

UCSB



Gaussian Elimination Based Algorithms on the GPU

Aydin Buluc

John R. Gilbert

Ceren Budak

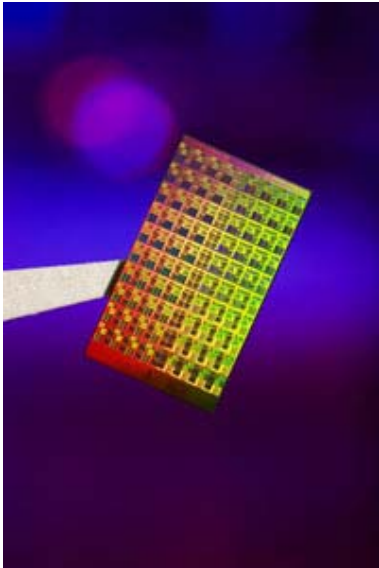
University of California, Santa Barbara

PMAA Workshop 2008

June 21, 2008

Support: DOE Office of Science, MIT Lincoln Labs

GPUs are powerful !



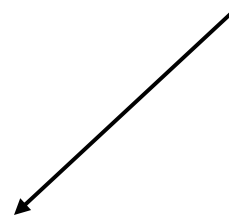
Intel 80-core
chip
> 1 TFLOPS

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



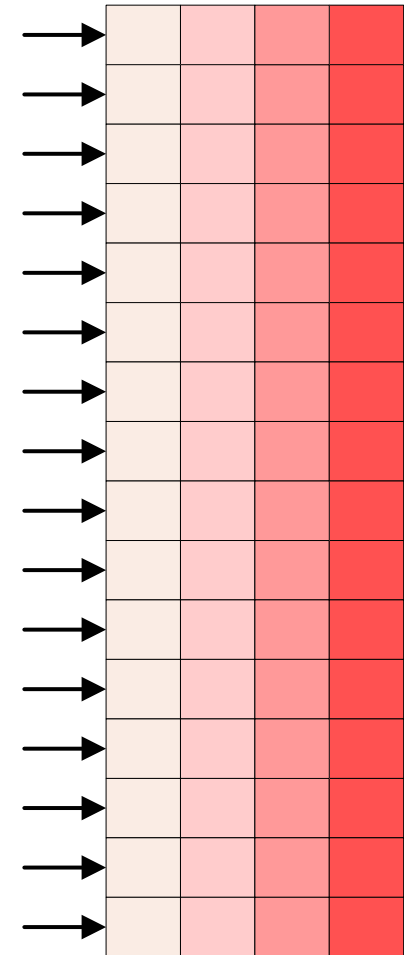
Two Nvidia
8800 GPUs
> 1 TFLOPS

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.
More than 100GB/sec on a single GPU.
 - Non-bandwidth bound methods have been discovered recently.
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 $\mathbf{C} = \mathbf{AB}$ (or matrix/vector):

$$\mathbf{C}_{i,j} = \mathbf{A}_{i,1} \times \mathbf{B}_{1,j} + \mathbf{A}_{i,2} \times \mathbf{B}_{2,j} + \dots + \mathbf{A}_{i,n} \times \mathbf{B}_{n,j}$$

- Replace scalar operations \times and $+$ by

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

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

- Then $\mathbf{C}_{i,j} = \mathbf{A}_{i,1} \otimes \mathbf{B}_{1,j} \oplus \mathbf{A}_{i,2} \otimes \mathbf{B}_{2,j} \oplus \dots \oplus \mathbf{A}_{i,n} \otimes \mathbf{B}_{n,j}$

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

- Same data reference pattern and control flow

Iterative APSP (Floyd-Warshall)

```
for k=1:n           // the induction sequence
  for i = 1:n
    for j = 1:n
      if(w(i→k)+w(k→j) < w(i→j))
        w(i→j):= w(i→k) + w(k→j)
```

Algebraic formulation:

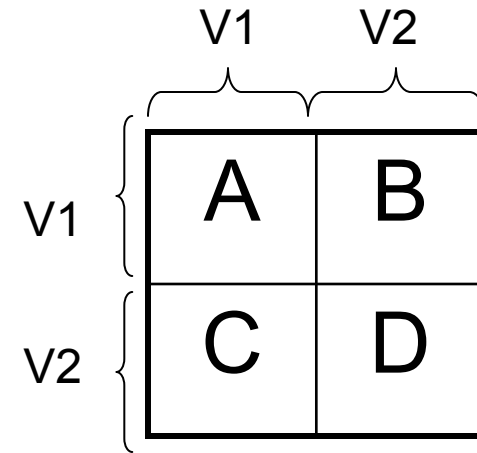
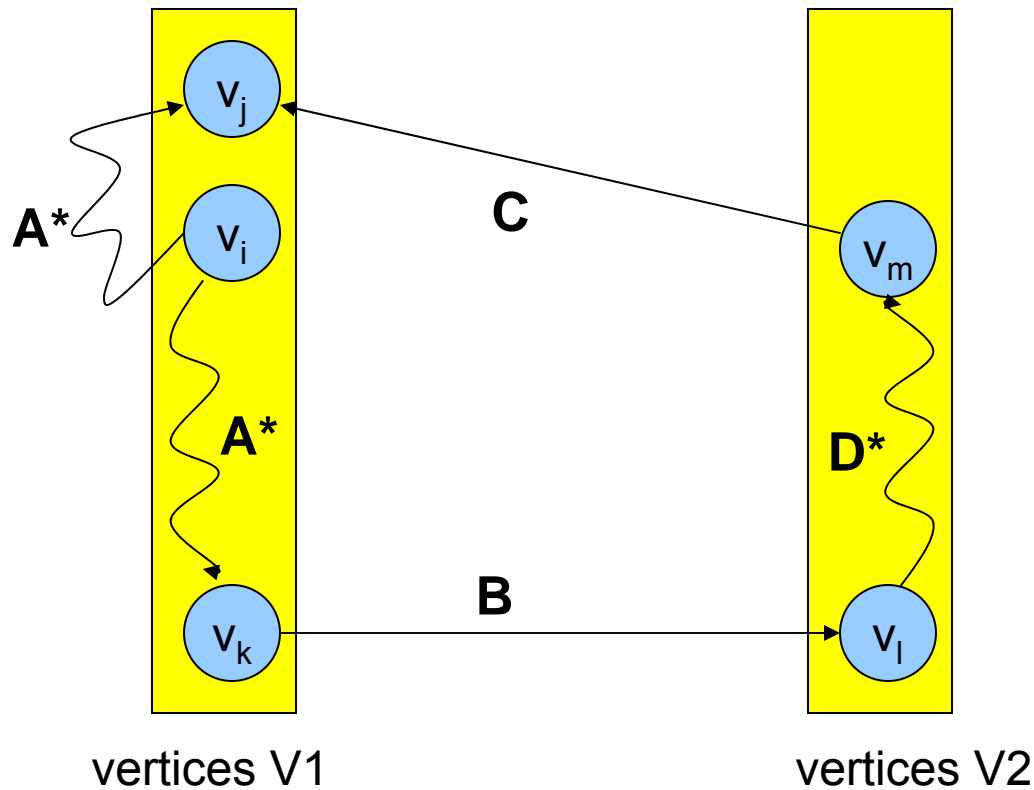
```
for k=1:n
   $D \leftarrow D \oplus [D(:,k) \otimes D(k,:)]$ 
```

\oplus : min

\otimes : outer product using +

k = 1 case

Recursive APSP



$A = \text{apsp}(A); \quad // A = A^*$

$B = AB;$

$C = CA;$

$D = D + CB;$

$D = \text{apsp}(D); \quad // D = D^*$

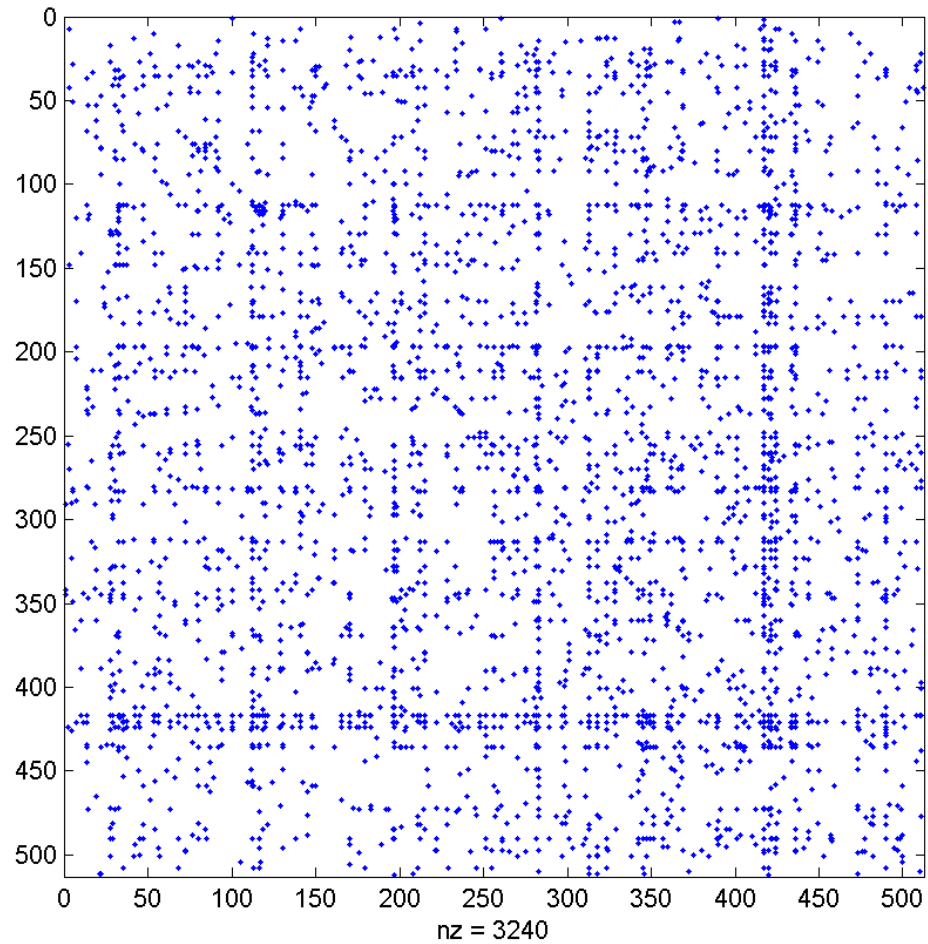
$B = BD;$

$C = DC;$

$A = A + BC;$

$A = A^* + A^*B(D + CA^*B)^*CA^*$

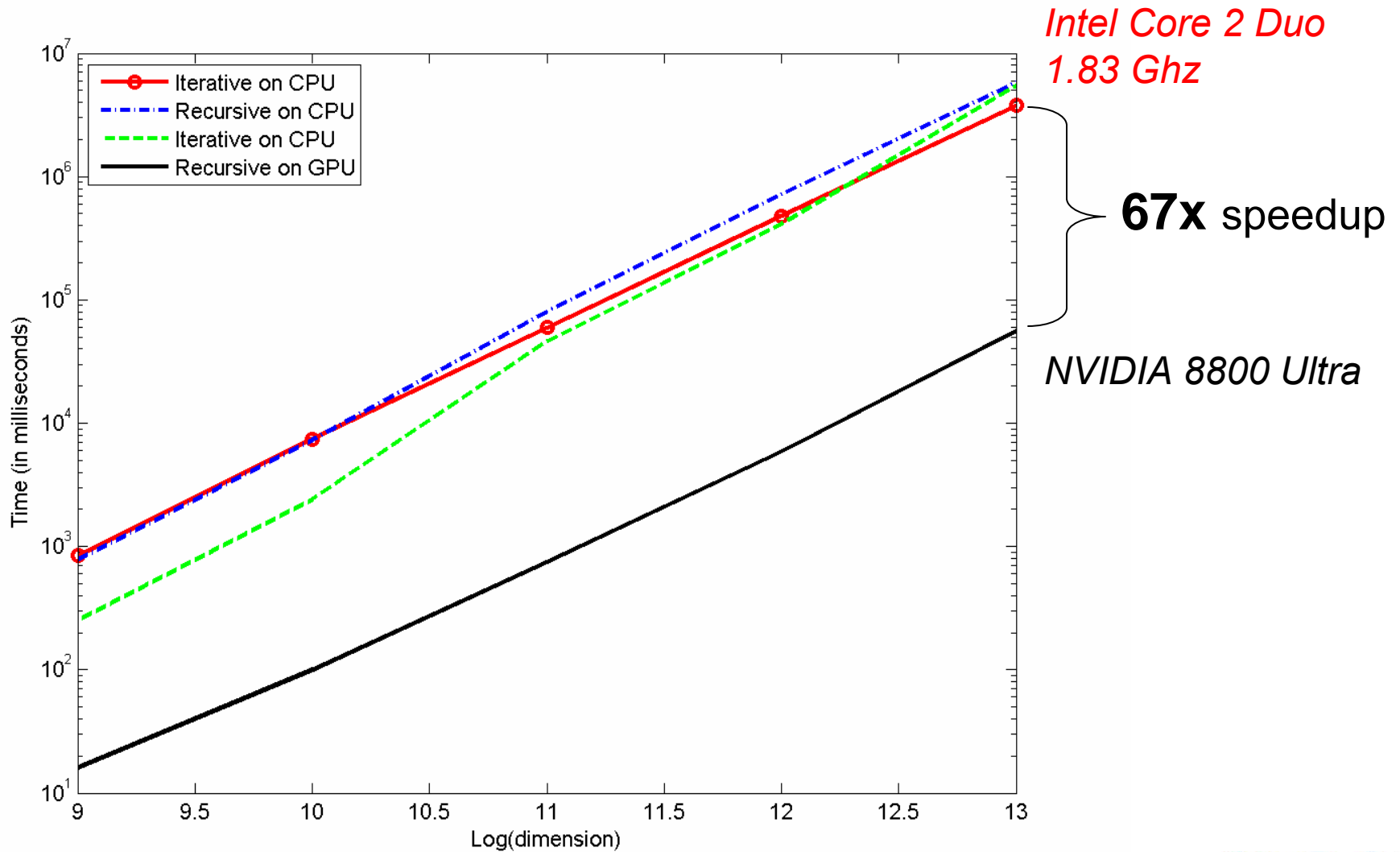
Execution of Recursive APSP



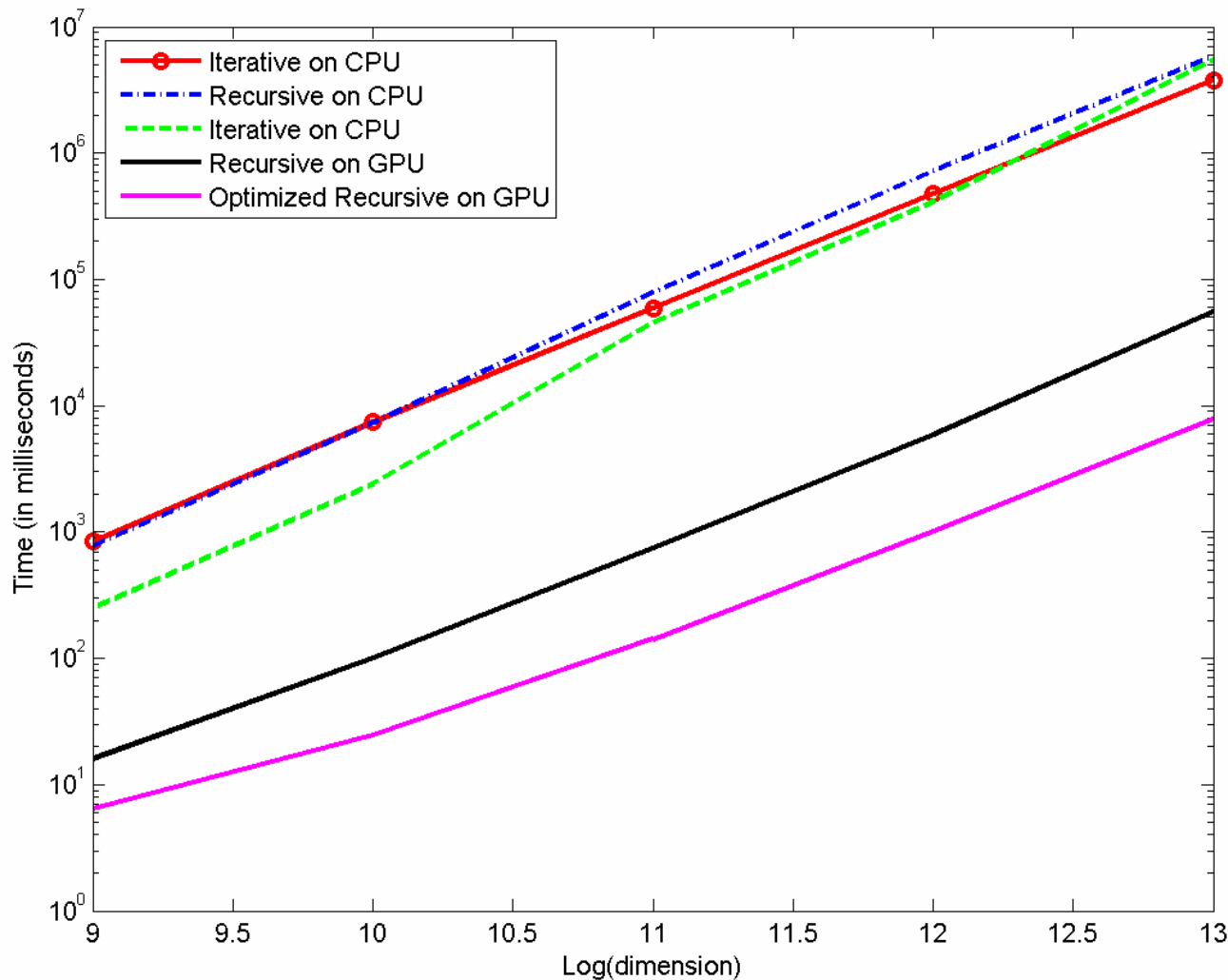
More on recursive APSP

- A similar recursive implementation to optimize graph algorithms on the CPU \rightarrow 2x-3x speedup [PPP04]
- Implementation of the Floyd-Warshall algorithm on NVIDIA 8800 GTX \rightarrow 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



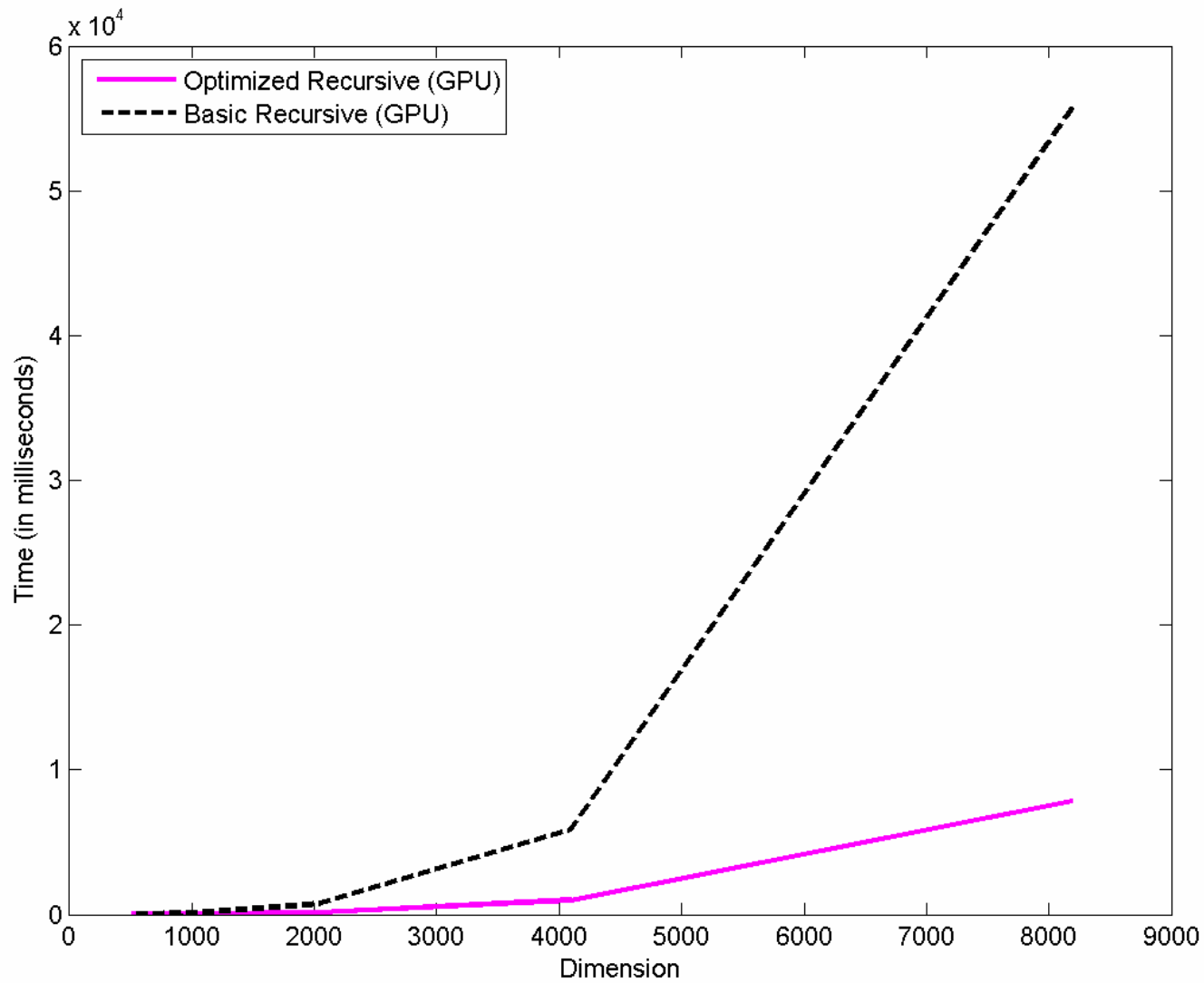
But we could do better...



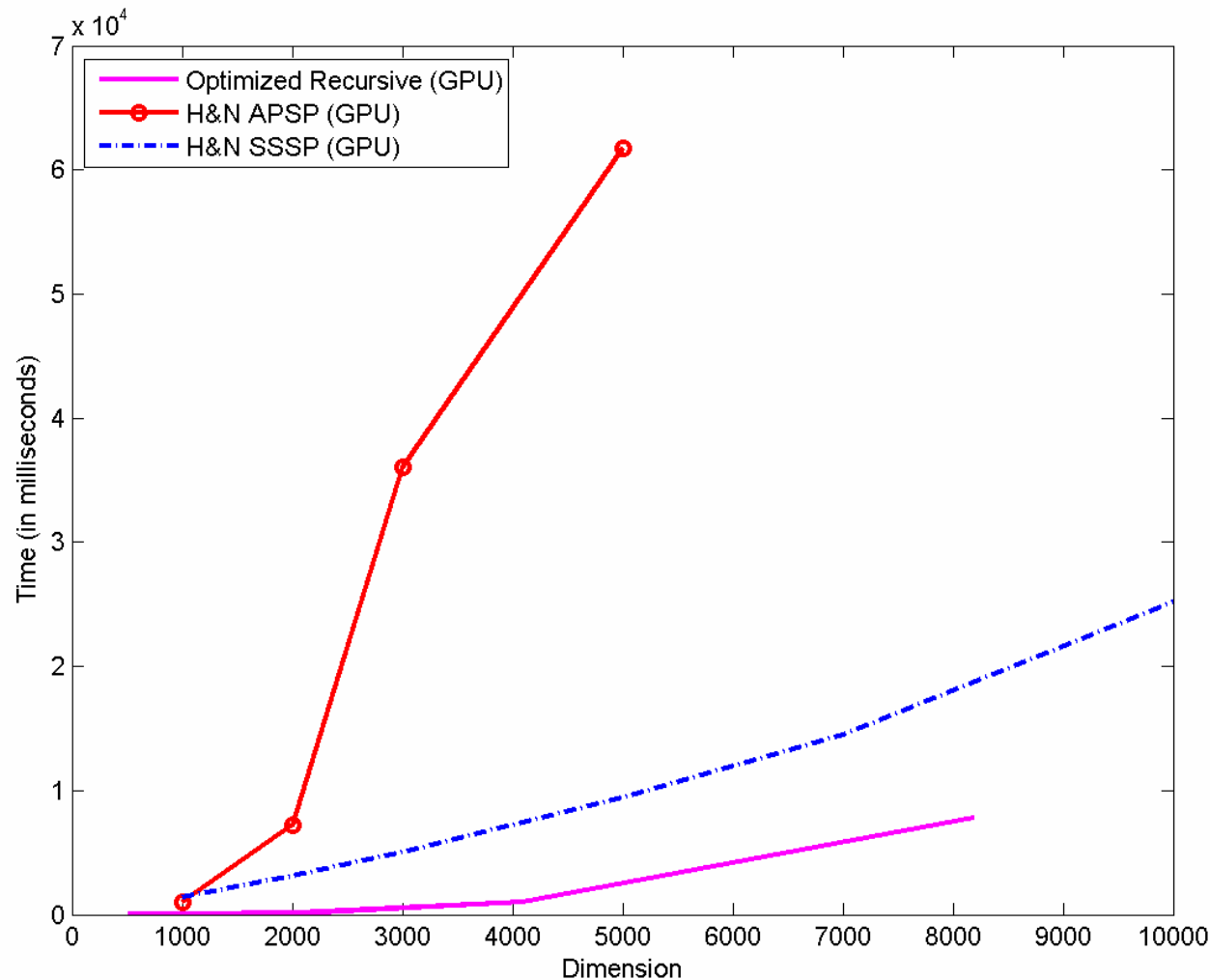
480x

*Using a variant of
new GEMM
(Volkov & Demmel)*

A closer look on optimization



How does it compare?

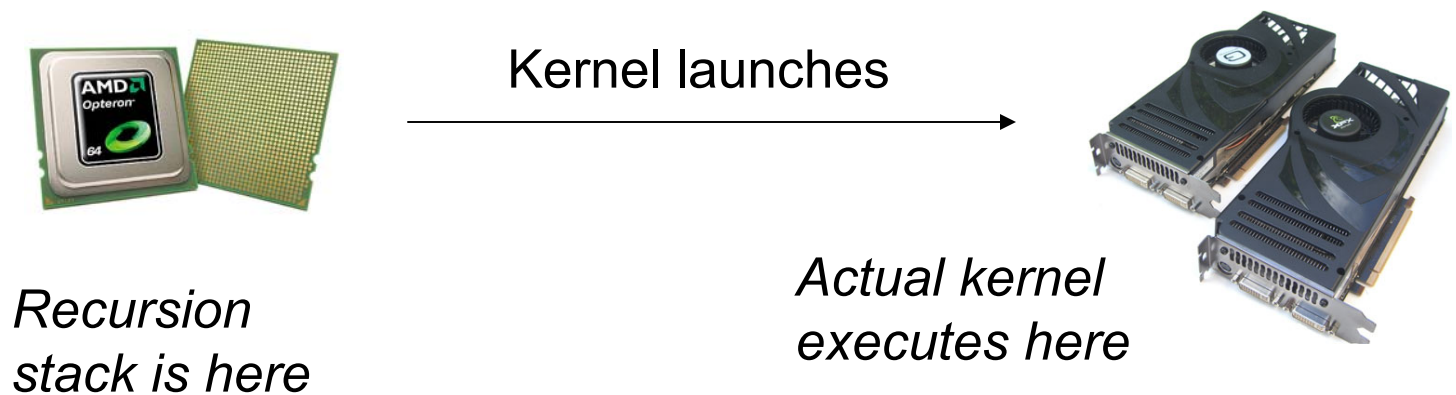


Even for extremely sparse graphs with average degree 6:

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.



- 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

- [T97] Sivan Toledo, “Locality of Reference in LU Decomposition with Partial Pivoting”. *SIAM Journal on Matrix Analysis and Applications*, 1997
- [DEL01] Jack Dongarra, Victor Eijkhout, Piotr Łuszczek, “Recursive approach in sparse matrix LU factorization”, *Scientific Programming*, 2001
- [EGJK04] Erik Elmroth, Fred Gustavson, Isak Jonsson, Bo Kagstrom, “Recursive Blocked Algorithms and Hybrid Data Structures for Dense Matrix Library Software”, *SIAM Review*, 2004.
- [PPP04] Joon-Sang Park, Michael Penner, Viktor K. Prasanna, “Optimizing Graph Algorithms for Improved Cache Performance”, *IEEE Transactions on Parallel and Distributed Systems*, 2004
- [FSH04] K. Fatahalian, J.Sugerman, and P.Hanrahan, “Understanding the Efficiency of GPU Algorithms for Matrix-Matrix Multiplication”, *Graphics Hardware*, 2004
- [VD08] Vasily Volkov, James Demmel, “LU, QR and Cholesky Factorizations using Vector Capabilities of GPUs”, *Technical Report, UC Berkeley*, 2008
- [HN07] Pawan Harish, P. J. Narayanan, “Accelerating Large Graph Algorithms on the GPU Using CUDA”, *HiPC*, 2007
- [DN07] P. D’Alberto, A.Nicolau, “R-Kleene: A High-Performance Divide-and-Conquer Algorithm for the All-Pair Shortest Path for Densely Connected Networks“, *Algorithmica*, 2007