Gaussian Elimination Based Algorithms on the GPU

Aydin Buluc
John R. Gilbert
Ceren Budak
University of California, Santa Barbara

PMAA Workshop 2008
June 21, 2008

Support: DOE Office of Science, MIT Lincoln Labs
GPUs are powerful!

Intel 80-core chip

> 1 TFLOPS

For now, this is currently a prototype
(in market by 2020 :-)

Two Nvidia 8800 GPUs

> 1 TFLOPS

You can go and buy these two for about $1000 now
But … GPGPU is hard to implement

- Performance is fragile:
  - E.g. add 1 line of code and you’re %40 slower

- Programming is counter-intuitive.
  - The code you’d write looks great with stride-1 access per thread.
  - But it’s terrible on CUDA!
  - If that was column-major storage, then it would be great on CUDA (Memory coalescing)

- Divergence should be avoided
  - Single instruction decoder for 8 FPUs
Lots of pitfalls for programmers

- Novice (anybody that hasn’t spend 5000 hours 😊) programmers, may fall to pitfalls.
- Extremely easy to underutilize the device.
- Example:
  - GEMM on GPU
    - Doing it wrong costs you 100x
    - Doing it slightly wrong costs you 10x
  - GEMM on CPU
    - Doing it wrong costs you 8x
    - Doing it slightly wrong costs you 2x

Primitives are more crucial on the GPU
Matrix Multiplication (GEMM) is fast

- Fatahalian et.al.[FSH04] published in 2004 about the inefficiency of GPU algorithms for GEMM.
- Things have changed since then:
  - We now have impressive bandwidth. 
    \textit{More than 100GB/sec on a single GPU.}
  - Non-bandwidth bound methods have been discovered recently.
    \textit{Volkov & Demmel} [VD08].
- Why not take advantage of such a primitive?
Gaussian Elimination Paradigm [CR07]

- Certain types of algorithms, with triple nested loops, have very similar data access patterns.
  - LU Decomposition without pivoting
  - All-Pairs Shortest-Paths (APSP)
  - Transitive Closure

- LU Decomposition (even with pivoting), achieved more than 300 GFlops using two NVIDIA 8800 GTX [VD08]

- What about the other two? As of now:
  - Numerical algorithms on GPU are for academic purposes only (double-precision non-existent)
  - No such problem for graph algorithms
Matrices over Semirings

- Matrix multiplication $C = AB$ (or matrix/vector):

$$C_{i,j} = A_{i,1} \times B_{1,j} + A_{i,2} \times B_{2,j} + \cdots + A_{i,n} \times B_{n,j}$$

- Replace scalar operations $\times$ and $+$ by

  $\otimes$: associative, distributes over $\oplus$, identity 1

  $\oplus$: associative, commutative, identity 0 annihilates under $\otimes$

- Then $C_{i,j} = A_{i,1} \otimes B_{1,j} \oplus A_{i,2} \otimes B_{2,j} \oplus \cdots \oplus A_{i,n} \otimes B_{n,j}$

- Examples: $(\times, +)$; (and, or); $(+, \min)$; . . .

- Same data reference pattern and control flow
Algebraic formulation:

for $k=1:n$
  
  $D \leftarrow D \oplus [D(:,k) \otimes D(k,:)]$

$\oplus$: \text{min}

$\otimes$: outer product using $+$

$k = 1$ case
Recursive APSP

vertices V1

vertices V2

A = \text{apsp}(A); \quad // A = A^*
B = AB;
C = CA;
D = D + CB;

D = \text{apsp}(D); \quad // D = D^*
B = BD;
C = DC;
A = A + BC;

A = A^* + A^*B(D + CA^*B)^*CA^*
Execution of Recursive APSP
More on recursive APSP

- A similar recursive implementation to optimize graph algorithms on the CPU → 2x-3x speedup [PPP04]
- Implementation of the Floyd-Warshall algorithm on NVIDIA 8800 GTX → 2x-3x speedup [HN07]
- Our formulation is based on R-Kleene [DN04]
  - B=AB or B=BA can be done in-place (bandwidth-friendly) as long as A=A*.
  - Input and output matrices in GEMM can be the same because we are on (+,min) semiring
  - If the algorithm prematurely overrides its input, correctness is still preserved.
Our first attempt

- Intel Core 2 Duo 1.83 Ghz
  - 67x speedup
- NVIDIA 8800 Ultra
But we could do better…

Using a variant of new GEMM (Volkov & Demmel)
A closer look on optimization

![Graph showing the comparison between Optimized Recursive (GPU) and Basic Recursive (GPU) in terms of time (in milliseconds) vs. dimension.](image)
How does it compare?

Even for extremely sparse graphs with average degree 6:

Ours is more than 3x faster than running Dijkstra from multiple source vertices
A possible GPU Moral

- Recursion on the host code is not decremental to the performance.

  Recursion stack is here

  Actual kernel executes here

  Kernel launches

- Recursive LU [T97], and recursive APSP [PPP04] are shown to have better locality of reference than their iterative counterparts.
Conclusions

• Use optimized primitives as much as possible
  – Parallel-Prefix Sums
  – Matrix-matrix multiplication
  – ?
• Straightforward porting of applications might not work.
  – Look for non-traditional, alternative algorithms.
  – Bandwidth is *usually* the bottleneck, choose BW-friendly algorithms
• Divide & conquer paradigm using recursion maps very well to GPU hardware
References


