A CPU-GPU Hybrid Implementation and Model-Driven Scheduling of the Fast Multipole Method

Jee Choi[1], Aparna Chandramowlishwaran[3], Kamesh Madduri[4], and Richard Vuduc[2]

[1]ECE, Georgia Tech [2]CSE, Georgia Tech [3]CSAIL, MIT [4]CSE PSU

March 1, 2014

Presented at GPGPU7, Salt Lake City, Utah

Georgia Tech College of Computing
Computational Science and Engineering

hpcgarage

Why?

- Importance
  - One of the most important algorithms in scientific computing
- Performance
  - Various phases of the Fast Multipole Method show different performance characteristics
- Power and energy
  - Everyone has a strong suit
- Just because we can
  - CPU(s) come bundled with GPU(s) (or is it vice versa?)

Contributions

- Optimized implementations of FMM for both CPUs and GPUs

Contributions

- Optimized implementations of FMM for both CPUs and GPUs
- Analytical performance model

Contributions

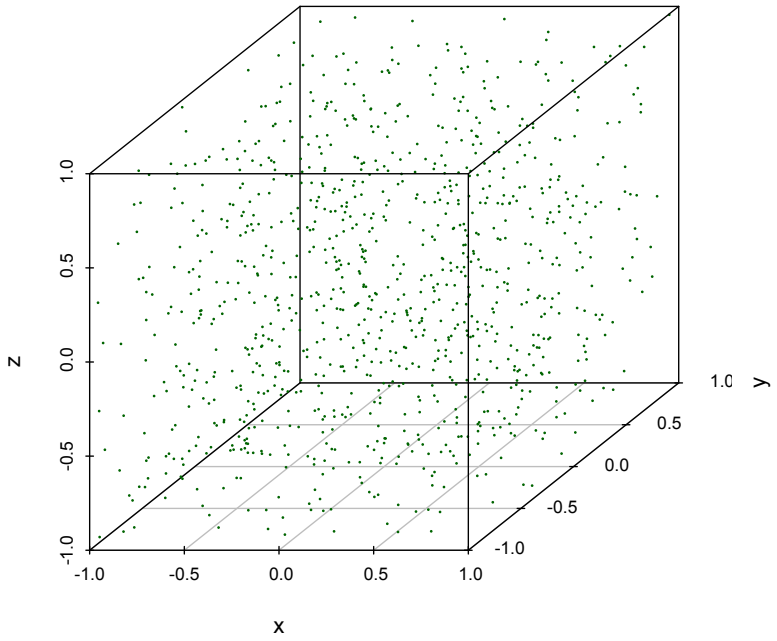
- Optimized implementations of FMM for both CPUs and GPUs
- Analytical performance model
- CPU-GPU hybrid implementation of FMM

Contributions

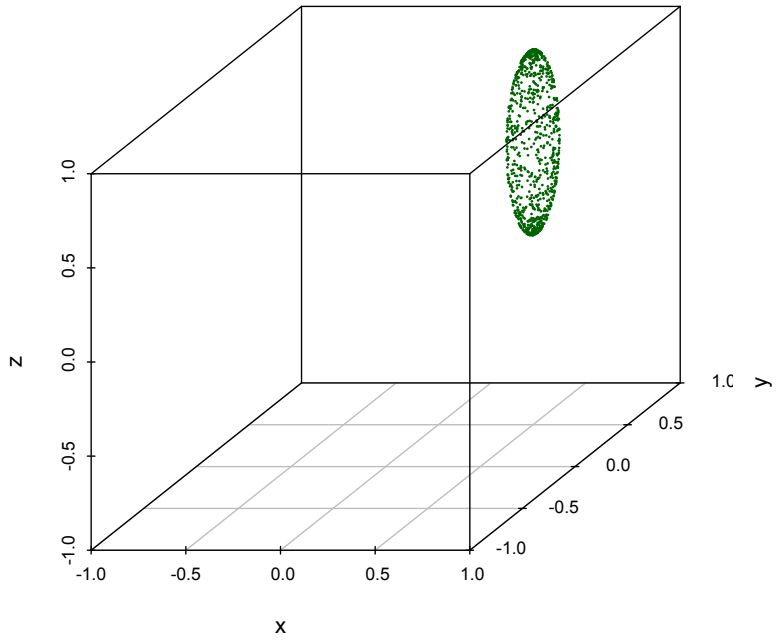
- Optimized implementations of FMM for both CPUs and GPUs
- Analytical performance model
- CPU-GPU hybrid implementation of FMM
  - Uses our analytical performance model to automatically controls various FMM-specific tuning knobs ***and*** maps phases to platforms

Contributions

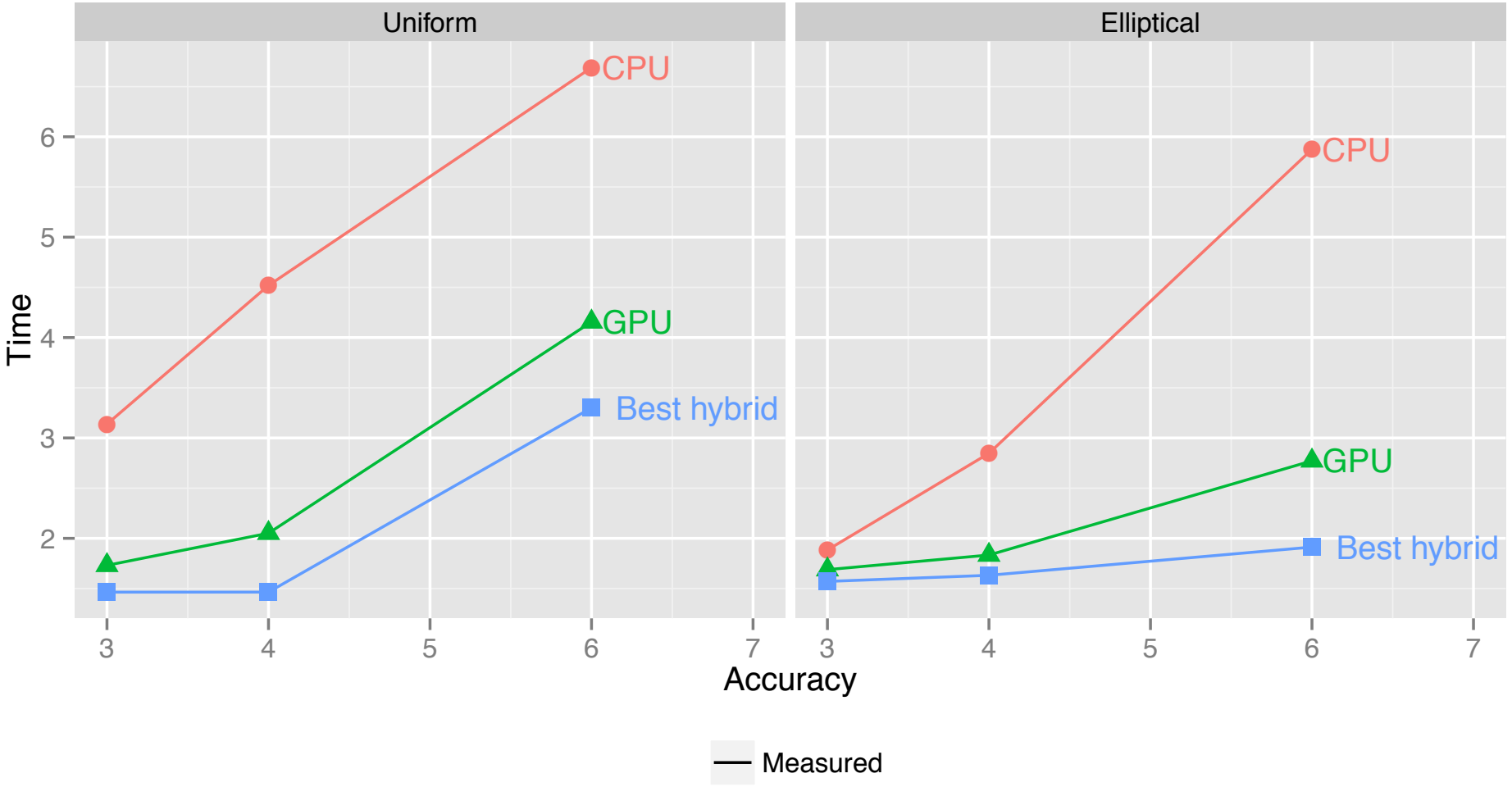
Uniform



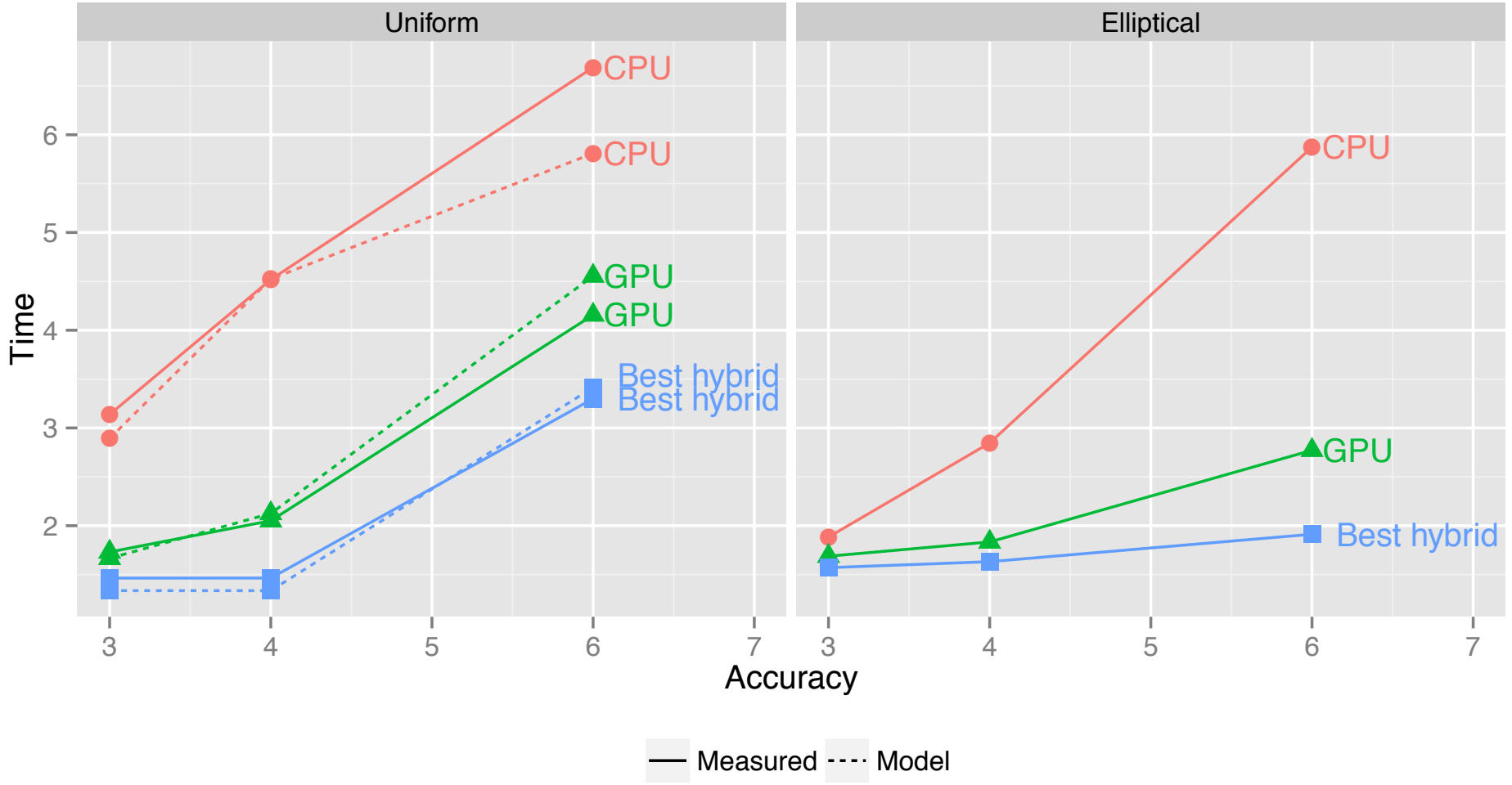
Elliptical



Summary of Results



Summary of Results



Limitations

- Analytical performance model is limited to *uniform* distribution of points
  - Elliptical distribution is more difficult to model
  - Model was driven by hand
- Hybrid scheduling is done by hand
  - No scheduler implementation

Overview

- Algorithmic characteristics
- GPU performance model
- Implementation
- Hybrid scheduling
- Exascale projections

Overview

- **Algorithmic characteristics**
- GPU performance model
- Implementation
- Hybrid scheduling
- Exascale projections

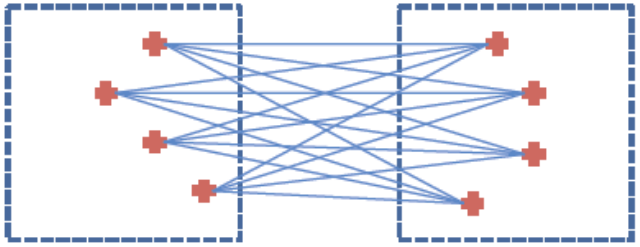
The problem

- Given a system of N source points with positions $\{ y_1, \ldots, y_N \}$ and N target points $\{ x_1, \ldots, x_N \}$
- We want to compute the N target sums,

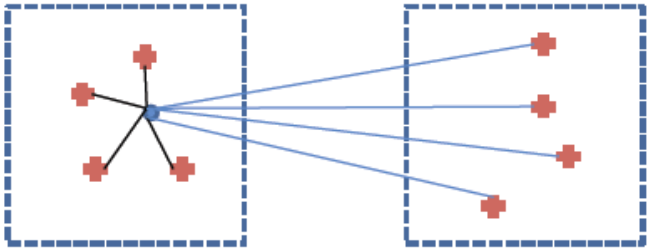
$$f(x_i) = \sum_{j=1}^{N} K(x_i, y_i) \cdot s(y_j), \qquad i = 1, \ldots, N$$

Direct vs. Tree-based

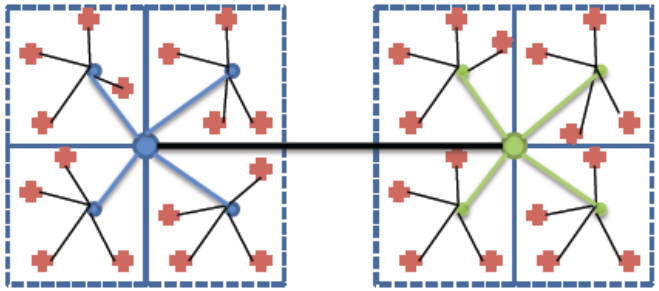
Direct evaluation: O(N^2)



Barnes-Hut: O($N \log N$)

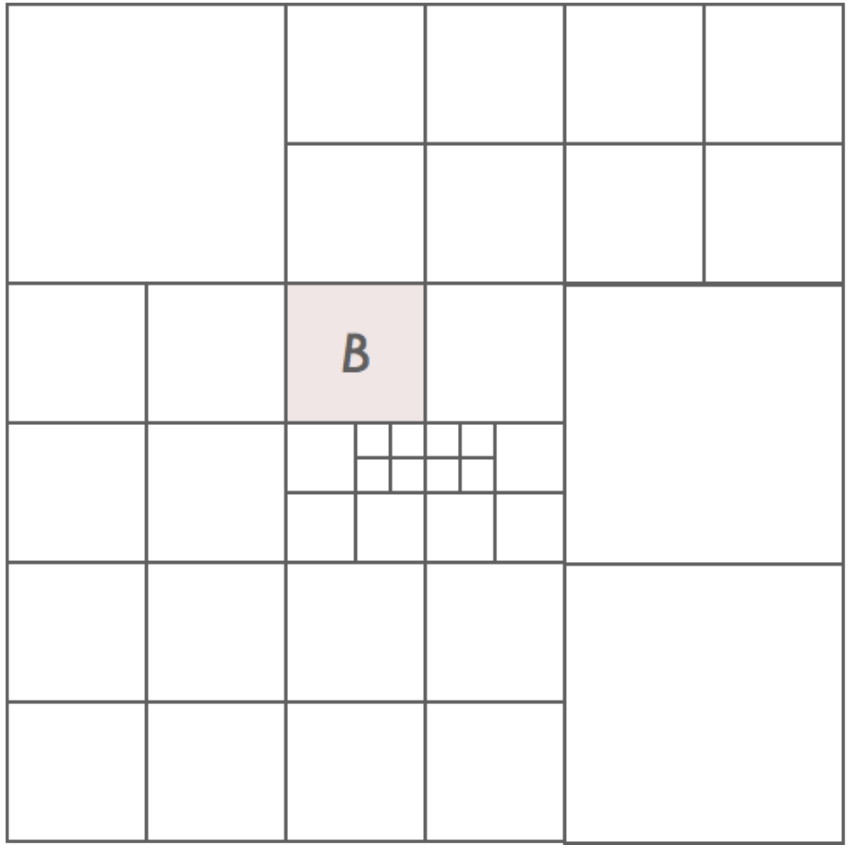


Fast Multipole Method (FMM): O(N)



Fast Multipole Method (FMM)

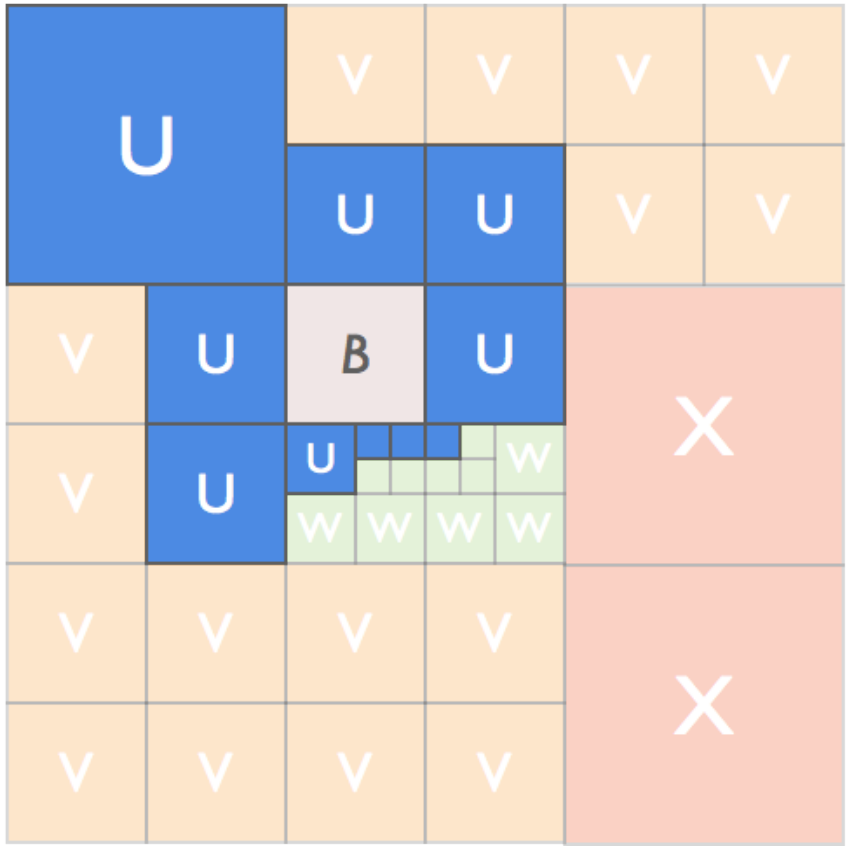
- Tree Construction
  - Recursively divide space until each box has at most ***q*** points
- Evaluation (Uniform)
  - Upward
  - U-List
  - V-List
  - Downward
- Phases vary in:
  - Data parallelism
  - Compute intensity



U-List

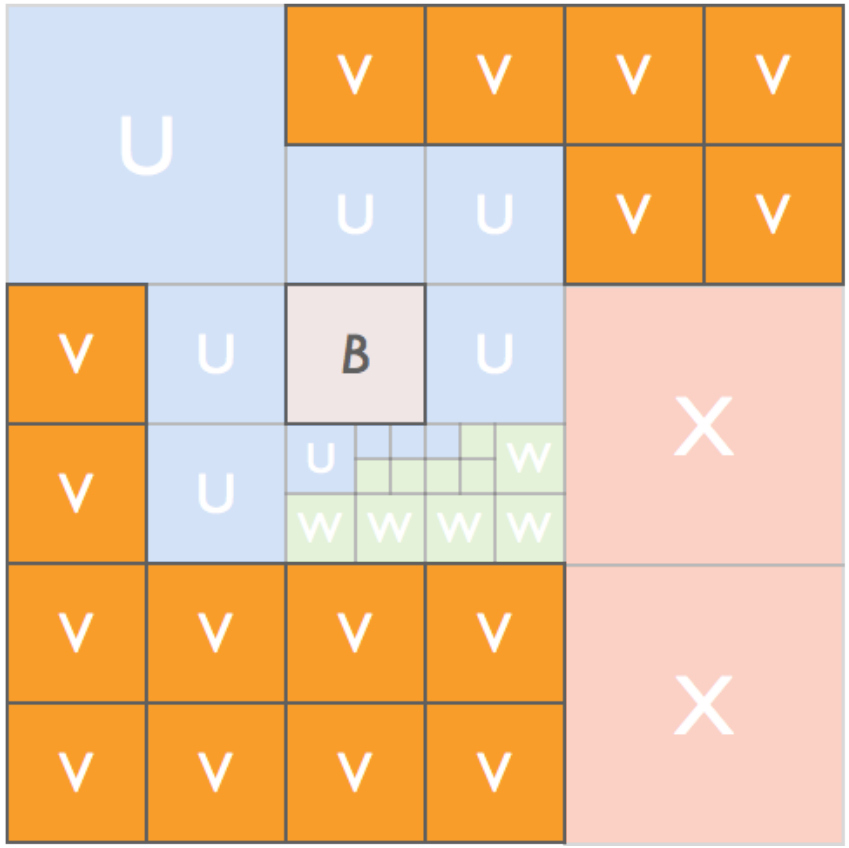
- Direct $B \otimes U$:

→ $O(q^2)$ *flops* : $O(q)$ *mops*



V-List

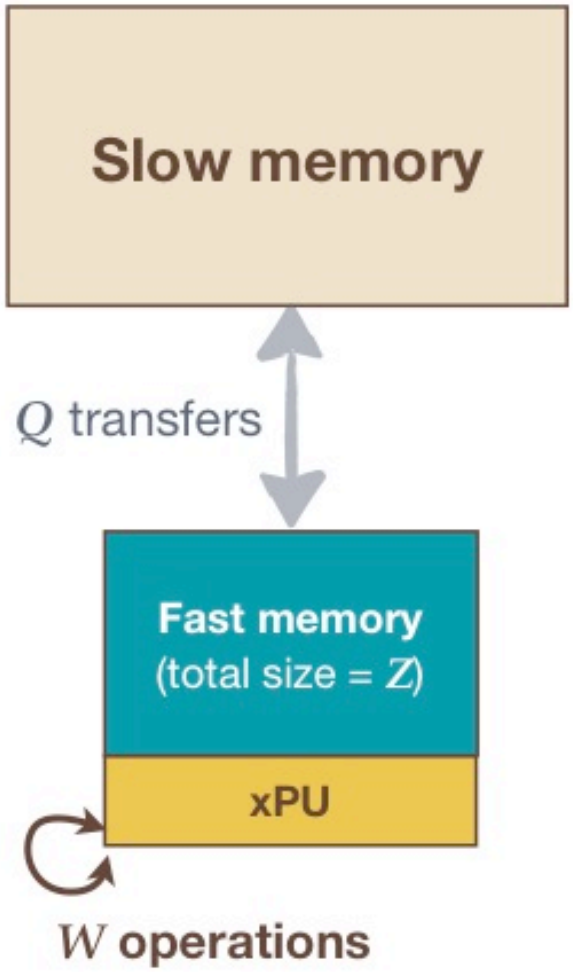
- 3-D FFT
- Point-wise multiplication
- 3-D IFFT



Overview

- Algorithmic characteristics
- **GPU performance model**
- Implementation
- Hybrid scheduling
- Exascale projections

Machine Model



CPU Performance Model

U-List

$$T_{comp,u} = \frac{C_u.(3b^{1/3} - 2)^3.q^2}{C_0}$$

$$T_{mem,u} = \frac{C_1 n}{\beta_{mem}} + \frac{C_2 n L}{\beta_{mem}(Z^{\frac{1}{3}} q^{\frac{2}{3}})}$$

V-List

$$T_{comp,v} = \frac{C_v\, k\, b\, p^{\frac{3}{2}}}{C_0}$$

$$T_{mem,v} = \frac{C_1 n p^{\frac{3}{2}}}{q\beta_{mem}} + \frac{C_2 n p^{\frac{1}{2}} L}{(Z'^{\frac{1}{3}} q)\beta_{mem}}$$

GPU Performance Model

U-List

$$T_{comp,u} = \frac{C_u.(3b^{1/3} - 2)^3.q^2}{C_0}$$

$$T_{mem,u} = \frac{C_1 n}{\beta_{mem}} + \frac{C_2 n L}{\beta_{mem}(Z^{\frac{1}{3}} q^{\frac{2}{3}})}$$

$$T_{u,gpu} = \frac{C_{u,gpu}\left(3b^{1/3} - 2\right)^3 q^2}{C_{peak,gpu}}$$

V-List

$$T_{comp,v} = \frac{C_v\, k\, b\, p^{\frac{3}{2}}}{C_0}$$

$$T_{mem,v} = \frac{C_1 n p^{\frac{3}{2}}}{q\beta_{mem}} + \frac{C_2 n p^{\frac{1}{2}} L}{(Z'^{\frac{1}{3}} q)\beta_{mem}}$$

GPU Performance Model

U-List

$$T_{comp,u} = \frac{C_u.(3b^{1/3} - 2)^3.q^2}{C_0}$$

$$T_{mem,u} = \frac{C_1 n}{\beta_{mem}} + \frac{C_2 n L}{\beta_{mem}(Z^{\frac{1}{3}} q^{\frac{2}{3}})}$$

$$T_{u,gpu} = \frac{C_{u,gpu}\left(3b^{1/3} - 2\right)^3 q^2}{C_{peak,gpu}}$$

V-List

$$T_{comp,v} = \frac{C_v\, k\, b\, p^{\frac{3}{2}}}{C_0}$$

$$T_{mem,v} = \frac{C_1 n p^{\frac{3}{2}}}{q\beta_{mem}} + \frac{C_2 n p^{\frac{1}{2}} L}{(Z'^{\frac{1}{3}} q)\beta_{mem}}$$

$$T_{v,gpu} = \frac{C_{1,gpu} n p^{\frac{3}{2}}}{q\beta_{mem,gpu}} + \frac{C_{2,gpu} n p^{\frac{1}{2}} L}{\left(Z'^{\frac{1}{3}}\right)\beta_{mem,gpu}}$$

GPU Performance Model

U-List

$$T_{comp,u} = \frac{C_u.(3b^{1/3} - 2)^3.q^2}{C_0}$$

$$T_{mem,u} = \frac{C_1 n}{\beta_{mem}} + \frac{C_2 n L}{\beta_{mem}(Z^{\frac{1}{3}} q^{\frac{2}{3}})}$$

$$T_{u,gpu} = \frac{C_{u,gpu}\left(3b^{1/3} - 2\right)^3 q^2}{C_{peak,gpu}}$$

V-List

$$T_{comp,v} = \frac{C_v\, k\, b\, p^{\frac{3}{2}}}{C_0}$$

$$T_{mem,v} = \frac{C_1 n p^{\frac{3}{2}}}{q\beta_{mem}} + \frac{C_2 n p^{\frac{1}{2}} L}{(Z'^{\frac{1}{3}} q)\beta_{mem}}$$

$$T_{v,gpu} = \frac{C_{1,gpu} n p^{\frac{3}{2}}}{q\beta_{mem,gpu}} + \frac{C_{2,gpu} n p^{\frac{1}{2}} L}{\left(Z'^{\frac{1}{3}}\right)\beta_{mem,gpu}}$$

GPU Performance Model

U-List

$$T_{comp,u} = \frac{C_u.(3b^{1/3} - 2)^3.q^2}{C_0}$$

$$T_{mem,u} = \frac{C_1 n}{\beta_{mem}} + \frac{C_2 nL}{\beta_{mem}(Z^{\frac{1}{3}} q^{\frac{2}{3}})}$$

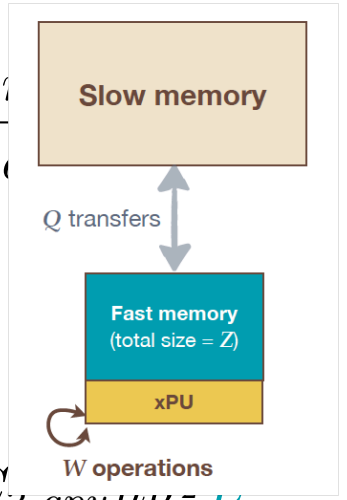
$$T_{u,gpu} = \frac{C_{u,gpu}\left(3b^{1/3} - 2\right)^3 q^2}{C_{peak,gpu}}$$

V-List

$$T_{comp,v} = \frac{C_v\, k\, b\, p^{\frac{3}{2}}}{C_0}$$

$$T_{mem,v} = \frac{C_1 np^{\frac{3}{2}}}{q\beta_{mem}} + \frac{C_2}{(Z'^{\frac{1}{3}}}$$

$$T_{v,gpu} = \frac{C_{1,gpu} np^{\frac{3}{2}}}{q\beta_{mem,gp}} \frac{C_{2,gpu}}{\left(Z'^{\frac{1}{3}}\right)\beta_{mem,gpu}}$$



GPU Performance Model

U-List

$$T_{comp,u} = \frac{C_u.(3b^{1/3} - 2)^3.q^2}{C_0}$$

$$T_{mem,u} = \frac{C_1 n}{\beta_{mem}} + \frac{C_2 n L}{\beta_{mem}(Z^{\frac{1}{3}} q^{\frac{2}{3}})}$$

$$T_{u,gpu} = \frac{C_{u,gpu}\left(3b^{1/3} - 2\right)^3 q^2}{C_{peak,gpu}}$$

V-List

$$T_{comp,v} = \frac{C_v\, k\, b\, p^{\frac{3}{2}}}{C_0}$$

$$T_{mem,v} = \frac{C_1 n p^{\frac{3}{2}}}{q\beta_{mem}} + \frac{C_2 n p^{\frac{1}{2}} L}{(Z'^{\frac{1}{3}} q)\beta_{mem}}$$

Why doesn't this work for **V-List**?

Small LLC on GPUs – can only fit ~50 translation vectors

GPU Performance Model

U-List

$$T_{comp,u} = \frac{C_u.(3b^{1/3} - 2)^3.q^2}{C_0}$$

$$T_{mem,u} = \frac{C_1 n}{\beta_{mem}} + \frac{C_2 n L}{\beta_{mem}(Z^{\frac{1}{3}} q^{\frac{2}{3}})}$$

$$T_{u,gpu} = \frac{C_{u,gpu}\left(3b^{1/3} - 2\right)^3 q^2}{C_{peak,gpu}}$$

V-List

$$T_{comp,v} = \frac{C_v\, k\, b\, p^{\frac{3}{2}}}{C_0}$$

$$T_{mem,v} = \frac{C_1 n p^{\frac{3}{2}}}{q\beta_{mem}} + \frac{C_2 n p^{\frac{1}{2}} L}{(Z'^{\frac{1}{3}} q)\beta_{mem}}$$

$$T_{v,gpu} = \frac{C_{v,gpu}\left(3bp^{3/2}\right) 189}{\beta_{mem,gpu}}$$

GPU Performance Model

Upward

$$T_{up,gpu} = \frac{C_{up,gpu}\left(4N + 2bf_1\left(p\right)\left(f_2\left(p\right)+1\right)\right)}{\beta_{mem,gpu}}$$

Downward

$$T_{down,gpu} = \frac{C_{down,gpu}\left(N + 2b\left(f_1\left(p\right)\right)^2 + 2bf_1\left(p\right)\right)}{\beta_{mem,gpu}}$$

GPU Performance Model

- ***Real*** peak memory throughput
  - Optimized streaming μbenchmark
  - Relatively close to specification (80-90%)
- ***Real*** Peak compute throughput
  - Misleading
    - Requires that ***fused multiply add*** (FMA) be issued by ***every*** scheduler at ***every*** cycle
    - ***No*** hardware SFU for double-precision (e.g., reciprocal, square root, etc.)

GPU Performance Model

- *U-list*
  - Inner-loop executes (in double-precision)
    - 3 *subtracts*
    - *1 add*
    - *1 multiply*
    - 2 *multiply-adds*
    - **1 *reciprocal square root***

GPU Performance Model

- *U-list*
  - Inner-loop executes (in double-precision)
    - 3 *subtracts*
    - *1 add*
    - *1 multiply*
    - *2 multiply-adds*
    - **1 *reciprocal square root***

How expensive is it?

GPU Performance Model

- μbenchmarking study
  - Reciprocal square root (in double-precision)
    - ~14 cycle latency, or equivalently ~14 *independent* instructions
    - It takes 14 + 7 instructions to execute 11 FLOPs

GPU Performance Model

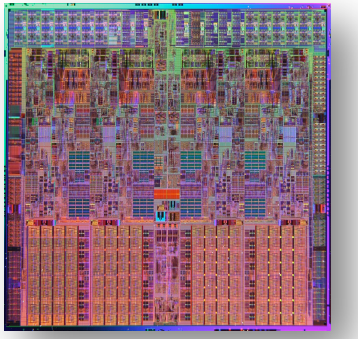
- μbenchmarking study
  - Reciprocal square root (in double-precision)
    - ~14 cycle latency, or equivalently ~14 *independent* instructions
    - It takes 14 + 7 instructions to execute 11 FLOPs
- U-list ***expected*** computational throughput

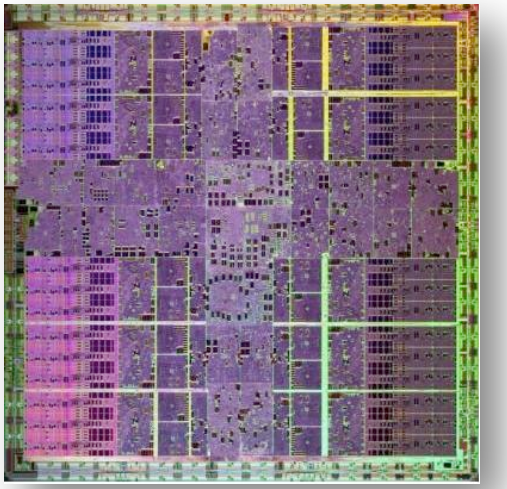
$$C_{peak,gpu} = \frac{11\ FLOPs}{21\ instructions} \times freq \times proc$$

Overview

- Algorithmic characteristics
- GPU performance model
- **Implementation**
- Hybrid scheduling
- Exascale projections

Platform 1 – Jinx @ GT

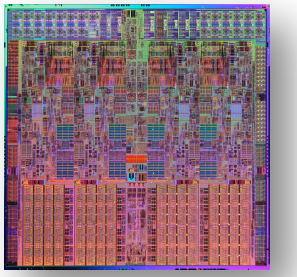


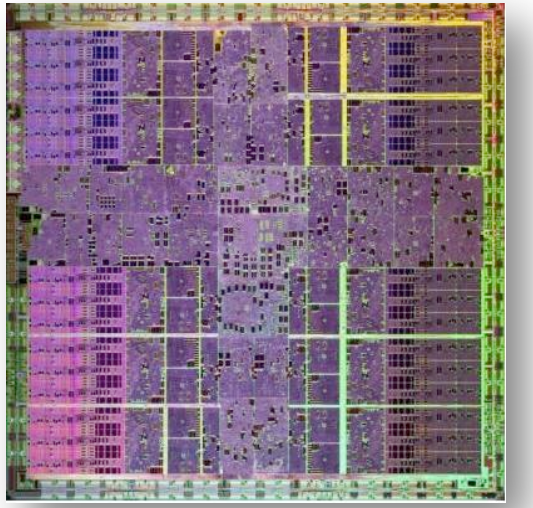


CPU
- Intel Xeon X5650 (Westmere)
- **2** CPUs/node
- **6** cores
- Running @ **2.66** GHz (3.06 GHz TB)
- 147 (SP) / **73** (DP) Gflops/s

GPU
- Tesla M2090 (Fermi)
- **2** GPUs/node
- 512 CUDA cores/**16** SM
- Running @ **1.3** GHz
- 1331 (SP) / **665** (DP) Gflops/s

Platform 2 – Condesa @ HPC Garage





CPU
- Intel Xeon E5-2603 (Sandy Bridge)
- **2** CPUs/node
- **4** cores
- Running @ **1.8** GHz (No TB)
- 58 (SP) / **29** (DP) Gflops/s

GPU
- GTX Titan (Kepler)
- **1** GPU/node
- 2688 CUDA cores/**14** SMX
- Running @ **837** MHz
- 4500 (SP) / **1500** (DP) Gflops/s

GPU Constant Derivation

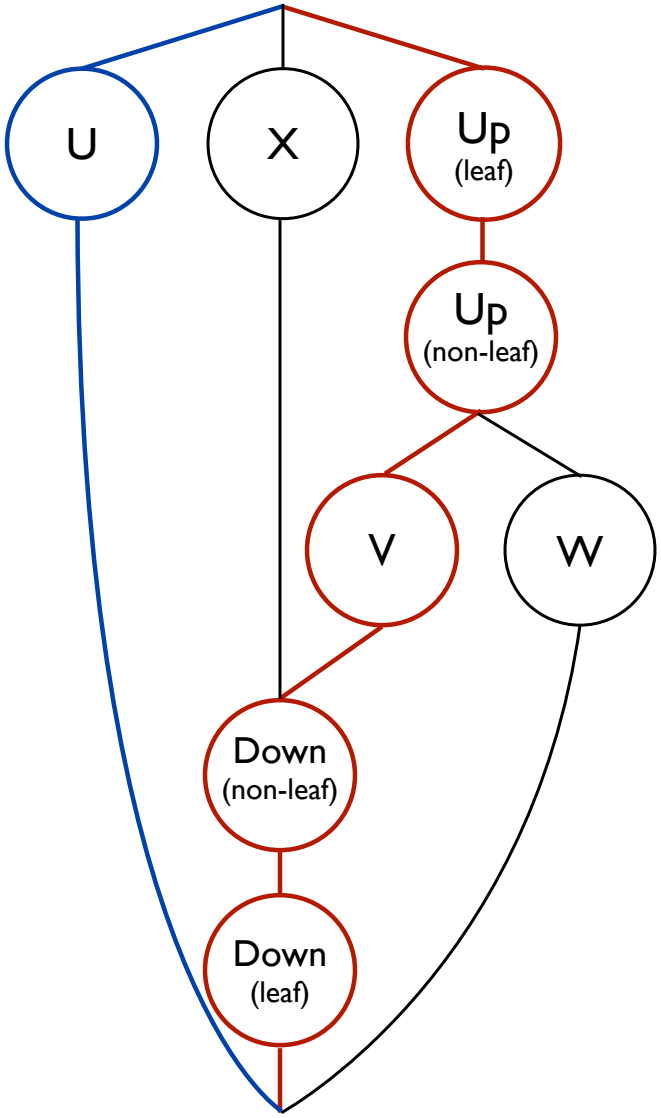
	Tesla M2090	GTX Titan
$C_{peak,gpu}$ (GFLOP/s)	174.3	392.9
$\beta_{mem,gpu}$ (GB/s)	129.4	237.2
$C_{up,gpu}$	2.99	4.16
$C_{u,gpu}$	1.56	2.09
$C_{v,gpu}$	0.95	1.4
$C_{down,gpu}$	7.61	6.83

- We want constants that are close to 1 (better implementation)
- More complicated kernels (upward, downward) are more difficult to model and consequently have higher constants
- Constant values of less than 1 indicates better than modeled performance (e.g., due to better than expected caching)

Overview

- Algorithmic characteristics
- GPU performance model
- Implementation
- **Hybrid scheduling**
- Exascale projections

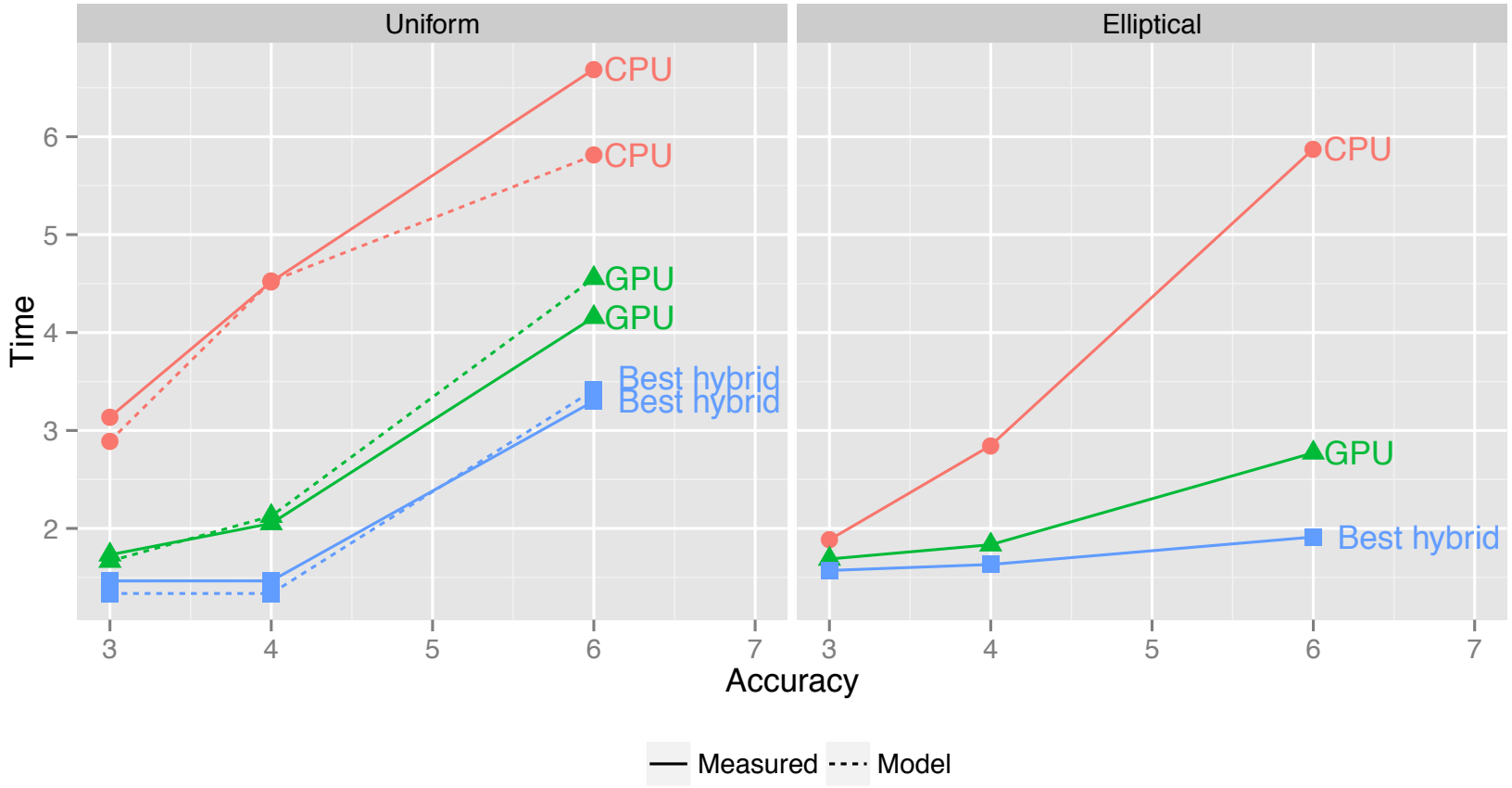
FMM Directed Acyclic Graph



	GPU	CPU
Hybrid1		
Hybrid2		

Hybrid (elliptical distribution)	
CPU	**GPU**
up	u-list
synchronize + memcpy	
v-list	x-list
synchronize + memcpy	
down	w-list
synchronize + memcpy	

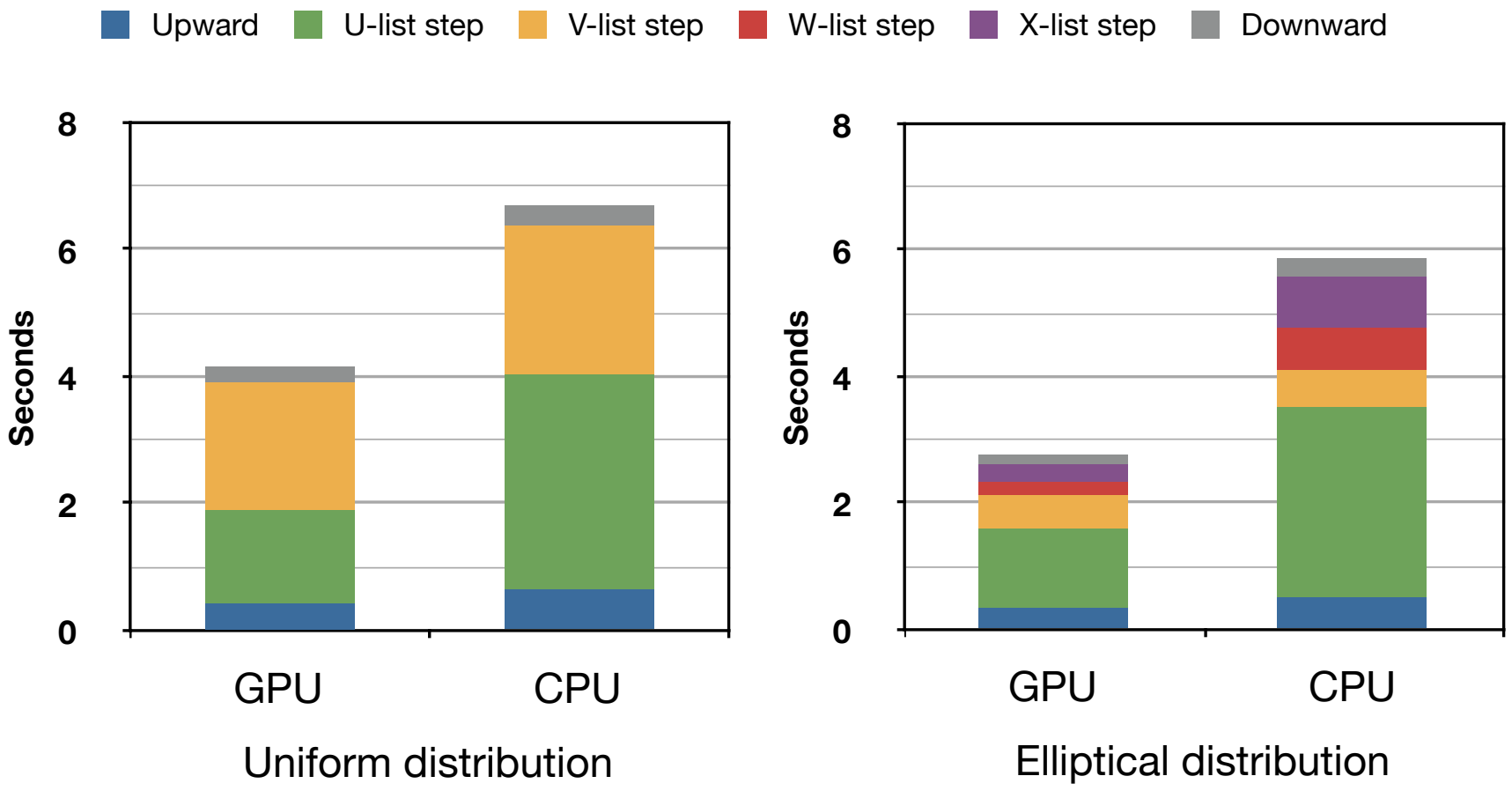
FMM Performance and Model Accuracy



Model Error

	Model median error
Tesla M2090	7.5 %
GTX Titan	6.9 %
X5650	2.2 %
E5-2603	2.0 %
Hybrid1	8.6 %
Hybrid2	7.1 %

FMM Performance Breakdown

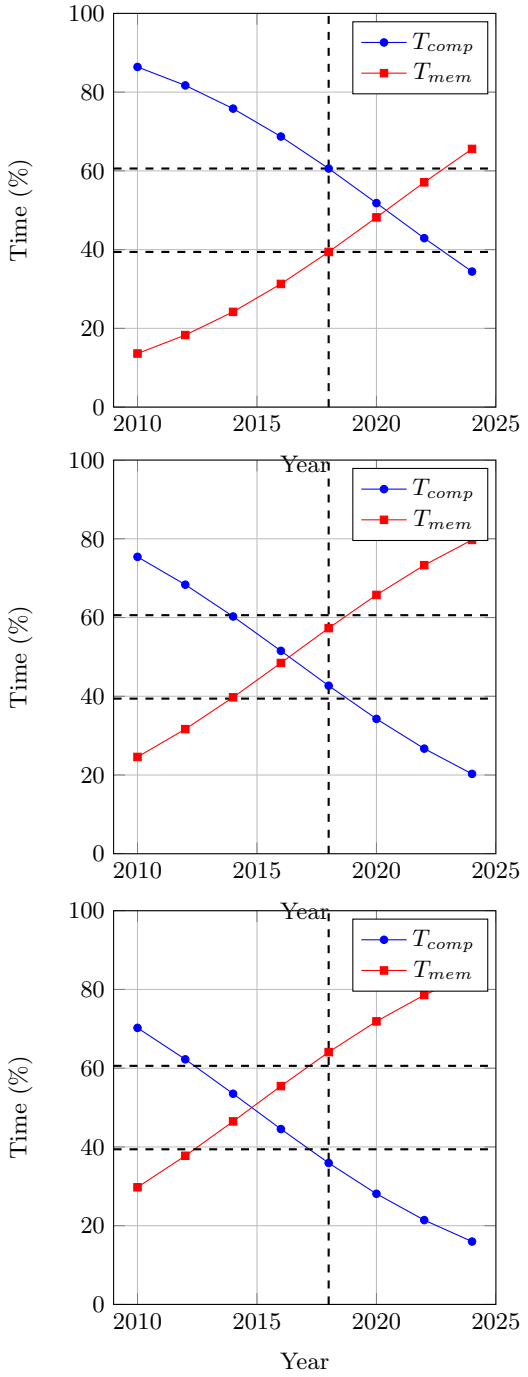


Overview

- Algorithmic characteristics
- GPU performance model
- Implementation
- Hybrid scheduling
- **Exascale projections**

Exascale Projection

- How will FMM scale in the future?
- FMM may become bandwidth-bound
  - No more scaling!
- Better system balance is required
  - Implications for power and energy allocation



Exascale Projection

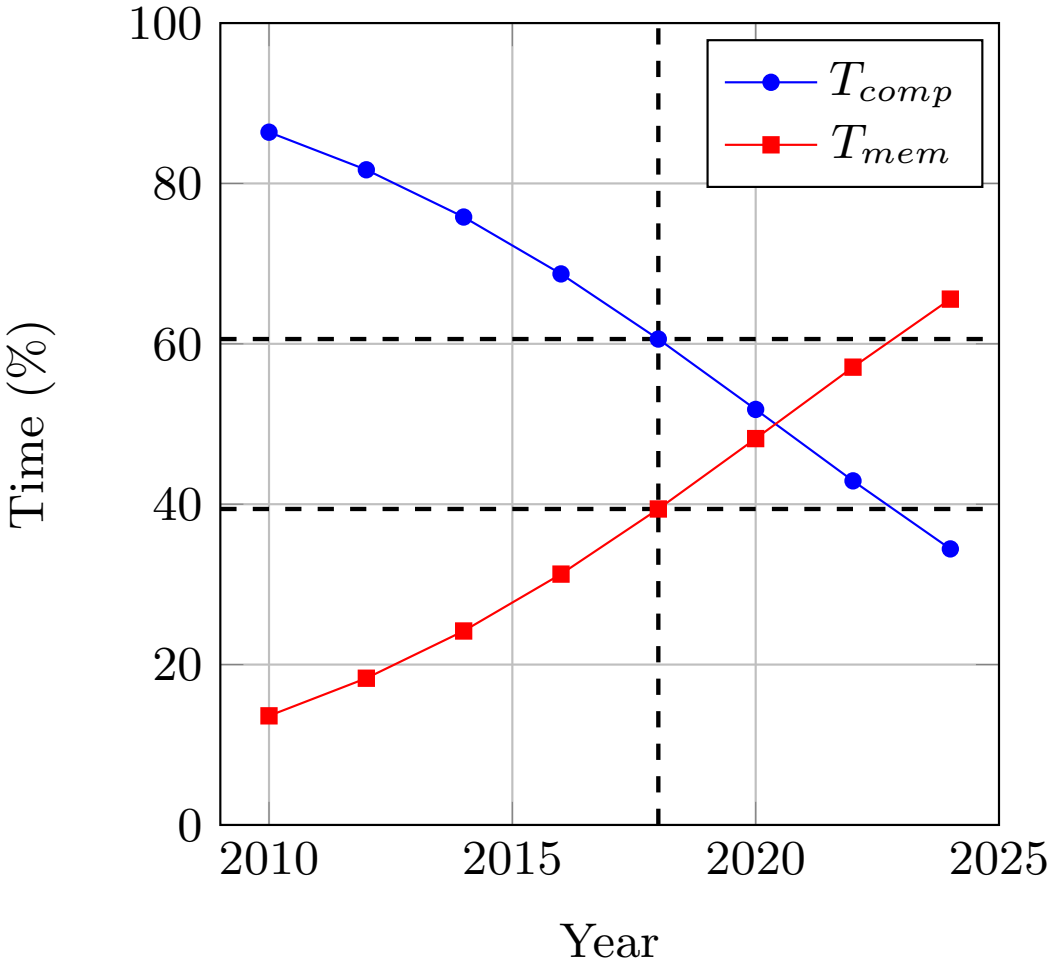
- How will FMM scale in the future?
- FMM may become bandwidth-bound
  - No more scaling!
- Better system balance is required
  - Implications for power and energy allocation

Exascale Projection

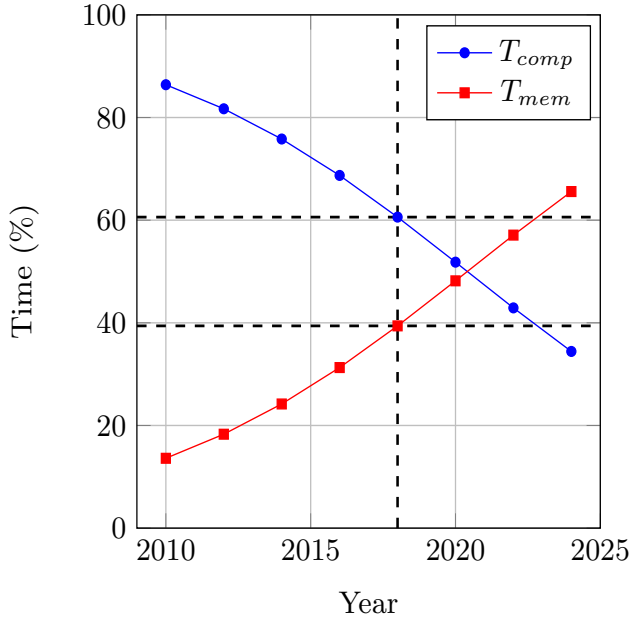
- How will FMM scale in the future?

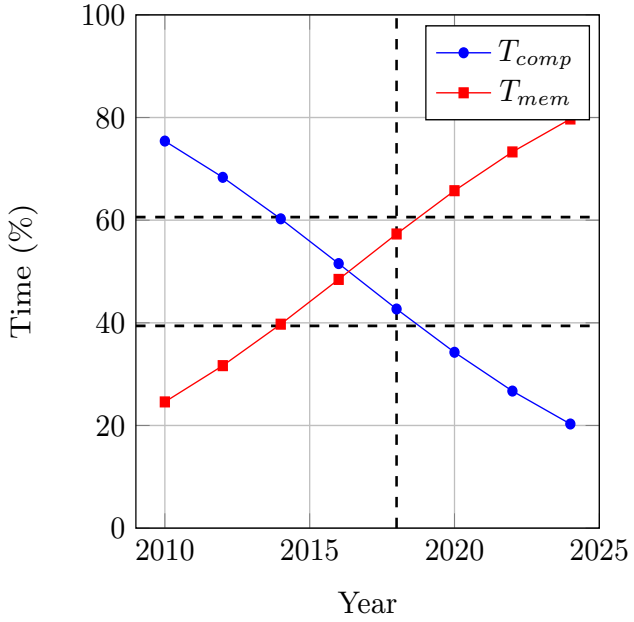
Exascale Projection

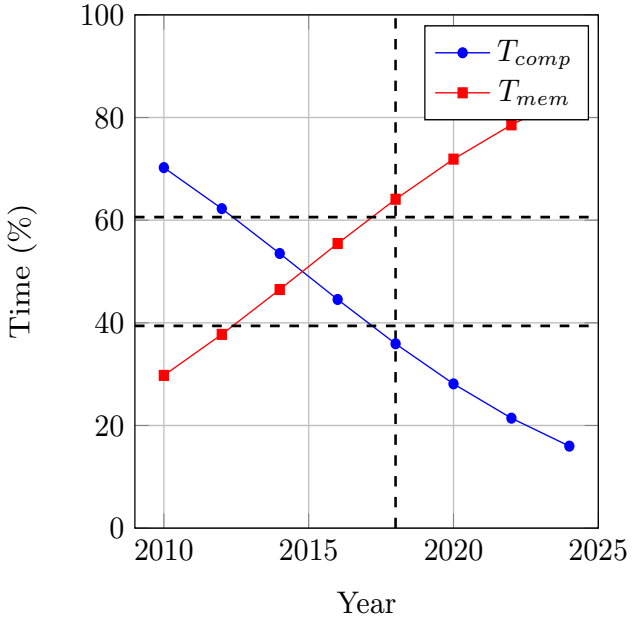
- How will FMM scale in the future?



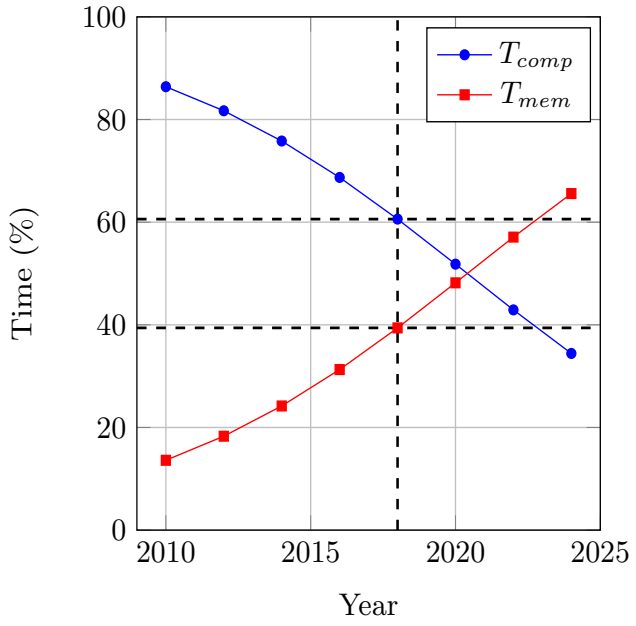
Exascale Projection

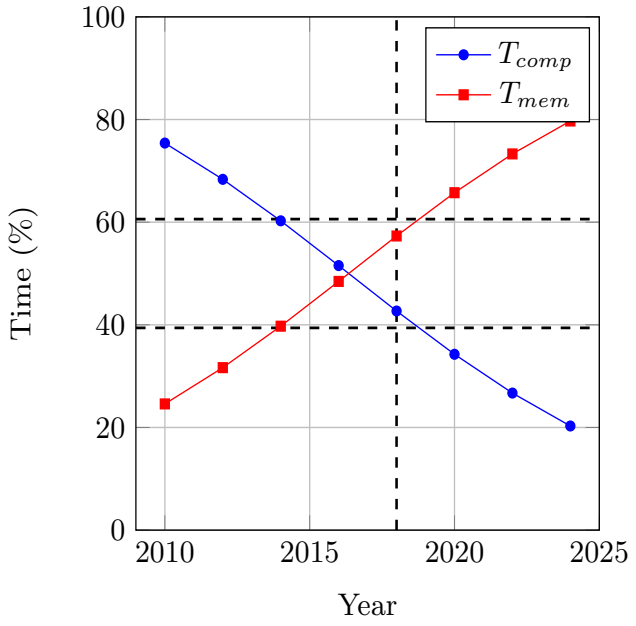


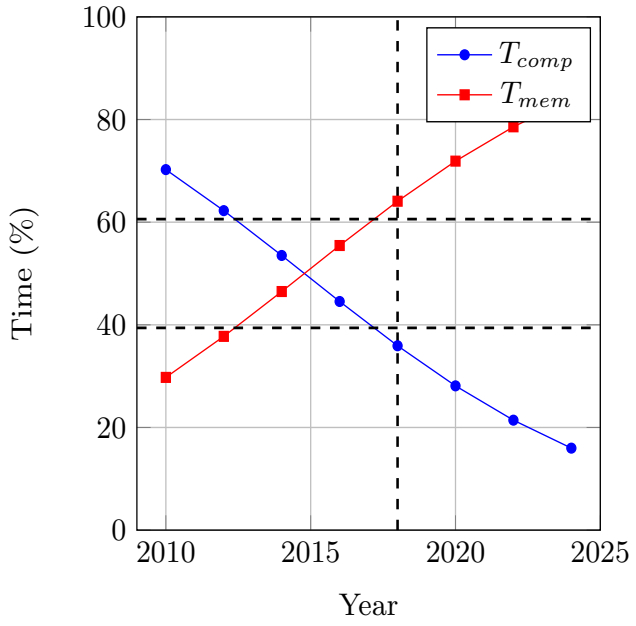




Exascale Projection







- FMM may become bandwidth-bound
  - No more scaling!
- Better system balance is required
  - Implications for power and energy allocation

Conclusions

- Optimized implementation of **FMM** on **CPU** and **GPU**
- An analytical performance model that ***could*** be used to schedule FMM ***efficiently*** on hybrid systems
- ***Exascale*** projection
- There is a need for a similar model for ***elliptical*** distribution of points

Future Work

- Analytical models for W-list and X-list for elliptical distribution
- Power and energy modeling
  - Roofline model of energy
- Support for Xeon Phi accelerator
- FMM for ARM?

Relevant Links

- Source code

  http://j.mp/kifmm--hybrid
- Energy and power
  - A roofline model of energy
  - Algorithmic time, energy, and power on candidate HPC compute building blocks
  - *http://j.mp/energy-roofline--ubenchmarks*