footprint ratio (any float)
  - data size
  - direction
  - memory type
- Broadly applicable model expression would be massive, most features unused for given computation

Approach

- Let developer build model that meets their needs
  - Custom model creation
  - Custom measurement kernel set generation



Simple Demo - Model Mat-mul on GTX Titan X GPU

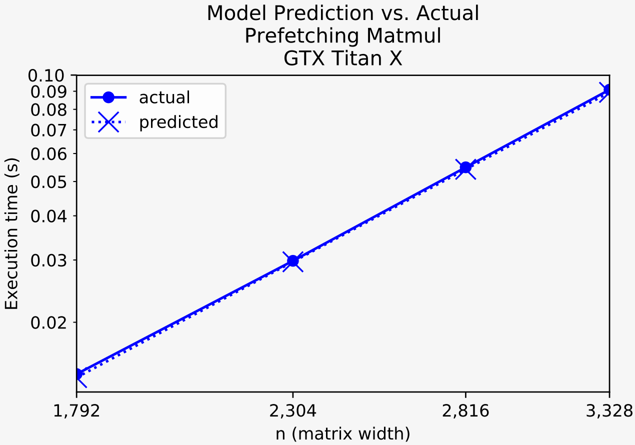
Predict execution time for square tiled matrix multiplication

- Very simple model: $t = \text{madds}(\mathbf{n}) \cdot p_{\text{madd}}$
- Measurement computations: more matmuls

Simple Demo - Model Matmul on GTX Titan X GPU

Quick demo

Simple Demo - Model Mat-mul on GTX Titan X GPU



Software components

- **Loopy.statistics:** Kernel stats counting
- **Perflex:** Model/feature construction
- **UIPiCK:** Measurement kernel set generation

Counting Statistics with Loopy

Kernel stats collected

- Memory traffic
  - Track mem access strides, data size, memory type, direction, access-to-footprint ratio
- (FL)OPs: $+$, $\times$, $\div$, a^b, **multiply-add**
  - Track data type
- Synchronization
  - Launch, local barrier

Counting Statistics with Loopy

```
knl = lp.make_kernel(
      "{[i,j]: 0<=i,j<n}",
      "a[i,j]=b[j,i]")
```

How do we count?

1. Recursively traverse **instruction expression tree** of a Loopy kernel, counting stats for single instruction
2. Determine how many times instruction executes
   - Barvinok counting library
     S. Verdoolaege et. al. Counting Integer Points in Parametric Polytopes Using Barvinok's Rational Functions, Algorithmica, v.48 n.1, March 2007

Creating Model and Features with Perflex

```
m = Model(
    "f_cl_wall_time_nvidia_geforce",
    "p_madd * f_op_float32_madd + "
    "p_mem * f_mem_access_global_float32_load_lstrides:{0:1;1:16}_ratio:<2")
```

Feature

- Quantitative kernel characteristic that affects execution time
- May create custom features - implement as object with eval(knl) function that returns numeric value
  - Number of **32-bit global** memory **loads** w/ local thread ID **strides** $\{\mathbf{0}, \mathbf{1}\}$ and memory access-to-footprint **ratio** $\leq$ **2**
  - Number of thread groups

Creating Model and Features with Perflex

Model

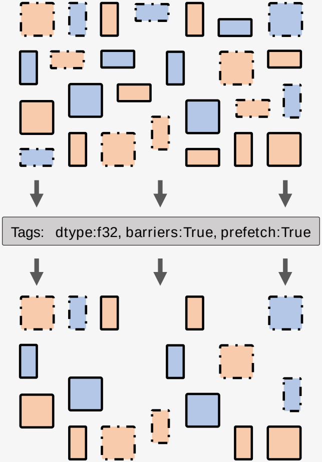
$$T_{\text{wall}}(\mathbf{n}) = \text{feat}^{\text{out}}(\mathbf{n}) \approx g\left(\text{feat}_0^{\text{in}}(\mathbf{n}), \ldots, \text{feat}_j^{\text{in}}(\mathbf{n}), p_0, \ldots, p_k\right)$$

- Mathematical expression relating input features and parameters to output feature, differentiable with respect to parameters
- Gather features for all measurement kernels, then fit model using nonlinear least squares to solve for model parameters

Generating Kernels with UIPiCK

```
tags = [
    "matmul_sq", "groups_fit:True", "dtype:float32",
    "lsize_0:16", "lsize_1:16", "tiled_prefetch:True"]
kc = KernelCollection(uipick.ALL_GENERATORS)
m_knls = kc.generate_kernels(tags)
```

- Use customizable tags to control which kernels will be produced
- Filter out, e.g.,
  - Kernels operating on float64 data
  - Kernels that don't use local mem



Modeling local-global overlap

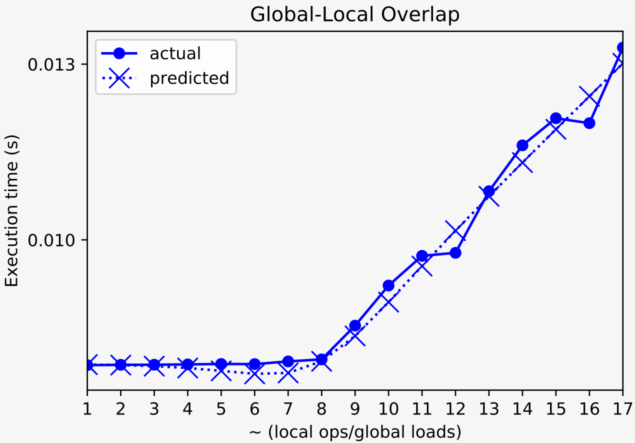
$$t \approx \max(c_{\text{global}}, c_{\text{local}})$$

$$s(x) = \begin{cases} 0 & \text{if } x < 0, \\ 1 & \text{if } x \geq 0, \end{cases}$$

$$\hat{s}(x) = (\tanh(p_{\text{edge}} \cdot x) + 1)/2$$

$$t \approx c_{\text{global}} \cdot \hat{s}(c_{\text{global}} - c_{\text{local}}) + c_{\text{local}} \cdot \hat{s}(c_{\text{local}} - c_{\text{global}})$$

Modeling local-global overlap



Two types of models

Linear model:

$$t \approx c_{\text{overhead}} + c_{\text{global}} + c_{\text{local}}$$

Nonlinear model:

$$t \approx c_{\text{overhead}} + c_{\text{global}} \cdot \hat{s}(c_{\text{global}} - c_{\text{local}}) + c_{\text{local}} \cdot \hat{s}(c_{\text{local}} - c_{\text{global}})$$

$$c_{\text{overhead}} = p_{\text{launch}} \cdot f\text{-launch} + p_{\text{group}} \cdot f\text{-group}$$
$$c_{\text{global}} = p_{\text{gmem-0}} \cdot f\text{-gmem}_0 + \ldots + p_{\text{gmem-}i} \cdot f\text{-gmem}_i$$
$$c_{\text{local}} = \text{(on-chip work: flops, local mem, barriers)}$$

Matmul Model

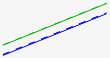
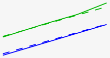
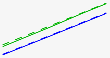
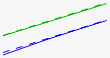
$$c_{\text{overhead}} = p_{\text{launch}} \cdot f\text{-launch} + p_{\text{group}} \cdot f\text{-group}$$

$$\begin{aligned} c_{\text{global}} = {} & p_{\text{r}} \cdot f\text{-gmem}^{\{1,>1\}\{16,\}}_{<f32>[1]} \\ & + p_{\text{bf}} \cdot f\text{-gmem}^{\{1,>1\}\{16,\}}_{<f32>[>8]} \\ & + p_{\text{af}} \cdot f\text{-gmem}^{\{1,>1\}\{0,\}}_{<f32>[>8]} \\ & + p_{\text{b}} \cdot f\text{-gmem}^{\{1,0\}\{,\}}_{<f32>[>8]} \\ & + p_{\text{a}} \cdot f\text{-gmem}^{\{0,>1\}\{,\}}_{<f32>[>8]} \end{aligned}$$

$$c_{\text{local}} = p_{\text{madd}} \cdot f\text{-madd}_{<f32>} + p_{\text{loc}} \cdot f\text{-lmem}_{<f32>}$$

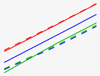
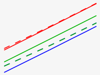
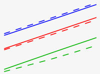
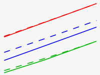
Notation: $f\text{-mem/op type}^{\{\text{local thread id strides}\}\{\text{global thread id strides}\}}_{<\text{data type}>[\text{access-to-footprint-ratio}]}$

Matrix Multiplication Accuracy

GPU	Variant	n	Time range	Error	t vs. n
Tesla K40c	prefetch —	1280…2816	0.013…0.142	**0.055**	
	no fetch —	1280…2816	0.024…0.252	**0.022**	
GTX Titan X	prefetch —	2304…3840	0.031…0.153	**0.042**	
	no fetch —	2304…3840	0.080…0.465	**0.048**	
Tesla C2070	prefetch —	768…2304	0.005…0.134	**0.047**	
	no fetch —	768…2304	0.010…0.289	**0.076**	
Radeon R9 Fury	prefetch —	1280…2816	0.008…0.101	**0.065**	
	no fetch —	1280…2816	0.034…0.344	**0.048**	

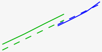
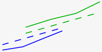
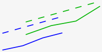
Nonlinear model — Actual - - Predicted

Discontinuous Galerkin Accuracy

GPU	Variant	elements	Time range	Error	t vs. elements
Radeon R9 Fury	fetch diff —	32768…557056	0.009…0.150	**0.460**	
	fetch vec —	32768…557056	0.005…0.091	**0.136**	
	no fetch —	32768…557056	0.017…0.278	**0.034**	
Tesla K40c	fetch diff —	65536…589824	0.005…0.042	**0.218**	
	fetch vec —	65536…589824	0.008…0.069	**0.257**	
	no fetch —	65536…589824	0.014…0.122	**0.027**	
Tesla C2070	fetch diff —	32768…294912	0.096…0.849	**0.127**	
	fetch vec —	32768…294912	0.009…0.082	**0.323**	
	no fetch —	32768…294912	0.038…0.340	**0.127**	
GTX Titan X	fetch diff —	131072…655360	0.011…0.054	**0.403**	
	fetch vec —	131072…655360	0.006…0.027	**0.024**	
	no fetch —	131072…655360	0.034…0.167	**0.003**	

Nonlinear model — Actual - - Predicted

Finite Difference Accuracy

GPU	Variant	n	Time range	Error	t vs. n
Tesla C2070	16x16 tiles — (blue)	10752…12096	0.016…0.021	**0.016**	
	18x18 tiles — (green)	9216…10944	0.013…0.018	**0.063**	
Tesla K40c	16x16 tiles — (blue)	17920…19264	0.021…0.025	**0.045**	
	18x18 tiles — (green)	18432…20160	0.026…0.032	**0.058**	
GTX Titan X	16x16 tiles — (blue)	17920…19264	0.012…0.014	**0.155**	
	18x18 tiles — (green)	18432…20160	0.013…0.017	**0.087**	
Radeon R9 Fury	16x16 tiles — (blue)	8960…11648	0.006…0.009	**0.273**	

Linear model — Actual - - Predicted