NVIDIA GPU - odd dwarfs

Julian Naß and Marcus Völker

12. Februar 2015

Overview

Dense Linear Algebra

Paper

Benchmarking GPUs to Tune Dense Linear Algebra, V. Volkov and J. Demmel

Problem

- Matrix-matrix multiply routine(GEMM)
- LU, QR, Cholesky factorizations
- Benchmarks to analyze the performance
- Improve vendor's implementation

Dense Linear Algebra - Setup

Hardware

- 4 GPUs
  - 8600GTS
  - 8800GTX
  - 9800GTX
  - GTX280
- 2 CPUs
  - Core2 Duo E6700 2.67GHz
  - Core2 Quad Q6850 3.0GHz
- PCIe 1.1 x16 interface

Software

- CUDA
- CUBLAS 1.1 / 2.0
- Intel MKL 10.0

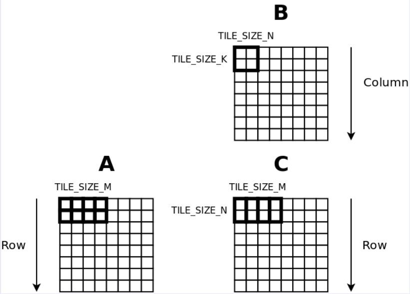
Dense Linear Algebra - GEMM Implementation

What is implemented?

- $C := \alpha AB + \beta C$ and $C := \alpha AB^t + \beta C$ cases of matrix multiplication(GEMM)
- $C := \alpha AA^t + \beta C$ for symmetric rank operations (SYRK)
- A(m x k), B(k x n) and C(m x n)

Dense Linear Algebra - GEMM Implementation

Implementation



How is it implemented?

- A,B and C are blocked
- A and C blocks are in saved registers and column major
- B blocks in shared memory and row major

http://cuda.ac.upc.edu/node/21

Dense Linear Algebra - GEMM Implementation

What is special?

- Optimization through micro-benchmarks
  - Vector length of 64
    - Short as possible to avoid extra costs
    - 98% of arithmetic peak in register-to-register multiply-and-add instructions
  - CUDA as fastest API for programming the GPU
  - Instructions with shared memory run slower
  - Global barrier much cheaper on GPU (1.3-2.0s)
    - Synchronization with CPU 1.5-5.4x slower
  - Pipeline latency best on NVIDIA GPUs (especially on GTX280)

Dense Linear Algebra - GEMM Implementation Comparison

Comparison

	CUBLAS 1.1	Our code
C's block, where stored	32×32, regs	64×16, regs
A's block, where stored	32×32, smem	64×1, regs
B's block, where stored	32×32, smem	16×16, smem
Vector length	512	64
Scalar registers per scalar thread	15	30
Registers per vector thread	30 KB	7.5 KB
smem per vector thread	8.3 KB	1.1 KB
Threads/core, 8800GTX	1	4
Warps/core, 8800GTX	16	8
Instructions in inner loop	115	312
MAD instructions	64	256
smem-to-register MOVs	32	0
Expected, % of peak	44%	58%
Sustained, % of peak	36—44%	58—60%

Comparison vendor vs paper

- A and B blocks in CUBLAS in smem
- Smaller vector length
- Best performance on 4 threads
- 2x more warps per core in CUBLAS
- 2x less scalar registers per scalar thread in CUBLAS
- CUBLAS 1.6x slower

Dense Linear Algebra - GEMM Results

Comparison

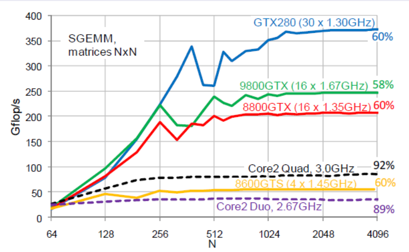
GPU	SP peak, Gflop/s	SGEMM("N", "N", ...)			SSYRK("L", "N", ...)		DP peak, Gflop/s	DGEMM	DSYRK	
		CUBLAS1.1	ours	estimate	CUBLAS2.0	ours		ours	CUBLAS2.0	ours
8600GTS	93	37%	60%	58%	36%	60%	—	—	—	—
8800GTX	346	37%	60%	58%	37%	60%	—	—	—	—
9800GTX	429	36%	58%	58%	36%	58%	—	—	—	—
GTX280	624	44%	60%	58%	45%	60%	78	97%	35%	95%

GPU Results

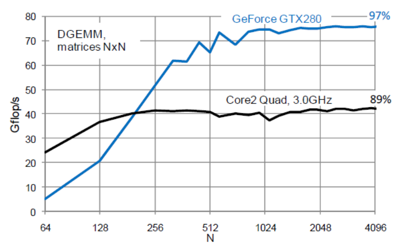
- On all GPUs 58-60% of peak => scales linearly with clock rate and number of cores
- Double precision on GTX280 97% of peak in GEMM and 95% of peak in SYRK

Dense Linear Algebra - GEMM Results

Comparison



Comparison



GPU Results

- CPUs 89-92% of peak
- In double precision CPU better in smaller matrices
- GTX280 better on bigger matrices

Dense Linear Algebra - LU, QR, Cholesky Implementation

What is implemented?

- Matrices in column-major layout

How is it implemented?

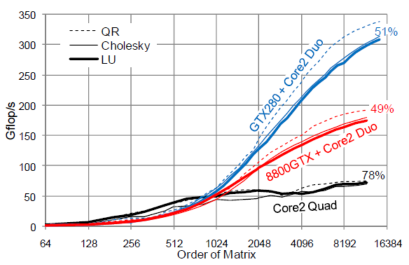
- Panel factorization
  - Only BLAS1 and BLAS2 operations
- LU factorization via right-looking scheme
  - More thread-level parallelism
- Update the entire matrix as soon as next block column is available in QR and Cholesky
- Transferring matrix panels from GPU to CPU memory and back

Dense Linear Algebra - LU, QR, Cholesky Results

Comparison

	Q6850	8800GTX+E6700		GTX280+E6700	
	Gflop/s	Gflop/s	speedup	Gflop/s	speedup
LU	73	179	2.5×	309	4.1×
Cholesky	70	183	2.7×	315	4.4×
QR	75	192	2.6×	340	4.4×
SGEMM	88	208	2.4×	375	4.3×
peak	96	388	4.0×	667	6.9×

Comparison



Results

- Core2Quad 78% of peak
- GPUs+Core2Duo 49-51% of peak

Dense Linear Algebra - Conclusion

Conclusion

- Fastest GEMM and SYRK implementation
- Fastest LU,QR and Cholesky factorization
- GEMM of CUBLAS 2.0 based on Volkov's and Demmel's implementation

Spectral Methods

Paper

High Performance Discrete Fourier Transforms on Graphics Processors

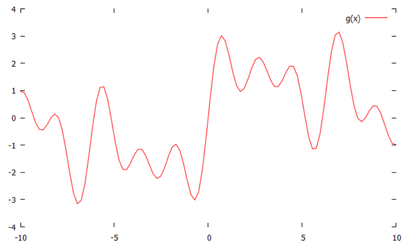
NK Govindaraju, B. Lloyd, Y. Dotsenko, B. Smith, and J. Manferdelli

Problem

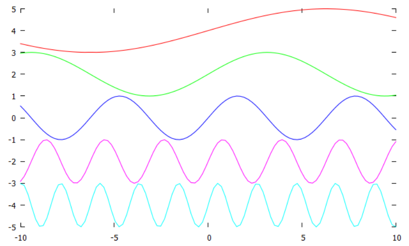
- Discrete Fourier Transforms (DFT)
- Implemented with Fast Fourier Transform (FFT)
- Fourier Transform decomposes a function into a sum of sine waves (frequencies)
- Applications in many engineering fields, physics, cryptography, etc.

Spectral Methods - Fourier Transform

Function



Decomposition



Discrete Fourier Transform

DFT transforms an N-point sequence into a different N-point sequence

Spectral Methods - Setup

Hardware

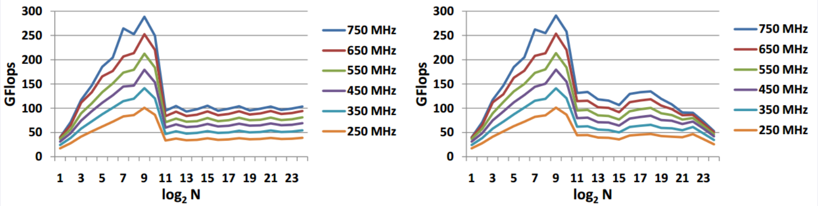
- 3 GPUs
  - 8800 GTX
  - 8800 GTS
  - GTX280
- Intel QX9650 CPU (3.0 GHz quad-core)
- 4 GB DDR3 RAM

Software

- Paper implementation (global memory and hierarchical memory versions)
- CUFFT 1.1 (NVIDIA)
- MKL 10.0.2 (Intel)

Spectral Methods - Results

Different memory algorithms

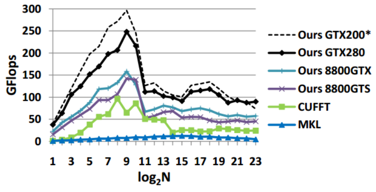


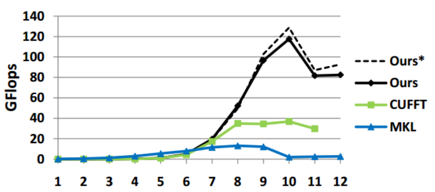
GPU Results - General

$N > 2^{10}$ is performed with different memory algorithms (Because of shared memory limit)

Spectral Methods - Results

Batched 1D, Single 2D FFTs





Comparisons

- For Batched 1D, up to 4 times faster than CUFFT, up to 19 times faster than MKL
- For Single 2D, up to 3 times faster than CUFFT, up to 61 times faster than MKL

Structured Grid

Paper

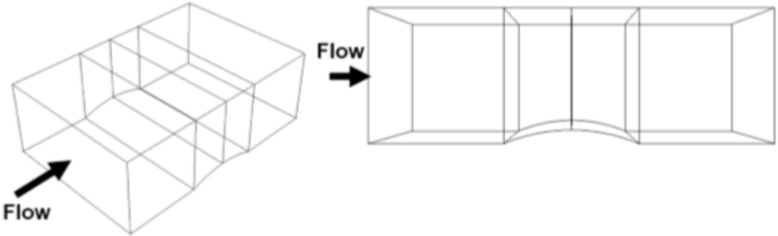
GPGPU parallel algorithms for structured-grid CFD codes
C. P. Stone, E. P. N. Duque, Y. Zhang, D. Car, J. D. Owens and R. L. Davis

Problem

- Computational Fluid Dynamics (CFD)
- Many CFD implementations share component algorithms
- Applied to Navier-Stokes with approximate factorization (AF)

Structured Grid - Fluid Simulation

World



Fluid Simulation

Goal: Simulate fluid moving in an environment

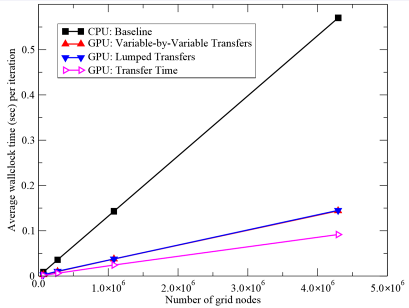
Structured Grid - Setup

Hardware

- Intel X5677 (quad-core) Xeon
- 12 GB DDR3 memory
- NVIDIA Tesla C2050 GPU (Fermi architecture)

Structured Grid - Results

Comparison with CPU



Inviscid Fluid test

- Speed-up of 3.2 to 3.9
- 63% of time is transfer time

$\Rightarrow$ Speed-up of 11-21x theoretically possible when eliminating transfer times

- Authors estimate more performance with efficient memory usage

MapReduce

Paper

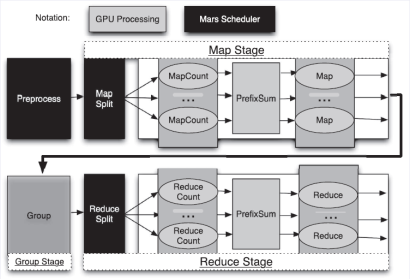
Mars: Accelerating MapReduce with Graphics Processors

Problem

- Improve MapReduce
- Flexibility, Programmability and High Performance

MapReduce - Mars

Mars



Mars

- group output by key
- not all stages needed for some applications

MapReduce - Setup

Hardware

- NVIDIA GTX280
- Intel Core2Quad Q6600(2.4Ghz)

Software

- CentOS 5.1
- MarsCUDA, MarsCPU
- Phoenix 2.0
- CUDA 2.2

MapReduce - Programability

Application Code Size

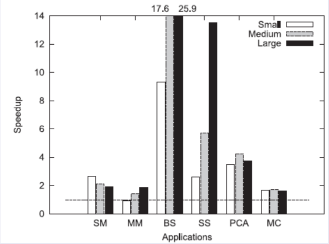
Applications	Phoenix	MarsCUDA/MarsCPU	CUDA
String Match	206	147	157
Matrix Multiplication	178	72	68
Black-Scholes	199	147	721
Similarity Score	125	82	615
Principal component analysis	297	168	583
Monte Carlo	251	203	359

Comparison

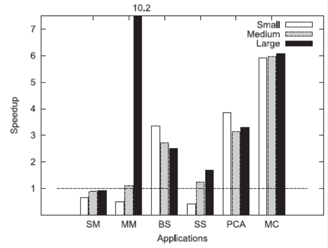
- Smaller code size on Mars
- MarsCUDA up to 7x smaller than CUDA

MapReduce - MarsCUDA vs MarsCPU

MarsCPU over Phoenix



MarsCUDA over MarsCPU

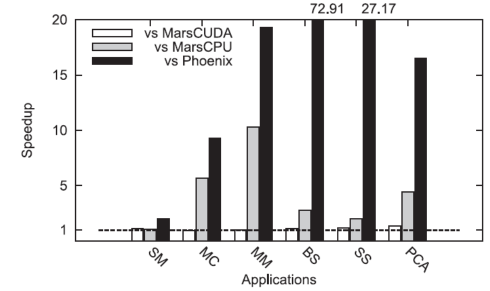


Comparison

- MarsCPU speed-up up to 25.9x over Phoenix
- MarsCUDA up to 10x faster over MarsCPU

MapReduce - MarsCUDA vs MarsCPU

GPU/CPU coprocessing



Comparison

- high speed-up over Phoenix and MarsCPU
- speed-up over MarsCUDA is limited

Graph Traversal

Paper

High Performance and Scalable GPU Graph Traversal
D. Merrill, M. Garland and A. Grimshaw

Problem

- Breadth-first search (BFS)
- Core primitive for higher-level algorithms

Graph Traversal - Setup

Data

- 13 different data sets
- from 400k to 50M vertices

Hardware

- 3 different CPUs
  - 3.4GHz Core i7 2600K (for sequential)
  - 2.5GHz Core i7 4-core (for parallel non-random)
  - 2.7 GHz Xeon X5570 8-core (for parallel random)
- up to four Tesla C2050 (Fermi architecture)

Graph Traversal - Results

Comparison with CPU

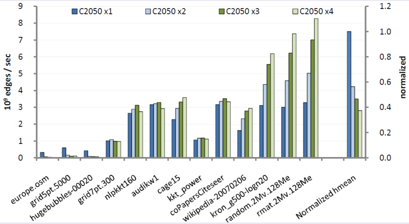
Graph Dataset	CPU Sequential†	CPU Parallel	NVIDIA Tesla C2050 (*hybrid*)			
			Label Distance		Label Predecessor	
	10^9 TE/s	10^9 TE/s	10^9 TE/s	Speedup	10^9 TE/s	Speedup
europe.osm	0.029		0.31	11x	0.31	11x
grid5pt.5000	0.081		0.60	7.3x	0.57	7.0x
hugebubbles-00020	0.029		0.43	15x	0.42	15x
grid7pt.300	0.038	0.12††	1.1	28x	0.97	26x
nlpkkt160	0.26	0.47††	2.5	9.6x	2.1	8.3x
audikw1	0.65		3.0	4.6x	2.5	4.0x
cage15	0.13	0.23††	2.2	18x	1.9	15x
kkt_power	0.047	0.11††	1.1	23x	1.0	21x
coPapersCiteseer	0.50		3.0	5.9x	2.5	5.0x
wikipedia-20070206	0.065	0.19††	1.6	25x	1.4	22x
kron_g500-logn20	0.24		3.1	13x	2.5	11x
random.2Mv.128Me	0.10	0.50†††	3.0	29x	2.4	23x
rmat.2Mv.128Me	0.15	0.70†††	3.3	22x	2.6	18x

Results

- Speed-up of up to 29x
- Speed-up is dependant on average out-degree
- Using very sophisticated approach

Graph Traversal - Results

Multiple GPUs



Results

- Improvement dependant on search depth
- In cases with high search depth worse than single GPU

Evaluation

Core points

- CUDA is C-like, so easy to learn for programmers
- Nice speed-up compared to CPU (up to 60x for selected problems)
- Memory usage is important
- Optimizations are still necessary

References

NVIDIA Tesla: A Unified Graphics and Computing Architecture
Lindholm, E.; Nickolls, J.; Oberman, S.; Montrym, J., Micro, IEEE , vol.28, no.2, pp.39,55, March-April 2008

Fermi: NVIDIA's Next Generation CUDA Compute Architecture
NVIDIA, 2009

Benchmaking GPUs to Tune Dense Linear Algebra
V . Volkov and J. W. Demmel, International Conference for High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008

High Performance Discrete Fourier Transforms on Graphics Processors
NK Govindaraju, B. Lloyd, Y. Dotsenko, B. Smith, and J. Manferdelli, Proceedings of the 2008 ACM/IEEE conference on Supercomputing

References

GPGPU parallel algorithms for structured-grid CFD codes C.P. Stone, E.P.N. Duque, Y. Zhang, D. Car, J.D. Owens and R.L. Davis, AIAA CFD Conference 2011

Mars: Accelerating MapReduce with Graphics Processors Wenbin Fang; Bingsheng He; Qiong Luo; Govindaraju, N.K, IEEE Transactions on Parallel and Distributed Systems, vol.22, no.4, pp.608,620, April 2011

High Performance and Scalable GPU Graph Traversal D. Merrill, M. Garland and A. Grimshaw, 17th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming, 2011

Credits

Julian Naß

Discrete Linear Algebra + Implementation, MapReduce, Evaluation

Marcus Völker

Architecture, Spectral Methods, Structured Grid, Graph Traversal

Thank you for your attention!