

# Heterogeneous Parallelism and GPU Offloading: Optimization and Synchronization Challenges

Margaret Martonosi  
Princeton University




# Where are GPUs Used? Cellphones, Laptops, Desktops, and...



# Where are GPUs Used?

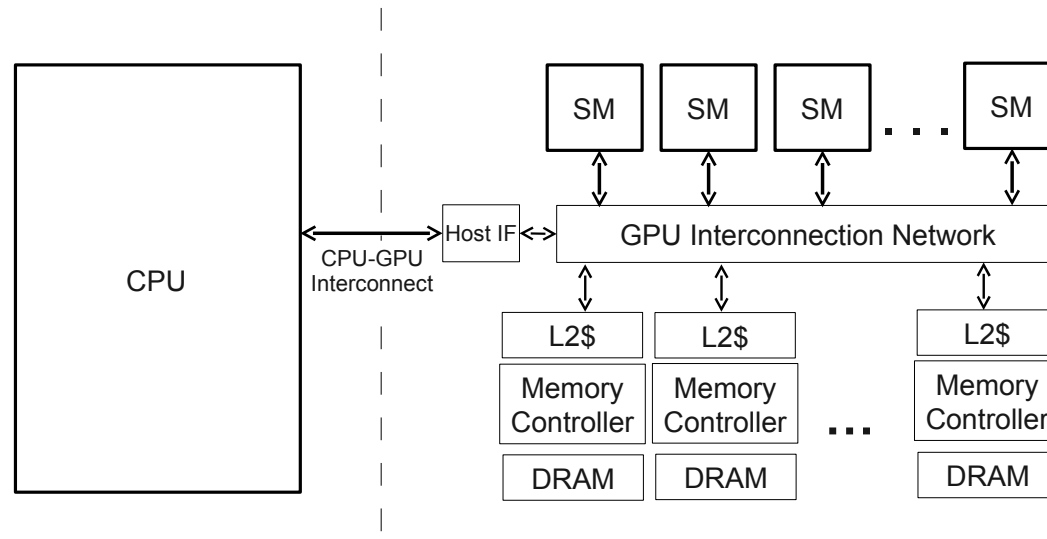
## Supercomputers and the Cloud

| TOP500 – Nov. 2012 |           |   |
|--------------------|-----------|---|
| Rank               | Computer  | Accelerator   |
| 1                  | Titan     |    |
| ...                |           |   |
| 7                  | Stampede  |   |
| 8                  | Tianhe-1A |  |

| GREEN500 – Nov. 2012 |          |   |
|----------------------|----------|---|
| Rank                 | Computer | Accelerator   |
| 1                    | Beacon   |    |
| 2                    | SANAM    |    |
| 3                    | Titan    |   |
| 4                    | Todi     |  |



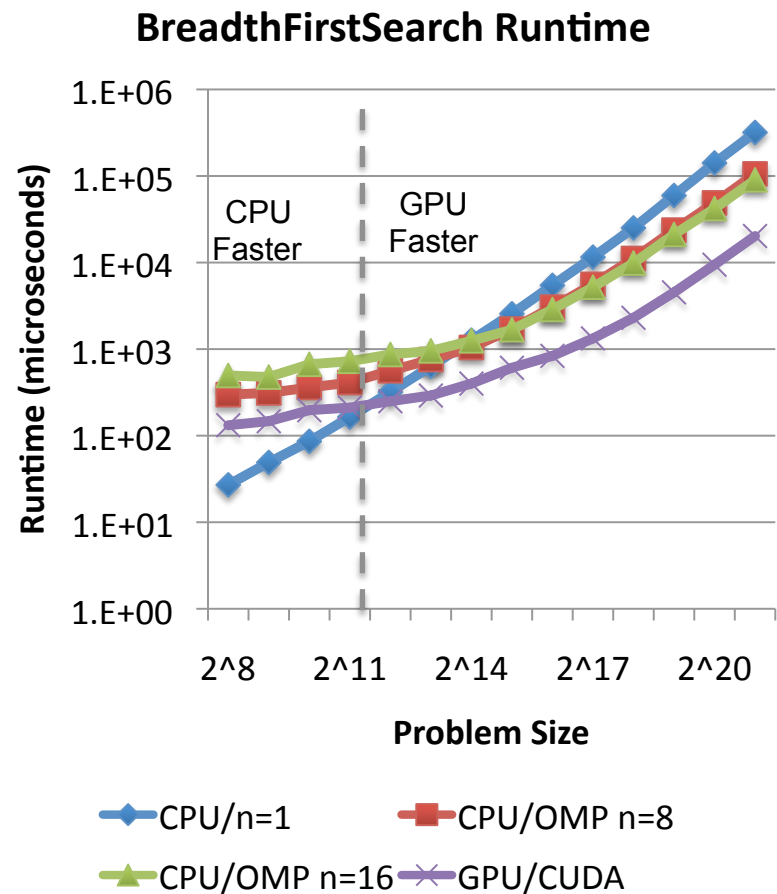
# CPU and GPU: So close and yet so far...



- Goal: Achieve heterogeneous parallelism by offloading (general-purpose) computation from CPU to GPU
- Promise: GPUs are great for highly parallel throughput-oriented workloads
- Reality: CPU $\leftrightarrow$ GPU latency too large.
  - GPUs not (yet!) broadly applicable to smaller, more latency-sensitive application

# Motivation #1: Offload Latency

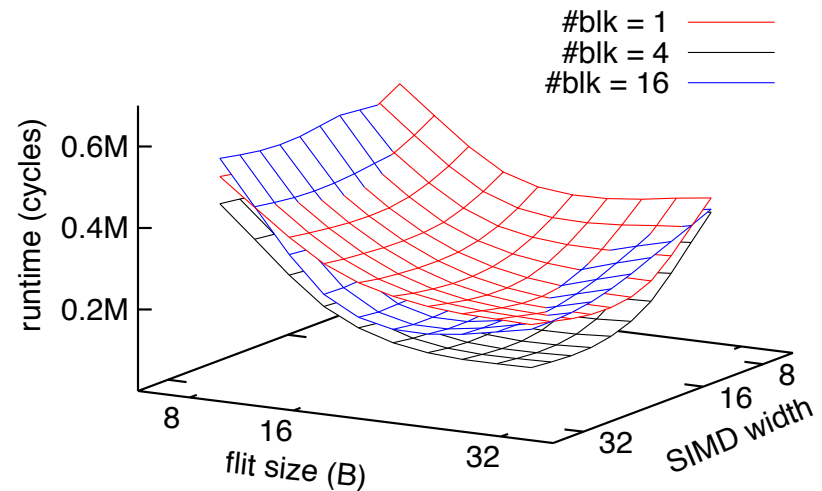
- As kernel size increases, eventually GPU is fastest
- Our goal: lower this threshold so more kernels see offload benefits



# Motivation #2: Offload Complexity

## Millions of design choices

- High-level: Should we offload to GPU or not?
- GPU software choices:
  - #threads
  - How/if to use programmer-controlled shared memory
  - should we use the instruction-configurable cache?
  - ...
- + Hardware design choices



Matrix Multiply—3 Factors



# This talk

- Mitigating Offload Latency
  - Improving GPU Synchronization Support
  - [Lustig & Martonosi. Paper to appear at HPCA 2013]
- Mitigating Offload Complexity
  - Statistical methods for design space exploration
  - [Jia, Shaw, & Martonosi. ISPASS 2012]



# This talk

- Mitigating Offload Latency
  - Improving GPU Synchronization Support
  - [Lustig & Martonosi. Paper to appear at HPCA 2013]
- Mitigating Offload Complexity
  - Statistical methods for design space exploration
  - [Jia, Shaw, & Martonosi. ISPASS 2012]





# Causes of GPU Offload Latency

1. Overhead of each API call
  - Runtime library + kernel driver
2. Interconnect overheads
  - For discrete cards in particular (PCIe)
3. Synchronization
  - Entire array must arrive at GPU (and be sync'ed) before any dependent kernel can even launch

“Isn't the offload problem solved by single-chip CPU-GPU systems?”

NO! #2 is smaller, but #1 and #3 remain!



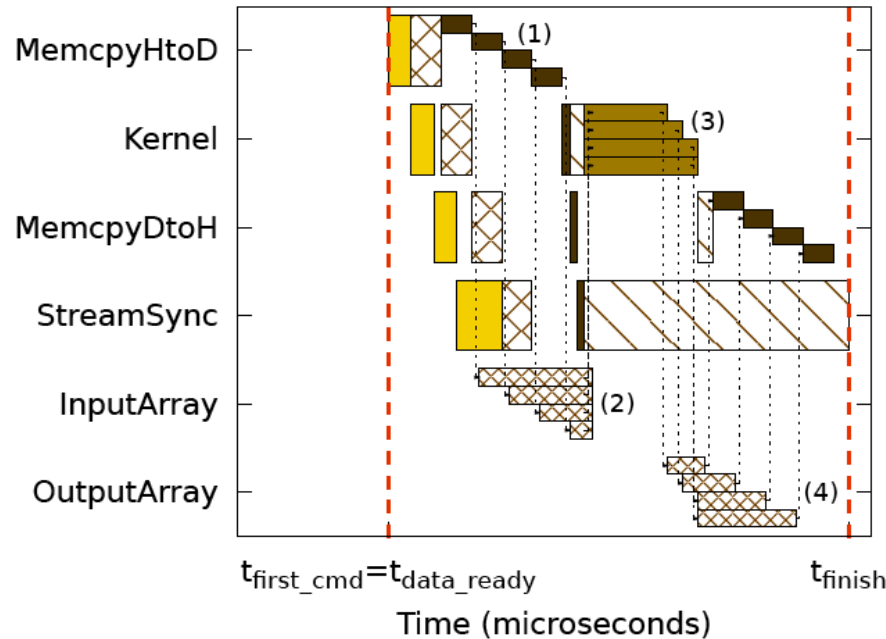
# Our Approach

- **Hardware and software support for fine-grained synchronization** as a means to reduce GPU offload latency
  - **Full/Empty bits in GPU DRAM** for fine-grained synchronization of CPU↔GPU data transfer
  - **API extensions of CUDA/OpenCL** to support proactive kernel launches and data transfers



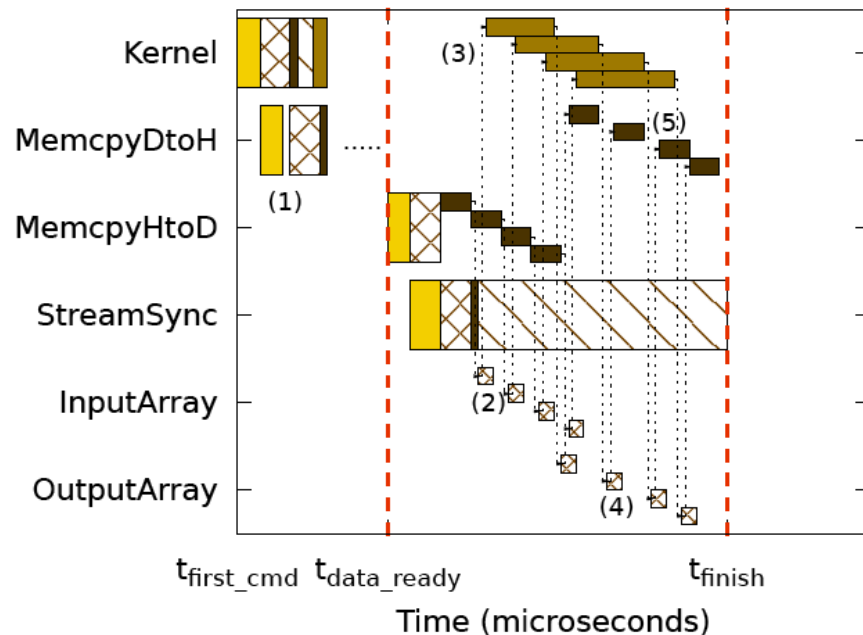
# Improving kernel launch and pipelining

Before

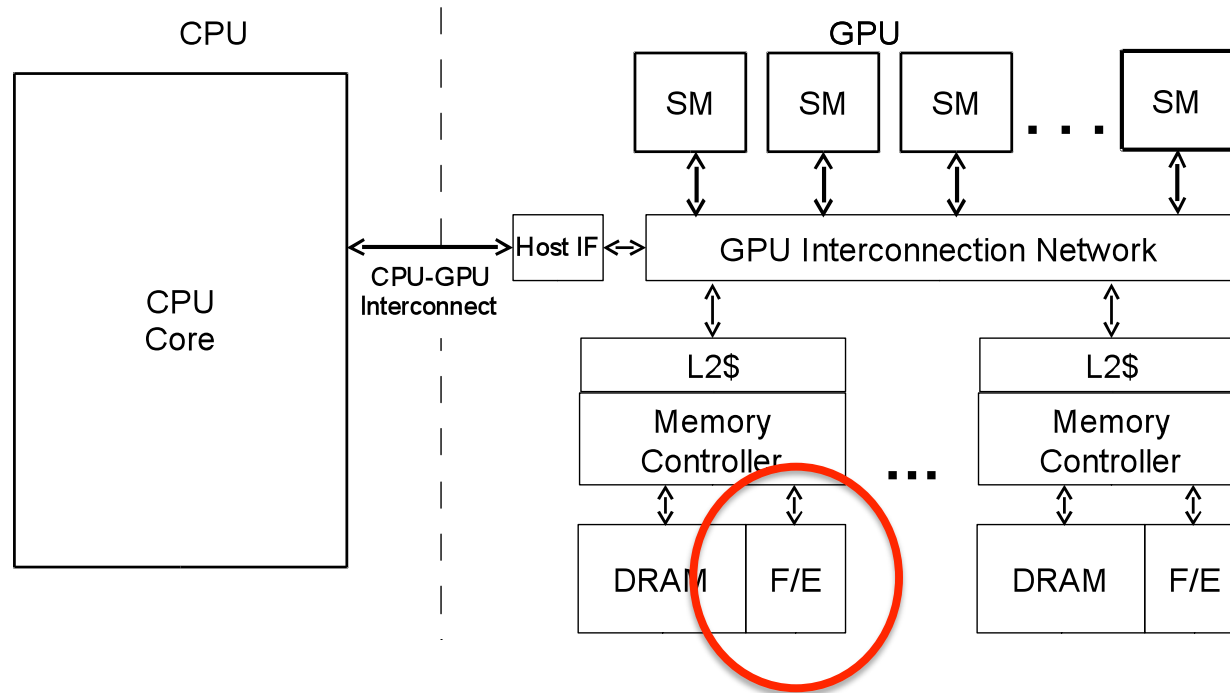


Goal:

- 1) Send kernel early.
- 2) Execution launches immediately, but stalls until data arrives at GPU.
- 3) Likewise, pipeline data return to CPU



# Hardware Support: Full/Empty Bits



- Drawing from Tera and MIT proposals of the 1990's...
- Associate full/empty bits with GPU memory
  - On the critical path for CPU-GPU communication



# F/E Bits: How they work?

## Basic Operation

- When a GPU thread accesses an “empty” location => Block.
- When a location becomes “full” => Unblock any waiting GPU threads.

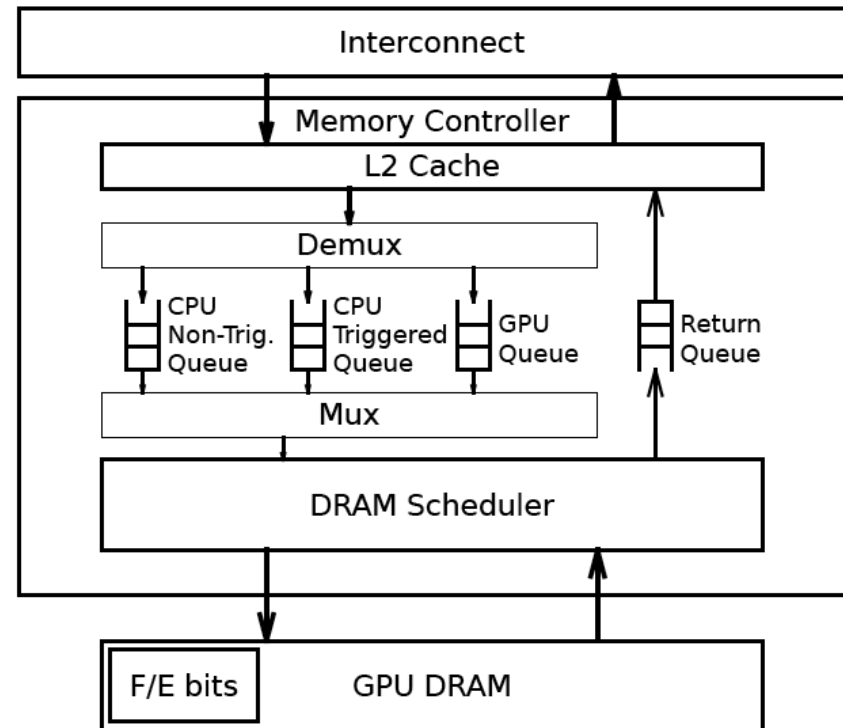
## Design Variations

- How are F/E bits initialized?
- Who can “fill” F/E bit? Just CPU or both CPU and GPU?
- Should F/E bit be cleared once the request is serviced?

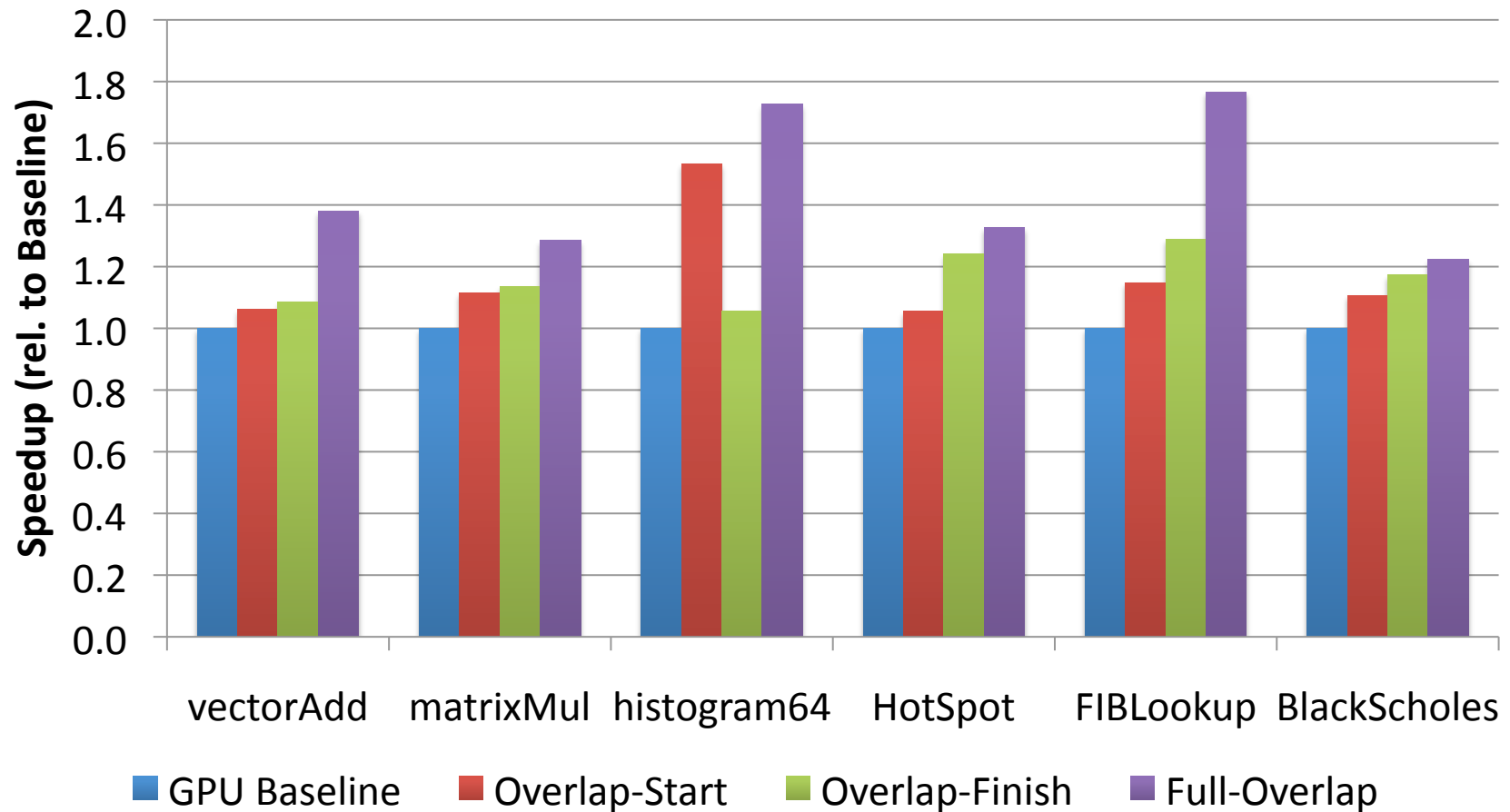


# Memory Controller

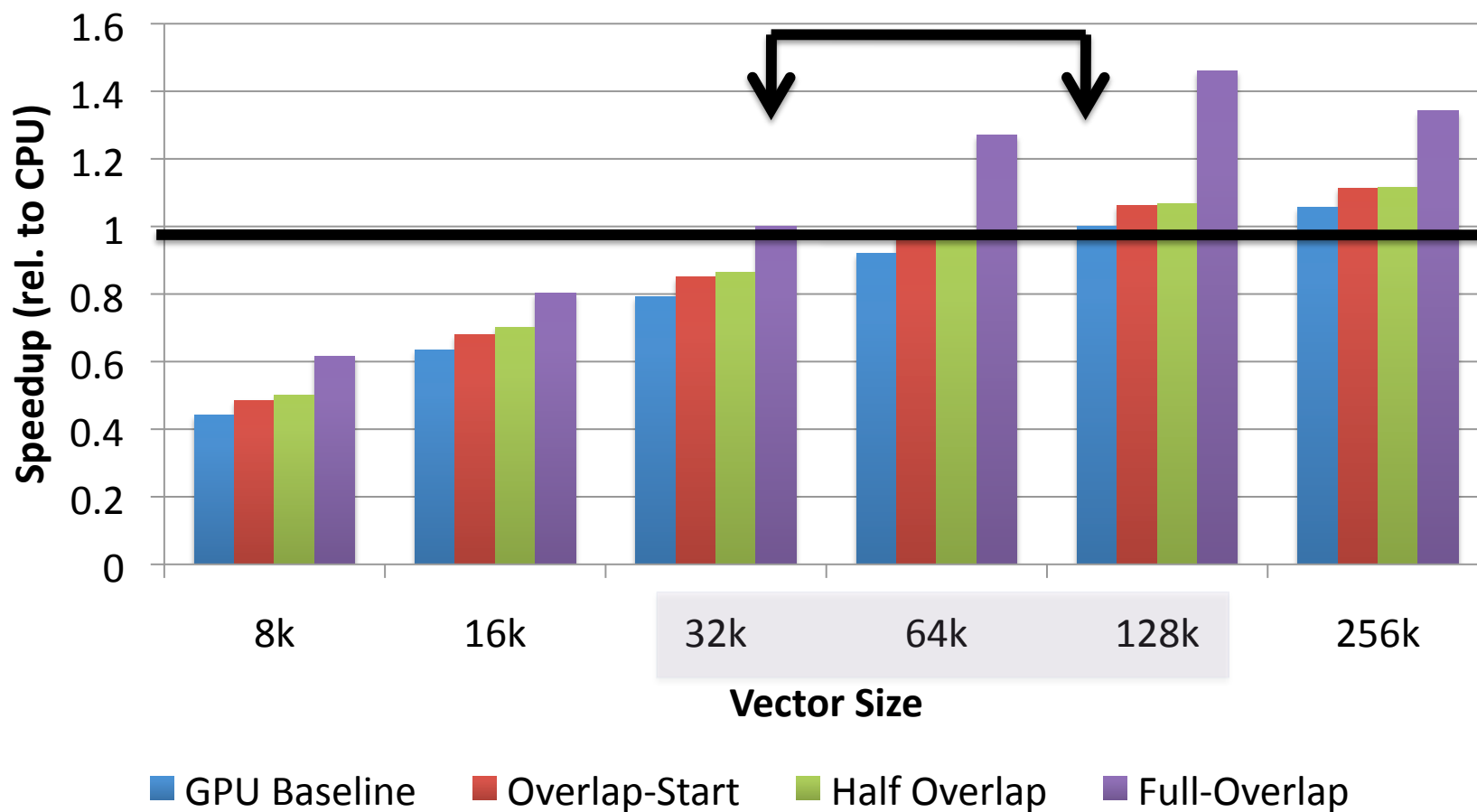
- Separate dependent requests into different queues for CPU reads, CPU writes, and GPU writes
- Avoids head-of-line blocking
- Allows simple implementation:
  - Only watch head of queue
- Aggressive: Watch first-N entries in queue



# Result: 1.2-1.8X Speedup



# Shifting CPU-GPU Crossover point: VectorAdd Example





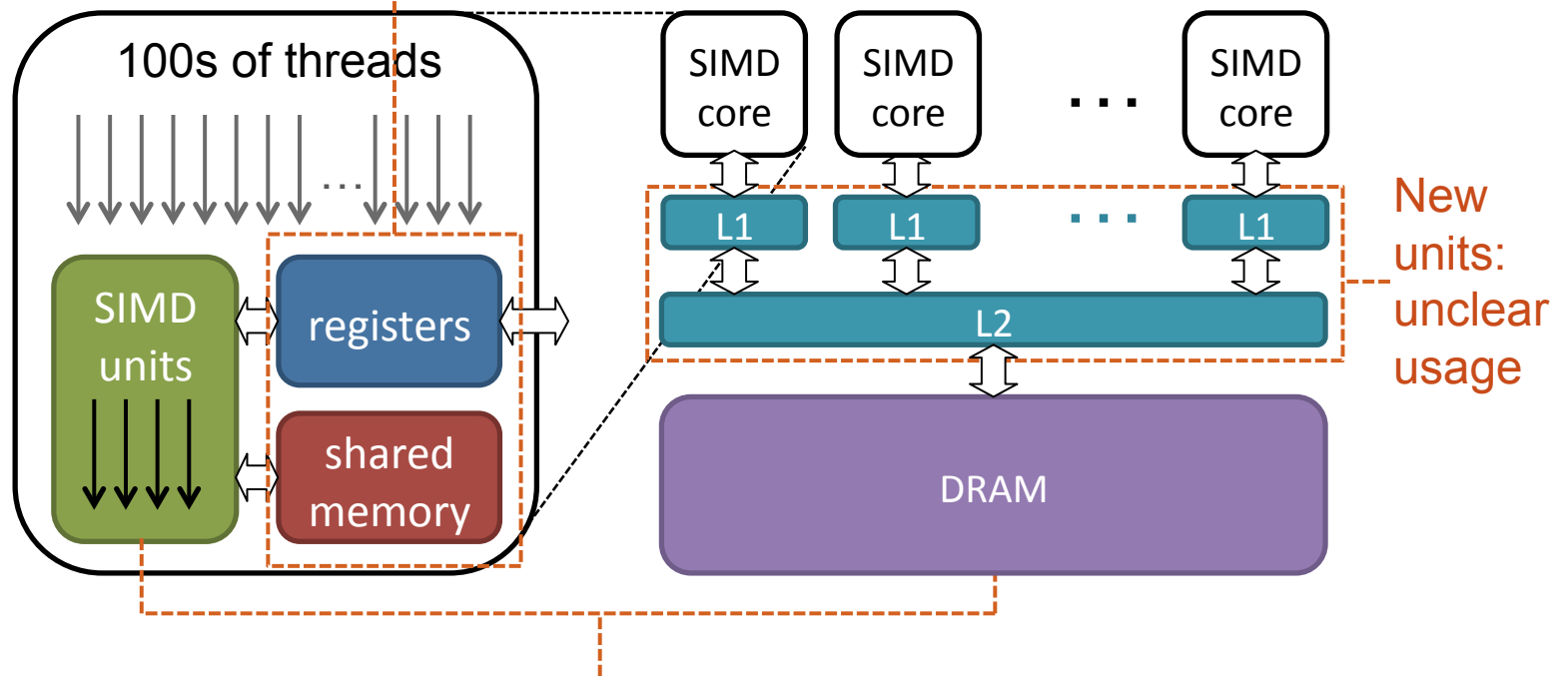
# This talk

- Mitigating Offload Latency
  - Improving GPU Synchronization Support
  - [Lustig & Martonosi. Paper to appear at HPCA 2013]
- Mitigating Offload Complexity
  - Statistical methods for design space exploration
  - [Jia, Shaw, & Martonosi. ISPASS 2012]



# GPU Design Complexity

Lack of resource abstraction:  
on-chip storage size limits thread  
count



Large number of concurrent  
threads: compute vs. memory  
trade-off

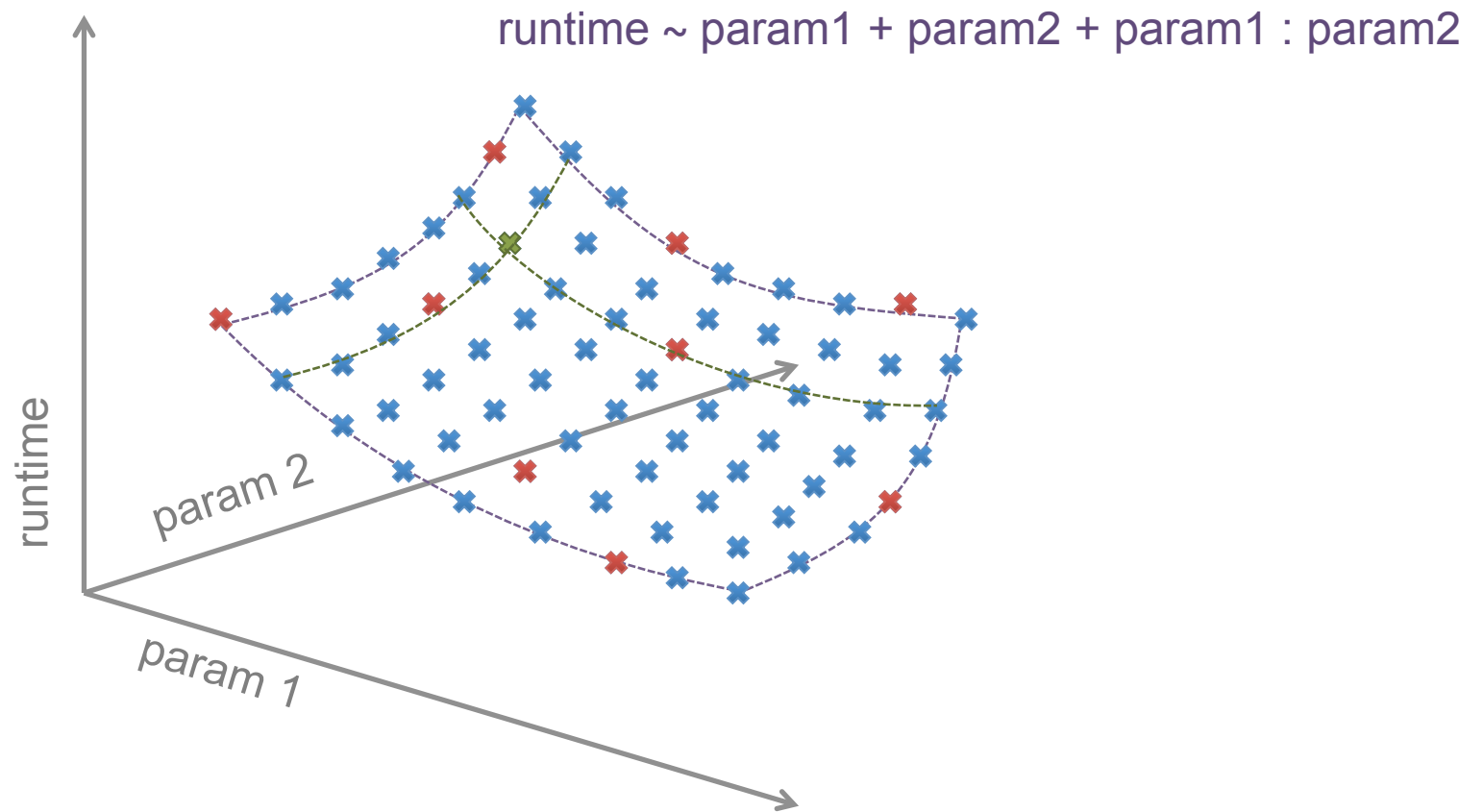


# Stargazer: Design Space Exploration

- Effective statistical regression-based GPU design space exploration framework
  - **Automated**: Automatically discover significant factors and their interactions
  - **Efficient**: Up to 15000× speed-up vs. exhaustive exploration
  - **Accurate**: 1.1% average prediction error when only 0.03% of the space is sampled
- Example uses of Stargazer
  - Design space pruning
  - Application characterization



# Regression: Sample–Model–Predict



Regression builds an application-specific performance model which can predict GPU performance through interpolation



# Stargazer Overview

- **Sample** randomly and uniformly across design parameters: SIMD width, memory bandwidth, concurrent block count, ...
  - Space is large:  $|P1| \times |P2| \times \dots \times |Pn|$  points!
- **Measure** performance or power at each sample point
- **Model**: runtime  $\sim f(P1) + g(P2) + \dots + h(Pn) + q(P1 \times P2) + r(P1 \times P3) + \dots + z(Pn-1 \times Pn)$
- **Stargazer**: Automatically discern most-relevant factors and pairwise interactions



# Stargazer: The Stepwise Algorithm

```
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 R2
  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

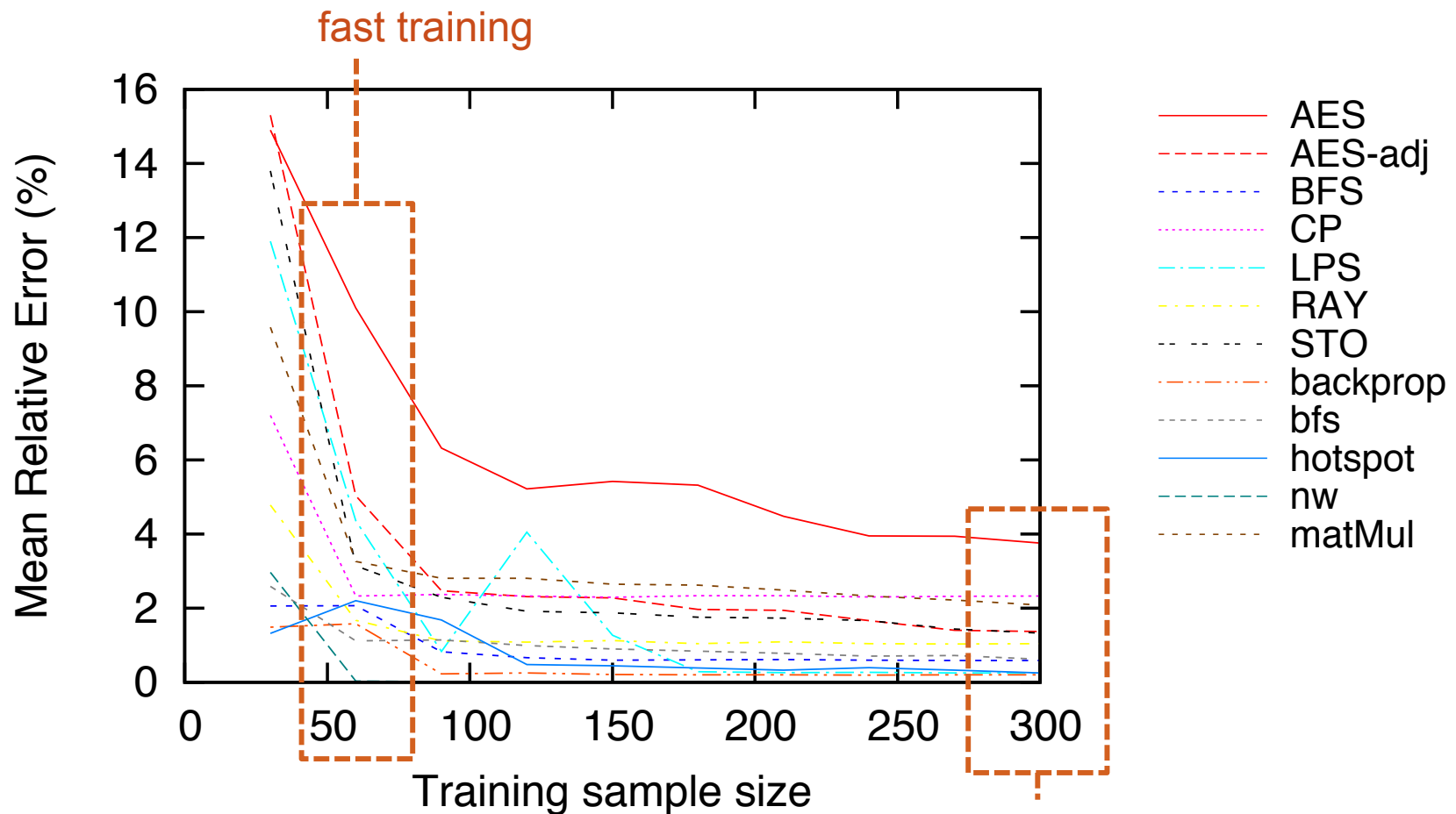
If the next most significant factor indeed affects runtime, include it in the model

Also test its interactions with included factors

Else exit the routine



# Prediction Accuracy vs. Sample Size

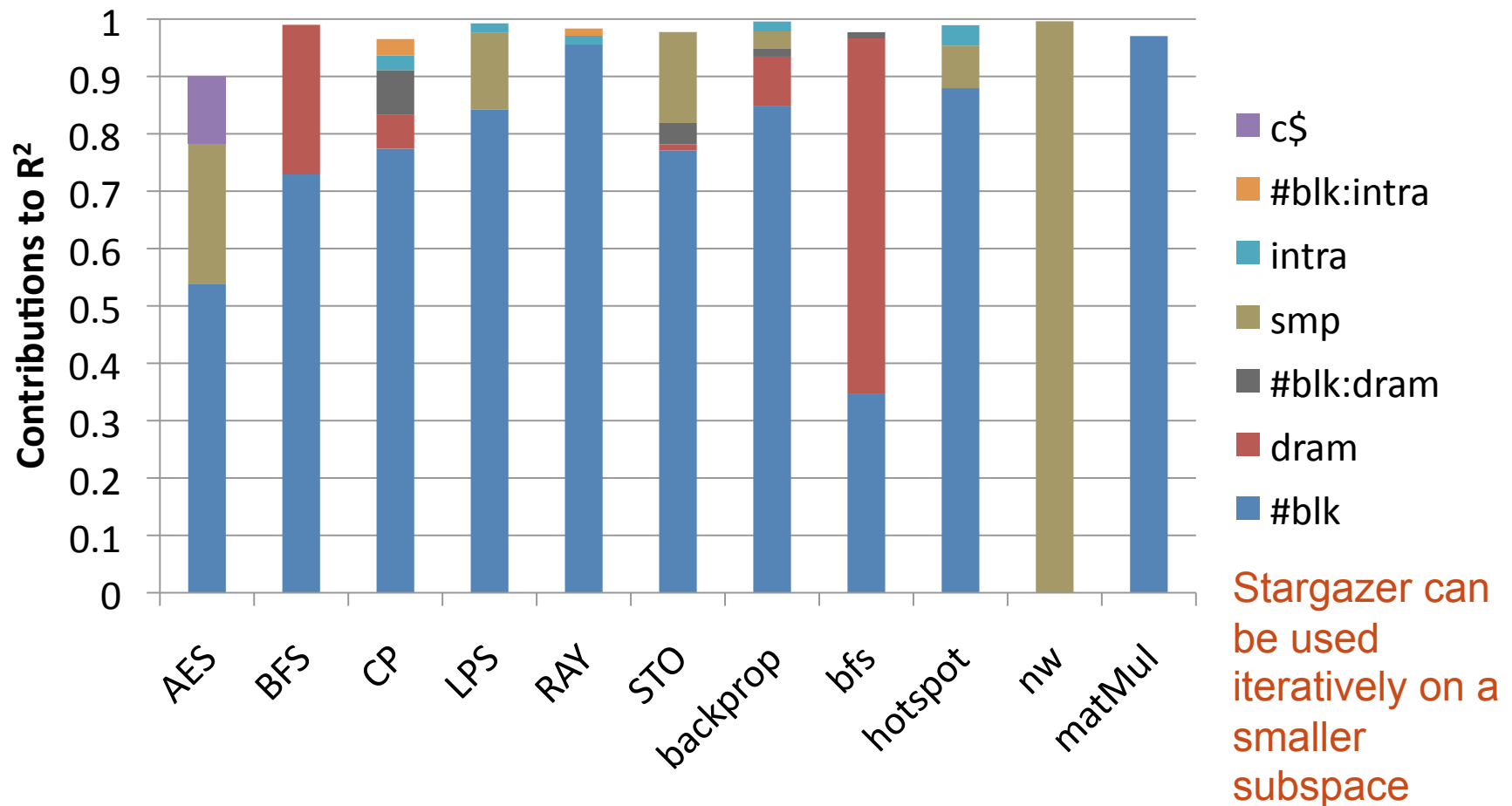


good accuracy (only 0.03% of the whole space!)



Test set size: 200  
Repeated 5 times

# At SIMD = 32: Diverse Secondary Factors





# Stargazer: Summary

- Reduce design exploration time
  - Automatically prune design space
  - 30–60 samples: < 5% error for most programs
  - Up to 15000× simulation time reduction (60 samples)
- Application characterization
  - Can be used to tune parameters
  - Can be used to plan offloads and schedules



# Overall Research Focus

- **How and where to compute?**
  - Offloading and planning on heterogeneous platforms
  - On-chip, on-device, and cross-cloud
  - Manage performance, power, ...
- **How and where to communicate?**
  - Scheduling and heuristics for planning off-device communication
  - Optimize latency, bandwidth, energy, cost.
- **Pushing towards a new Hardware-Software Contract**
  - For architects: Need abstractions above ISA to manage portability across different levels of capability and specialization.
  - For systems folks: Nimble shift portions of computation across very diverse and distributed heterogeneous pool of resources.



# Acknowledgments

- Students:
  - Elba Garza, Tae Jun Ham, Ali JavadiAbhari, **Wenhao Jia**, **Dan Lustig**, Ozlem Bilgir Yetim, Yavuz Yetim.
- Other Collaborators:
  - Ramon Caceres, Sibren Isaacman, Manos Koukoumidis, **Kelly Shaw**, Kevin Skadron, Ke Wang.

