GPU Performance Assessment with HPEC Challenge

Andrew Kerr, Dan Campbell, Mark Richards

andrew.kerr@gtri.gatech.edu, dan.campbell@gtri.gatech.edu, mark.richards@ece.gatech.edu

High Performance Embedded Computing (HPEC) Workshop

September 25, 2008

This work was supported in part by DARPA and AFRL under contracts FA8750-06-1-0012 and FA8650-07-C-7724. The opinions expressed are those of the authors.

Georgia Tech Research Institute

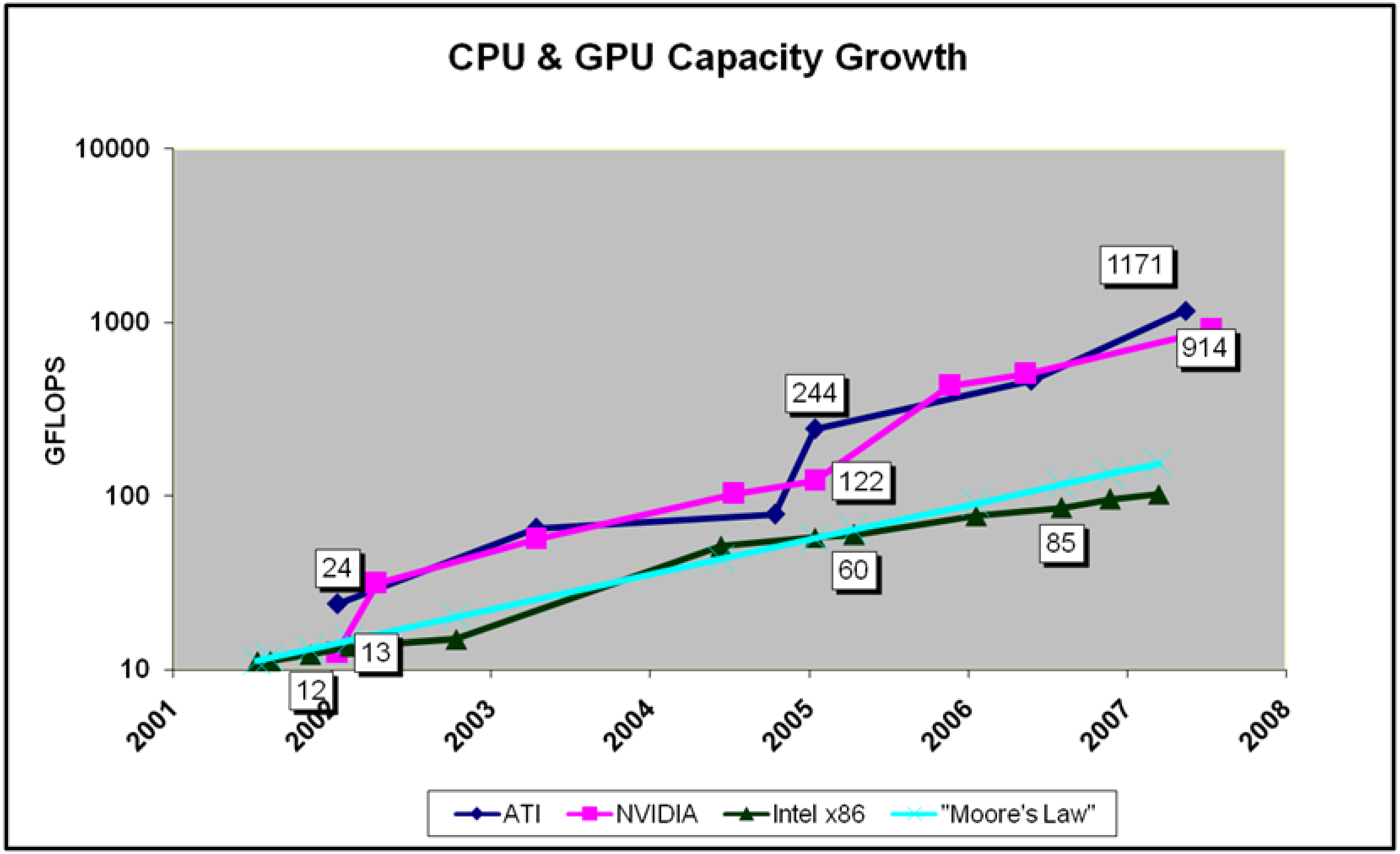
General Purpose GPU Computing

- Modern GPUs have unified shader architecture
  - Highly parallel programmable processing units
  - Flexibility extends GPU beyond rasterized 3D graphics
  - New vendor focus on high-performance computing:
    - NVIDIA's CUDA, ATI's CTM
- High theoretical performance (500 GFLOPs or more)
  - Leverages volume & competition in entertainment industry
    - Worldwide GPUs: $5B, 10M units per year
    - U.S. Video Games: $7.5B, 250M units 2004
    - Holds down unit-price, drives advancement
- Outstripping CPU capacity, and growing more quickly

General Purpose GPU Computing

- Modern GPUs have unified shader architecture
  - Highly parallel programmable processing units
  - Flexibility extends GPU beyond rasterized 3D graphics
  - New vendor focus on high-performance computing:
    - NVIDIA's CUDA, ATI's CTM
- High theoretical performance (500 GFLOPs or more)
  - Leverages volume & competition in entertainment industry
    - Worldwide GPUs: $5B, 10M units per year
    - U.S. Video Games: $7.5B, 250M units 2004
    - Holds down unit-price, drives advancement
- Outstripping CPU capacity, and growing more quickly

GPU Performance Trends: Unified Shaders



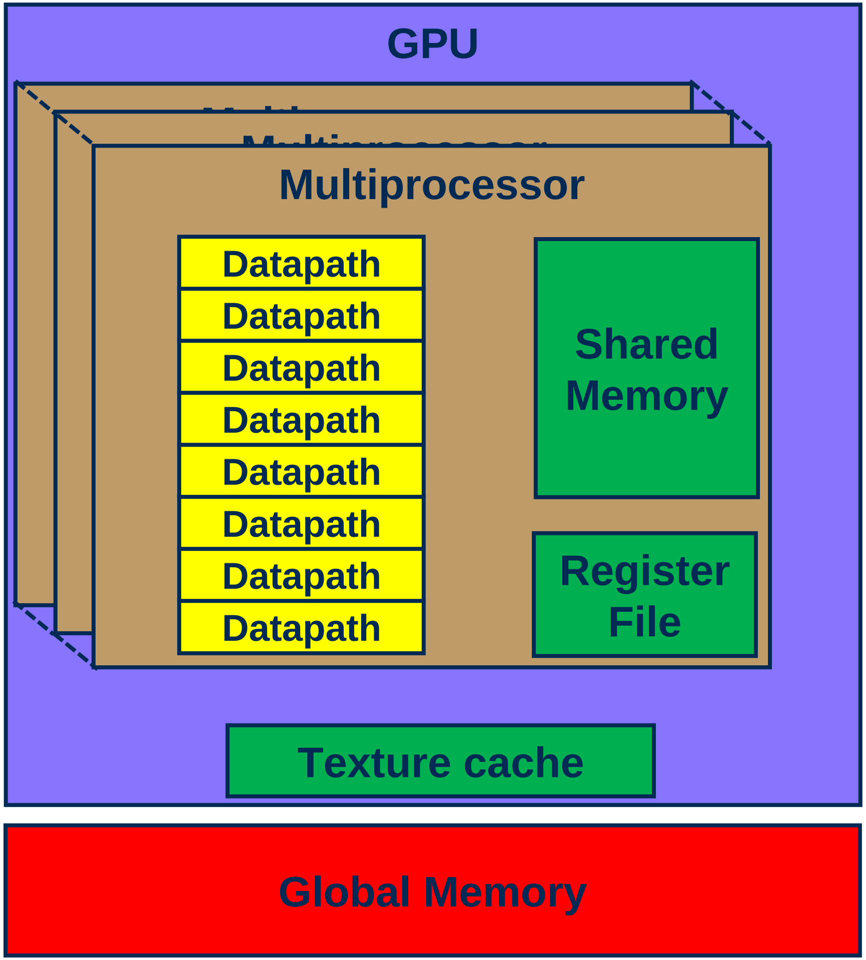
HPEC Challenge Benchmarks

- HPEC Challenge
  - How will candidate architecture perform in real application?
  - Nine kernel benchmarks and one application benchmark.
  - Seven attempted:
    - Corner turn, Time-domain FIR, Frequency-domain FIR, Constant False Alarm Rate, Pattern Matching, Graph Optimization via Genetic Algorithm, QR Factorization
  - http://www.ll.mit.edu/HPECchallenge/
- Experimental System
  - NVIDIA GeForce 8800 GTX
  - Intel Core2 Q6600 2.4 GHz
  - Windows XP Professional, Visual C++ 2005 host C++ compiler
  - NVIDIA CUDA 1.1

CUDA Programming Model

- Compute Unified Device Architecture (CUDA)
  - C-like programming language for executing kernels on GPU without casting as 3D graphics operation
  - Keywords denote memory placement, grid environment, thread index
  - Built-in functions for synchronization, fast math, cycle counts
  - Runtime API for memory management, launching kernels, synchronizing host

GPU Architecture (G80)

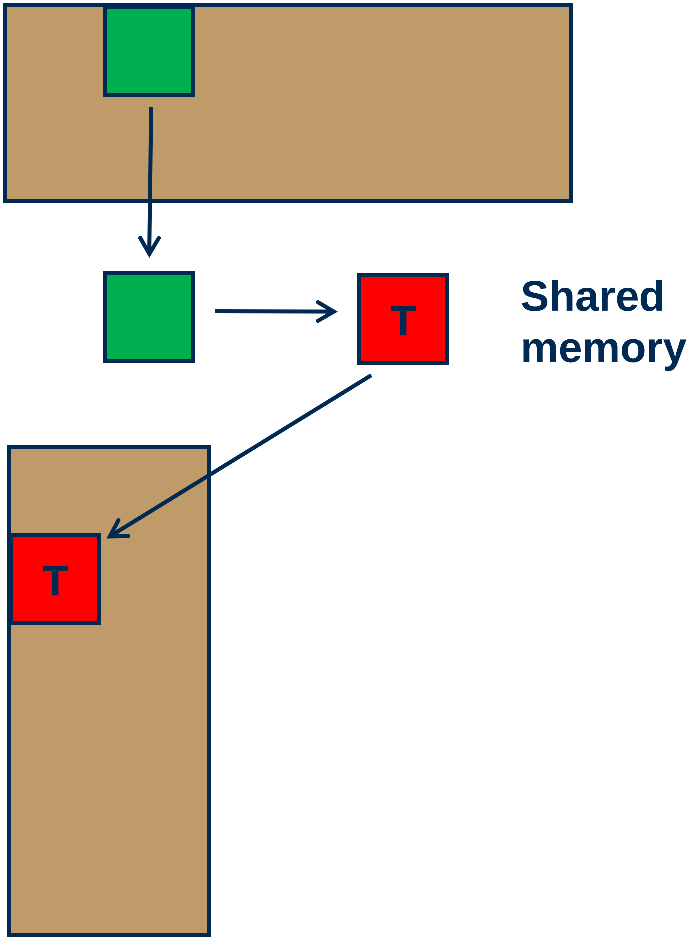


- Programmable units arranged as 16 “multiprocessors”
- For multiprocessor:
  - eight datapaths
    - Single-precision and int
  - 16 kB scratchpad
  - 8,192 word register file
  - Scheduler
- 384-bit memory bus handles requests from all threads
- 1.3 GHz core clock, 575 MHz memory

CUDA Grids, Threads, and Blocks

- Problem logically decomposed into “blocks”
  - Scheduler maps blocks to available multiprocessors for concurrent execution
  - Execution order not defined, synchronization not defined
- Blocks partitioned into threads
  - Threads meant to be executed in SIMD manner on multiprocessor
  - More threads than datapaths
    - set of active threads known as “warp”
    - scheduler devotes two cycles per “half warp”
    - floating-point MADD has latency of 4 cycles
  - When threads stall due to memory accesses, another warp is activated

Corner Turn



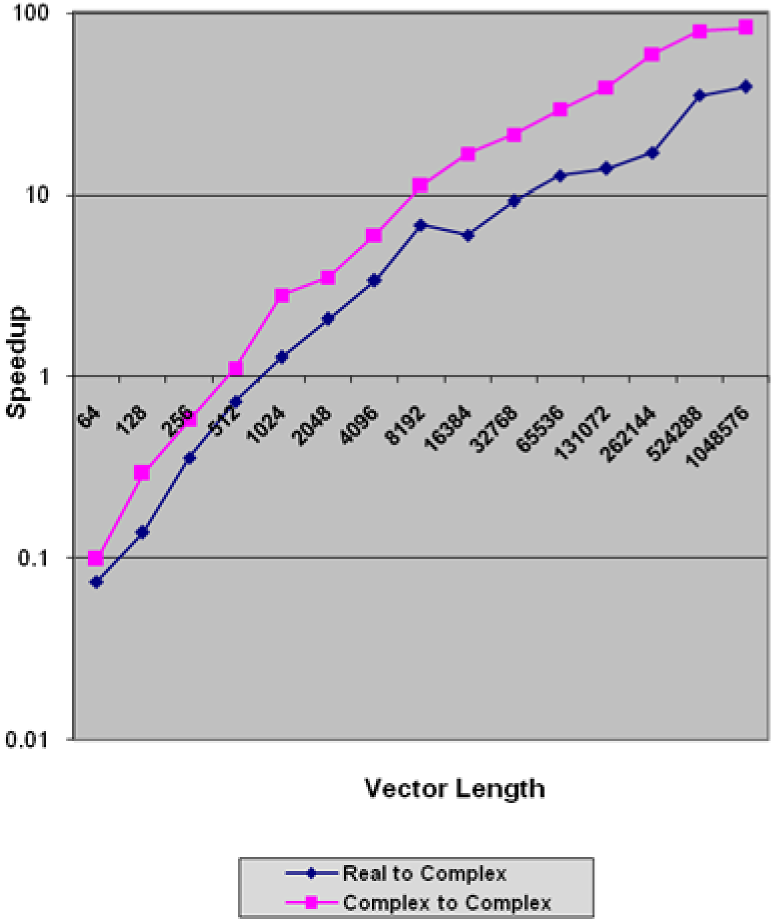
- **Benchmark:**
  - **Compute real-valued transpose out of place**
- **Strategies:**
  - **coalesce reads and writes of adjacent threads to adjacent global memory locations**
  - **transpose in shared memory**
  - **minimize overhead of address computation**
- **Good match for GPU:**
  - **Set 1: 0.30 ms – 8.32x speedup**
  - **Set 2: 4.60 ms – 11.4x speedup**

Time-Domain FIR

```
Yblock[thread] =
   hblock [0] * xblock [ thread ] +
   hblock [1] * xblock [ thread – 1] +
   hblock [2] * xblock [ thread – 2] +
   .
   .
   .
```

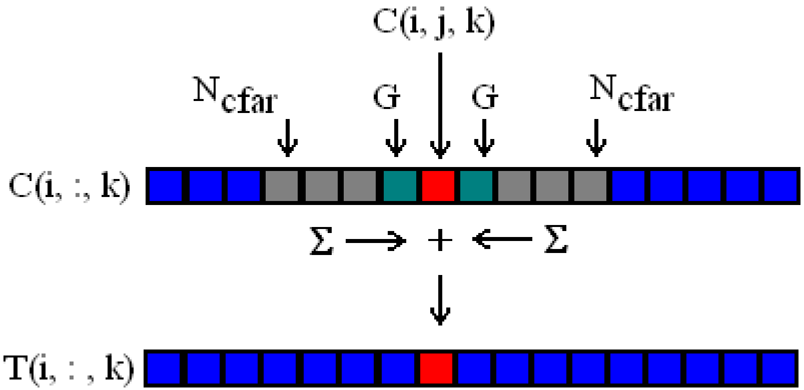
- **Benchmark:**
  - **convolve a set of FIR filters with a set of input vectors**
- **Strategies:**
  - **filter coefficients fit in shared memory**
  - **map each filter to a block**
  - **large number of threads per block overlap computation with streaming of input vector**
  - **loop unrolling to improve utilization**
- **Good match for GPU**
  - **Set 1: 2.54 ms - 151x speedup**
  - **Set 2: 0.09 ms – 22.2x speedup**

Frequency-Domain FIR



- **Benchmark:**
  - **fast convolution of set of FIR filters in the frequency domain**
- **Strategies:**
  - **NVIDIA's CUFFT library provides Fast Fourier Transform**
  - **kernel performs complex element-wise multiplication**
- **Good match for GPU**
  - **FFT speedup greater for large input vectors**
  - **Set 1: 3.25 ms – 19.7x speedup**
  - **Set 2: 0.26 ms – 11.5x speedup**

Constant False Alarm Rate Detection



$C(i, j, k) = T(i, j, k)^{-1} \mid C(i, j, k) \mid^2$

- **Benchmark:**
  - **Beams x Range Gates x Doppler Bins**
  - **Normalize each cell by surrounding noise estimate**
- **Strategies:**
  - **map each (beam, Doppler bin) to a block**
  - **Stream range gates and compute noise estimate**
- **Good match for GPU**
  - **Set 1: 0.29 ms – 2.3x speedup**
  - **Set 2: 3.5 ms – 166x speedup**
  - **Set 3: 3.4 ms – 46.8x speedup**
  - **Set 4: 2.7 ms – 25.6x speedup**

Pattern Matching

```
Pattern Matching {
  for each of K patterns {
    for each of Sr shift values {
      find MSE of input with
        shifted pattern;
    }
    select shift with least MSE;

    for each of Sm magnitudes {
      find MSE of input with
      scaled pattern;
    }
    choose gain with least MSE;
  }
  choose gain, shift, pattern with
    least MSE;
}
```

- Set 1: 0.24 ms – 12.7x speedup
- Set 2: 1.65 ms – 23.1x speedup

- **Benchmark:**
  - Compute mean squared error (MSE) of input vector with template library
  - Determine optimal shift and scale for minimum MSE
- **Strategies:**
  - Process each pattern in parallel (one per block)
  - Each thread computes one shift then one gain
- **Good match for GPU**

Graph **Optimization** via Genetic Algorithms

```
Genetic Algorithm {

  Initialization;
  Evaluation;

  while !finished {

    Selection;
    Reproduction;
    Crossover;
    Mutation;
    Evaluation;

  }
}
```

- **Set 1: 0.5 ms – 15.6x speedup**
- **Set 2: 11.7 ms – 33.3x speedup**
- **Set 3: 1.0 ms – 21.9x speedup**
- **Set 4: 4.1 ms – 23.7x speedup**

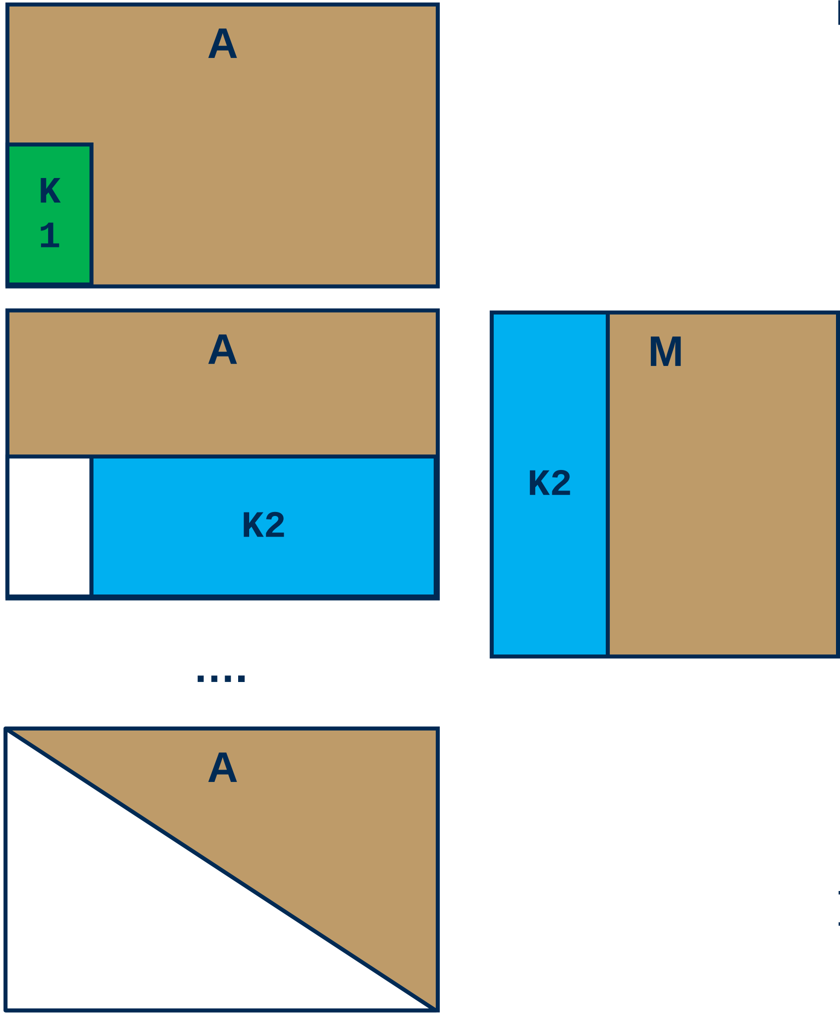
- **Benchmark:**
  - **use a genetic algorithm to search a problem space**
  - **Roulette wheel selection**
  - **Evaluation based on lookup table**
  - **Elite chromosomes immune to mutation**
- **Strategies**
  - **batch kernel calls to perform iteration**
  - **Implement parallel RNG**
  - **Selection and reproduction is a gather operation**
  - **Crossover, mutation are parallel**
  - **Evaluation is parallel**

QR Factorization: Fast Givens

```
M = eye(m, m);
d = ones(m);

for j = 1 : n {

  for i = m: -1: j+1 {

    [α, β, τ] = fast.givens(
      A(i-1:i, j:n), d(i-1:i));

    A(i-1:i, j:n) =
      G(α, β, τ)ᵀ A(i-1:i, j:n);

    M(j:m, i-1:i) =
      M(j:m, i-1:i) G(α, β, τ);

  }
}
D = diag(d);
Q = M D^(-1/2);
R = D^(1/2) A;
```

- **Benchmark:**
  - $A = QR$, $Q^H Q = I$, R **upper triangular**
  - **Fast Givens:**
    - **few square roots**
    - **fine-grain parallelization**
    - **streaming implementation requires different programs to run on several nodes**
  - **GPU Characteristics:**
    - **Fine-grain parallelization among threads of one block**
    - **SIMD execution among threads**
    - **Square roots inexpensive**
    - **Shared memory capacity limited**

Fast Givens: GPU Strategy




```
Fast Givens {
  do {
    // kernel 1 – one block
    load several columns of A;
    move up columns rotating A
         with threads staggered;
    write rotations to global memory;

    // kernel 2 – sixteen blocks
    load rotations;
    load columns from remaining
      submatrix of A;
    apply rotations to A in order;

    load submatrix of M;
    apply rotations to M in order;

    move active window right;

  } until all columns zeroed;
}
```

QR on GPU Conclusions

- **Fast Givens not greatest match**
  - **Parallelism well-suited to synchronous data flow architecture**
  - **Avoids calculations that are fast on GPU**
  - $2n^2(m-n/3)$ **flops**
- **Results:**
  - **Set 1: 20. ms – 4.6x speedup**
  - **Set 2: 4.5 ms – 1.5x speedup**
  - **Set 3: 1.8 ms – 5.6x speedup**
- **Other QR methods:**
  - **Householder reflections:**
    - **compute** v **such that** $(I - \beta v v^T)x = ||x|| e_1$
    - $A - v(\beta A^T v)^T \rightarrow A$
    - ***serial, parallel, serial, parallel, …*** **fast with batched calls**
    - $2n^2(m-n/3)$ **flops**

GPU Limitations

- GPU Memory Architecture
  - G80 lacks globally visible, writable cache
  - Global memory has high latency
  - Shared memory fast, limited in capacity

- Fine-grain Parallelism
  - Threads share data directly with fast synchronization
  - Blocks share via global memory, multiple kernel invocations
  - Atomic memory operations possible with newer GPUs

- Kernel latency
  - CPU ⬄ GPU communications limited by PCI-Express Bus
    - Newer GPUs permit DMA while kernels execute (G92)
  - Delay incurred when calling kernel, copying results
  - Tolerable for large data sizes and batched calls

Conclusions

- GPU speedup possible for most classes of problems
  - Memory hierarchy and threading model drive implementation
  - High memory bandwidth, high parallelism good implementation of streaming architecture
  - Cleverness required for fast implementations
  - High performance
- Fine-grain parallelism not great match
  - No formal synchronization across blocks
- Benchmarks should grant flexibility to implementation
  - don’t require obscure algorithms to solve common problems
  - don’t define metrics biased away from coprocessors without necessity

References

- **HPEC Challenge Benchmarks**
  - **http://www.ll.mit.edu/HPECchallenge/**

- **Golub and Van Loan. *Matrix Computations*. Johns Hopkins University Press, 3rd edition. 1996.**

- ***NVIDIA CUDA Programming Guide 1.1***
  - ***http://www.nvidia.com/object/cuda_develop.html***

Georgia Tech Research Institute

Questions

Questions?

Georgia Tech Research Institute