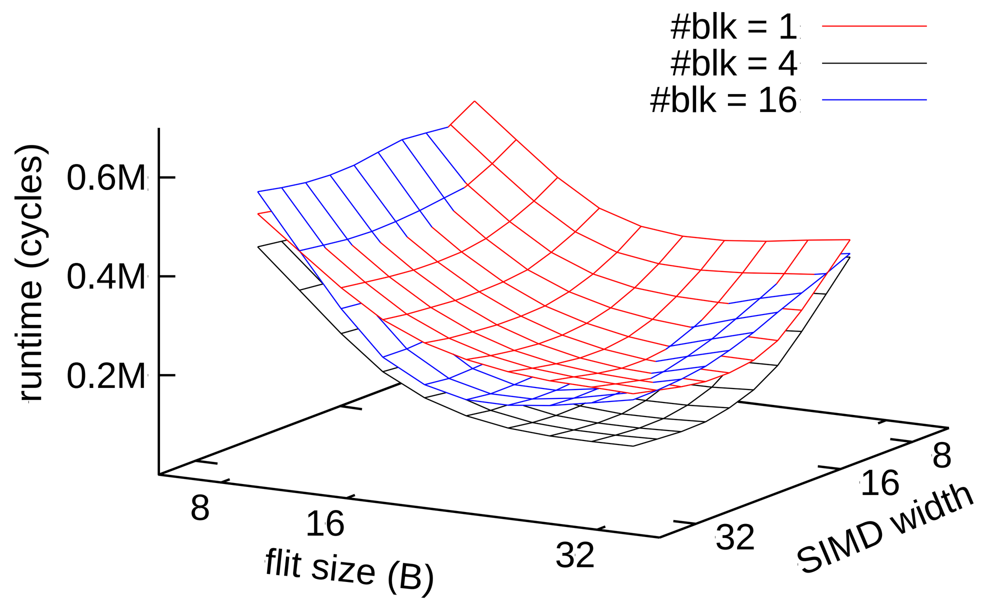
Analyzing and Optimizing GPU Communication and Computation

Wenhao Jia (Princeton Univ), Kelly A. Shaw (Univ of Richmond), Margaret Martonosi (Princeton Univ)

Automated GPU Design Space Exploration

Reference: Stargazer: Automated Regression-Based GPU Design Space Exploration, Wenhao Jia, Kelly A. Shaw, and Margaret Martonosi, *ISPASS 2012*

Observation: Designing GPUs involves discerning relative importance and potential interactions of many design options, which is difficult even just for small design spaces.



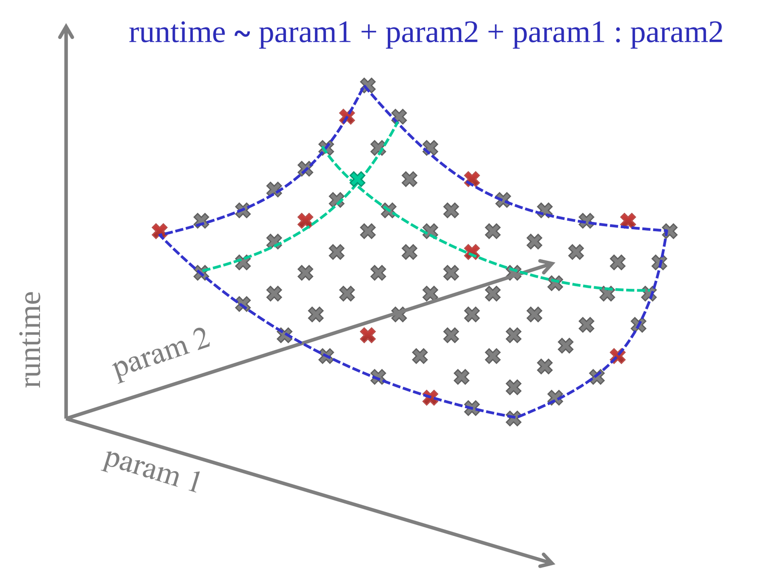
A 3-parameter matrix multiply design space is full of nonlinear parameter interactions.

Our Work: Build GPU performance models from random design samples using automated parameter selection that can account for parameter pair interactions.

Sample
Randomly sample a *small* number of designs from a space with n parameters (P_i, i = 1..n).

Model
A stepwise regression algorithm selects *relevant* parameters and interactions.

Predict
The resulting regression model predicts performance of *any* design in the space.



In a hypothetical 2-parameter design space, the 3 steps are shown color-coded.

Method: A stepwise algorithm automatically selects key design parameters and significant parameter interactions to build performance models with only relevant terms.

```
current model M = {}
unused parameter set T = {P1, P2, …, Pn}
while T is not empty
  for each Pi in T
    generate a tentative model Mi = M + Pi
  select the Mimax with the highest adjusted R^2
  if Mimax's adjusted R² > M's R²
    M = Mimax, T = T - Pi
    for each Pj (j != i) already in M
      if interaction Pi:Pj is significant
        M = M + Pi:Pj
    else
        return M
```

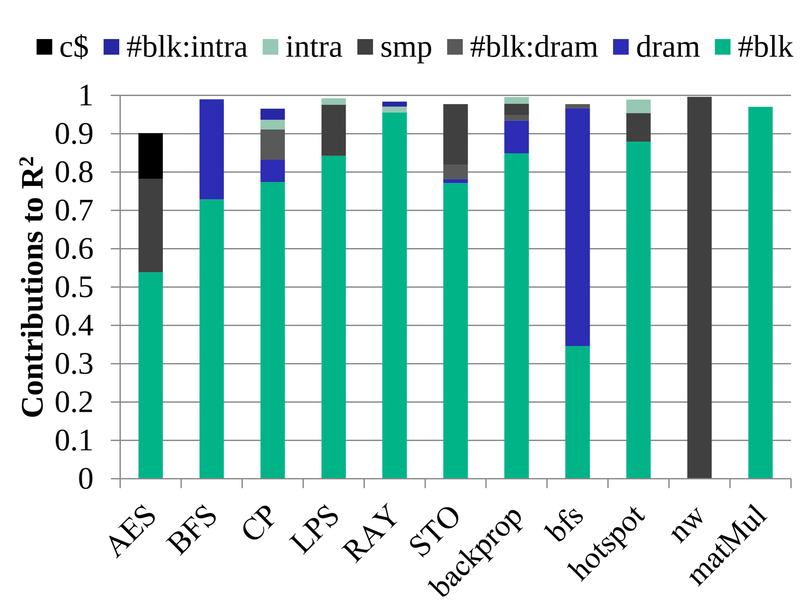
Initialization

If the next most significant factor indeed affects runtime, include it in the model

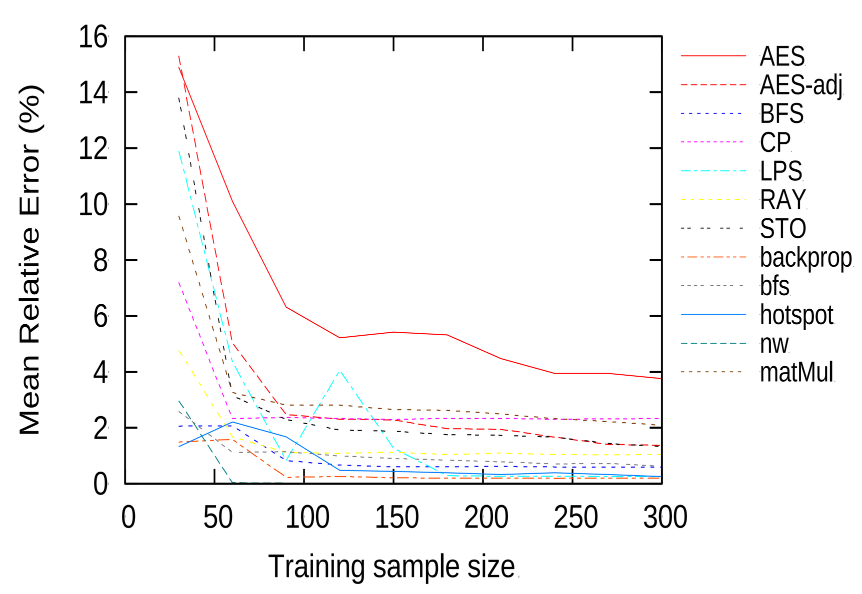
Also test its interactions with included factors

Else exit the routine

Results: Our method automatically and efficiently reveals the relative importance of different design options for a 1 million-point design space in GPGPU-Sim.



Rodinia benchmarks rely on diverse architectural features.



Only 300 samples out of 1 million possible designs yield good model accuracy.

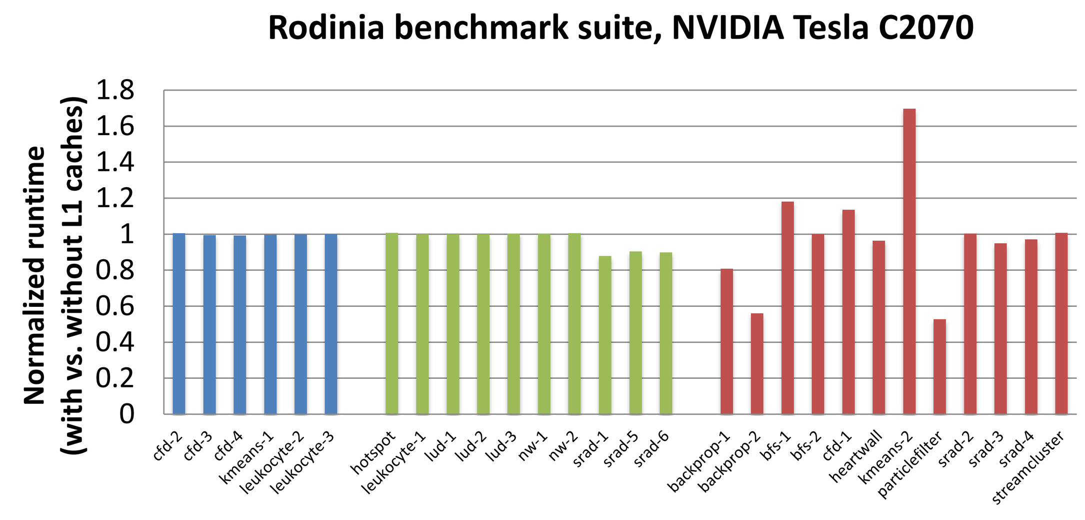
Conclusions:

- Regression methods can automatically and accurately model complex trade-offs in GPU design spaces.
- 4 orders of magnitude reduction in design space evaluation time with less than 1.1% average error.

Optimizing GPU Cache Utility

Reference: Charactering and Improving the Use of Demand-Fetched Caches in GPUs, Wenhao Jia, Kelly A. Shaw, and Margaret Martonosi, *Intl. Conf. Supercomputing 2012*

Observation: GPU caches have unpredictable and even detrimental performance impact.

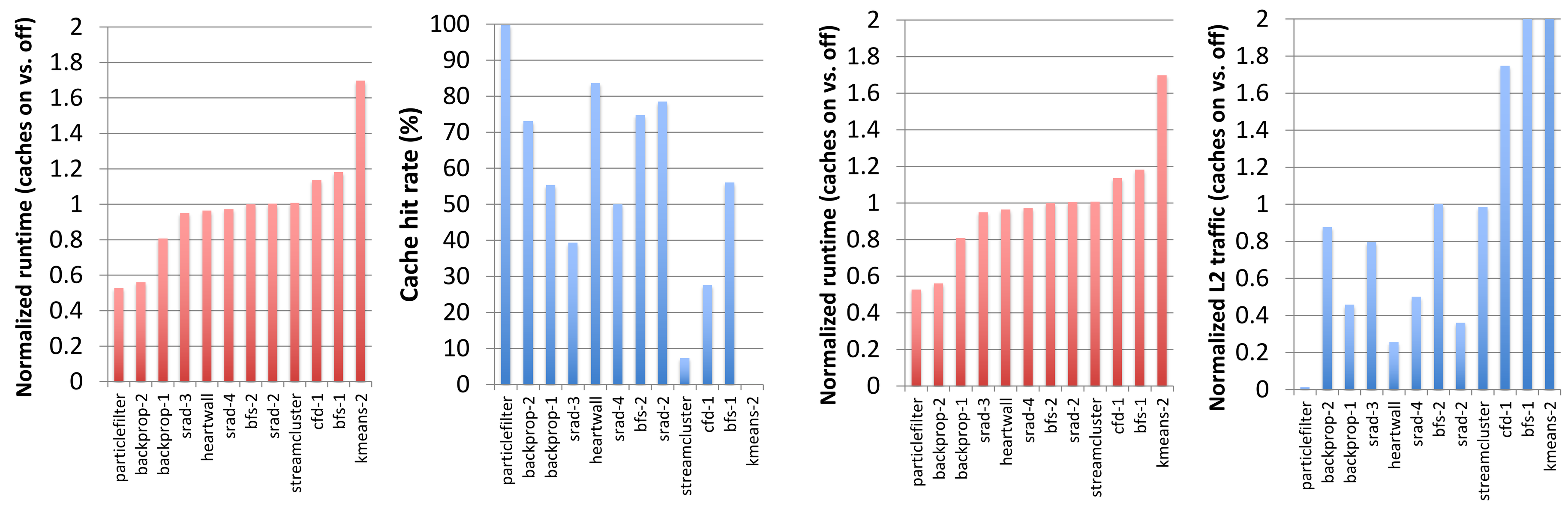


Class I kernels: Texture/constant loads only, requests don't use L1 caches
Class II kernels: Mainly use shared memory, limited benefits from caching
Class III kernels: Use DRAM and thus caches frequently, but see large and unpredictable performance variations from caching

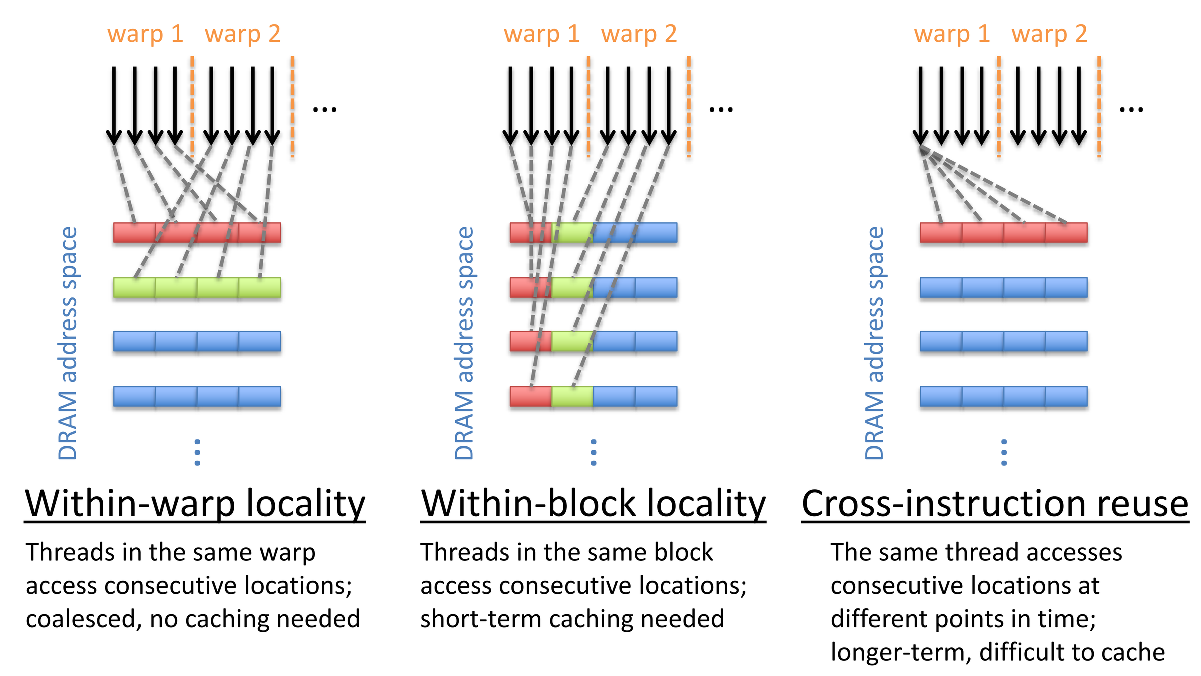
Our Work:
- Automate GPU cache payoff analysis.
- Guide per-instruction GPU cache configuration.

Analysis 1: Cache hit rate is a poor predictor of performance payoff.

Analysis 2: Cache-induced memory traffic reduction is a better performance predictor.



Result 1: We propose a GPU memory access locality taxonomy, which directly links memory access patterns to cache utility and performance impact.

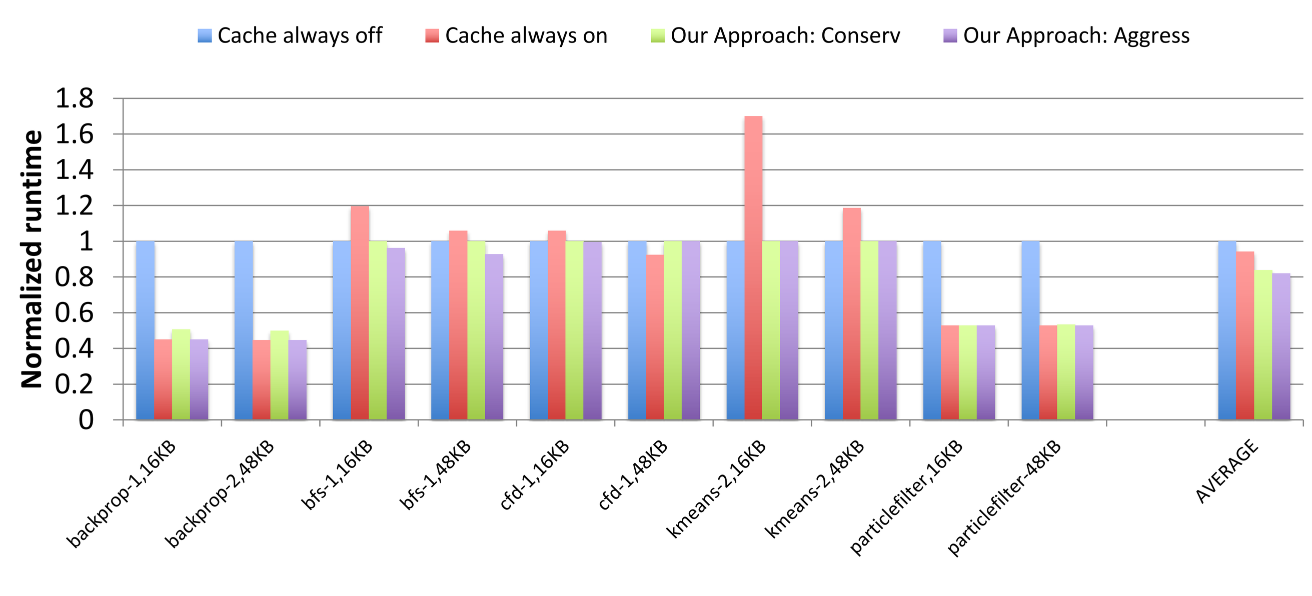


Result 2: Based on the taxonomy, a compile-time algorithm can intelligently enable or disable caching on a per-instruction basis to improve performance.

- Step 1: Compute load addresses of load instructions
- Step 2: Estimate cache-on and cache-off traffic
- Step 3: Decide whether to cache for each instruction based on cache-on traffic vs. cache-off traffic.

Load Inst	Expr	Requested Memory Addresses (Byte)				Cache-on Traffic	Cache-off Traffic	Cache?
		Warp 0	Warp 1	…	Warp 8			
A[index / 32]	tid / 32	0-3	4-7	…	28-31	128 B	32 × 8 = 256 B	Y
B[index * 32]	tid * 32	0-3 32-35 …	1024-1027 …	…	8196-8199 …	128 × 256 = 32 KB	32 × 256 = 8 KB	N
C[y]	∞							N

In real-system evaluation with an NVIDIA Tesla C2070, this algorithm improves the average benefit of caching from 5.8% to 18%.



Conclusions:
- Conserving memory bandwidth instead of hiding latency is GPU caches' main purpose.
- A locality-based taxonomy helps programmers and tools predict GPU cache utility.
- A compile-time caching control algorithm improves the benefit of caching by 3X.

Intel Science & Technology Center for Cloud Computing

Acknowledgement
In addition to Intel ISTC funding, this work was supported in part by the National Science Foundation under Grant No. CCF-0916971. The authors also acknowledge the support of the Gigascale Systems Research Center, one of six centers funded under the Focus Center Research Program (FCRP), a Semiconductor Research Corporation entity. Finally, we acknowledge equipment donations from NVIDIA and Intel.

Carnegie Mellon University · Georgia Tech · intel · Princeton University · UC Berkeley