L11: Sparse Linear Algebra on GPUs

Administrative Issues

- Next assignment, triangular solve
  - Due 5PM, Tuesday, March 15
  - handin cs6963 lab 3 <probfile>"
- Project proposals
  - Due 5PM, Wednesday, March 7 (hard deadline)
  - handin cs6963 prop <pdffile>

Triangular Solve (STRSM)

```
for (j = 0; j < n; j++)
   for (k = 0; k < n; k++)
       if (B[j*n+k] != 0.0f) {
            for (i = k+1; i < n; i++)
               B[j*n+i] -= A[k * n + i] * B[j * n + k];
       }
```

Equivalent to:

```
cublasStrsm('l' /* left operator */, 'l' /* lower triangular */,
            'N' /* not transposed */, 'u' /* unit triangular */,
             N, N, alpha, d_A, N, d_B, N);
```

See: http://www.netlib.org/blas/strsm.f

A Few Details

- C stores multi-dimensional arrays in row major order
- Fortran (and MATLAB) stores multi-dimensional arrays in column major order
  - **Confusion alert:** BLAS libraries were designed for FORTRAN codes, so column major order is implicit in CUBLAS!

Dependences in STRSM

```
for (j = 0; j < n; j++)
  for (k = 0; k < n; k++)
     if (B[j*n+k] != 0.0f) {
          for (i = k+1; i < n; i++)
            B[j*n+i] -= A[k * n + i] * B[j * n + k];
     }
```

Which loop(s) “carry” dependences? Which loop(s) is(are) safe to execute in parallel?

Assignment

- Details:
  - Integrated with simpleCUBLAS test in SDK
  - Reference sequential version provided

1. Rewrite in CUDA
2. Compare performance with CUBLAS library

Performance Issues?

- \+ Abundant data reuse
- \- Difficult edge cases
- \- Different amounts of work for different <j,k> values
- \- Complex mapping or load imbalance

Outline

- Next assignment
- For your projects:
  - “Debunking the 100X GPU vs. CPU Myth: An Evaluation of Throughput Computing on CPU and GPU”, Lee et al., ISCA 2010.
- Sparse Linear Algebra
- Readings:
  - “Implementing Sparse Matrix-Vector Multiplication on Throughput Oriented Processors,” Bell and Garland (Nvidia), SC09, Nov. 2009.
  - “Model-driven Autotuning of Sparse Matrix-Vector Multiply on GPUs”, Choi, Singh, Vuduc, PPoPP 10, Jan. 2010.
  - “Optimizing sparse matrix-vector multiply on emerging multicore platforms,” Journal of Parallel Computing, 35(3):178-194, March 2009. (Expanded from SC07 paper.)

Overview: CPU and GPU Comparisons

- Many projects will compare speedup over a sequential CPU implementation
  - Ok for this class, but not for a research contribution
- Is your CPU implementation as "smart" as your GPU implementation?
  - Parallel?
  - Manages memory hierarchy?
  - Minimizes synchronization or accesses to global memory?

The Comparison

- Architectures
  - Intel i7, quad-core, 3.2GHz, 2-way hyper-threading, SSE, 32KB L1, 256KB L2, 8MB L3
  - Same i7 with Nvidia GTX 280
- Workload
  - 14 benchmarks, some from the GPU literature

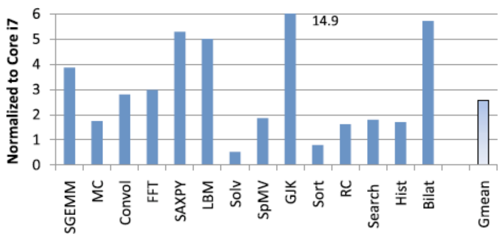
Architectural Comparison

	Core i7-960	GTX280
Number PEs	4	30
Frequency (GHz)	3.2	1.3
Number Transistors	0.7B	1.4B
BW (GB/sec)	32	141
SP SIMD width	4	8
DP SIMD width	2	1
Peak SP Scalar FLOPS (GFLOPS)	25.6	116.6
Peak SP SIMD Flops (GFLOPS)	102.4	311.1/933.1
Peak DP SIMD Flops (GFLOPS)	51.2	77.8

Workload Summary

Kernel	Application	SIMD	TLP	Characteristics
SGEMM (**SGEMM**) [48]	Linear algebra	Regular	Across 2D Tiles	Compute bound after tiling
Monte Carlo (**MC**) [34, 9]	Computational Finance	Regular	Across paths	Compute bound
Convolution (**Conv**) [16, 19]	Image Analysis	Regular	Across pixels	Compute bound; BW bound for small filters
FFT (**FFT**) [17, 21]	Signal Processing	Regular	Across smaller FFTs	Compute/BW bound depending on size
SAXPY (**SAXPY**) [46]	Dot Product	Regular	Across vector	BW bound for large vectors
LBM (**LBM**) [32, 45]	Time Migration	Regular	Across cells	BW bound
Constraint Solver (**Solv**) [14]	Rigid body physics	Gather/Scatter	Across constraints	Synchronization bound
SpMV (**SpMV**) [50, 8, 47]	Sparse Solver	Gather	Across non-zero	BW bound for typical large matrices
GJK (**GJK**) [38]	Collision Detection	Gather/Scatter	Across objects	Compute Bound
Sort (**Sort**) [15, 39, 40]	Database	Gather/Scatter	Across elements	Compute bound
Ray Casting (**RC**) [43]	Volume Rendering	Gather	Across rays	4-8MB first level working set, over 500MB last level working set
Search (**Search**) [27]	Database	Gather/Scatter	Across queries	Compute bound for small tree, BW bound at bottom of tree for large tree
Histogram (**Hist**) [53]	Image Analysis	Requires conflict detection	Across pixels	Reduction/synchronization bound
Bilateral (**Bilat**) [52]	Image Analysis	Regular	Across pixels	Compute Bound

Performance Results



Apps.	SGEMM	MC	Conv	FFT	SAXPY	LBM	Solv	SpMV	GJK	Sort	RC	Search	Hist	Bilat
Core i7-960 GTX280	94 364	0.8 1.4	1250 3500	71.4 213	16.8 88.8	85 426	103 52	4.9 9.1	67 1020	250 198	5 8.1	50 90	1517 2583	83 475

CPU optimization

- Tile for cache utilization
- SIMD execution on multimedia extensions
- Multi-threaded, beyond number of cores
- Data reorganization to improve SIMD performance

Sparse Linear Algebra

- Suppose you are applying matrix-vector multiply and the matrix has lots of zero elements
  - Computation cost? Space requirements?
- General sparse matrix representation concepts
  - Primarily only represent the nonzero data values
  - Auxiliary data structures describe placement of nonzeros in "dense matrix"

GPU Challenges

- Computation partitioning?
- Memory access patterns?
- Parallel reduction

BUT, good news is that sparse linear algebra performs TERRIBLY on conventional architectures, so poor baseline leads to improvements!

Some common representations

A = [1 7 0 0; 0 2 8 0; 5 0 3 9; 0 6 0 4]

data = [* 1 7; * 2 8; 5 3 9; 6 4 *]   offsets = [-2 0 1]

DIA: Store elements along a set of diagonals.

ptr = [0 2 4 7 9]
indices = [0 1 1 2 0 2 3 1 3]
data = [1 7 2 8 5 3 9 6 4]

Compressed Sparse Row (CSR): Store only nonzero elements, with "ptr" to beginning of each row and "indices" representing column.

data = [1 7 *; 2 8 *; 5 3 9; 6 4 *]   indices = [0 1 *; 1 2 *; 0 2 3; 1 3 *]

ELL: Store a set of K elements per row and pad as needed. Best suited when number non-zeros roughly consistent across rows.

row = [0 0 1 1 2 2 2 3 3]
indices = [0 1 1 2 0 2 3 1 3]
data = [1 7 2 8 5 3 9 6 4]

COO: Store nonzero elements and their corresponding "coordinates".

CSR Example

```
for (j=0; j<nr; j++) {
  for (k = ptr[j]; k<ptr[j+1]-1; k++)
    t[j] = t[j] + data[k] * x[indices[k]];
```


Summary of Representation and Implementation

Kernel	Granularity	Coalescing	Bytes/Flop 32-bit	Bytes/Flop 64-bit
DIA	thread : row	full	4	8
ELL	thread : row	full	6	10
CSR(s)	thread : row	rare	6	10
CSR(v)	warp : row	partial	6	10
COO	thread : nonz	full	8	12
HYB	thread : row	full	6	10

Table 1 from Bell/Garland: Summary of SpMV kernel properties.

Other Representation Examples

- Blocked CSR
  - Represent non-zeros as a set of blocks, usually of fixed size
  - Within each block, treat as dense and pad block with zeros
  - Block looks like standard matvec
  - So performs well for blocks of decent size
- Hybrid ELL and COO
  - Find a "K" value that works for most of matrix
  - Use COO for rows with more nonzeros (or even significantly fewer)

Stencil Example

What is a 3-point stencil? 5-point stencil? 7-point? 9-point? 27-point?

Examples:

```
a[i] = (b[i-1] + b[i+1])/2;
a[i][j] = (b[i-1][j] + b[i+1][j] + b[i][j-1] + b[i][j+1])/4;
```

How is this represented by a sparse matrix?

Stencil Result (structured matrices)

See Figures 11 and 12, Bell and Garland

Unstructured Matrices

See Figures 13 and 14

Note that graphs can also be represented as sparse matrices. What is an adjacency matrix?

PPoPP paper

- What if you customize the representation to the problem?
- Additional global data structure modifications (like blocked representation)?
- Strategy
  - Apply models and autotuning to identify best solution for each application

Summary of Results

BELLPACK (blocked ELLPACK) achieves up to 29 Gflop/s in SP and 15.7 Gflop/s in DP

Up to 1.8x and 1.5x improvement over Bell and Garland.

This Lecture

- Exposure to the issues in a sparse matrix vector computation on GPUs
- A set of implementations and their expected performance
- A little on how to improve performance through application-specific knowledge and customization of sparse matrix representation

What's coming

- Next time: Application case study from Kirk and Hwu (Ch. 8, real-time MRI)
- Wednesday, March 2: two guest speakers from last year's class
  - BOTH use sparse matrix representation!
  - Shreyas Ramalingam: program analysis on GPUs
  - Pascal Grosset: graph coloring on GPUs