Porting the NAS-NPB Conjugate Gradient Benchmark to CUDA

NVIDIA Corporation

Outline

- Overview of CG benchmark
- Overview of CUDA Libraries
  - CUSPARSE
  - CUBLAS
- Porting Sequence
  - Algorithm Analysis
  - Data/Code Analysis

This porting approach uses CUDA Libraries *exclusively*. (We will not write any kernels or device code.)

NPB Benchmarks

- Written by NASA in 1994 to help benchmark and prove out parallel coding methodologies and architectures.
- Suite of benchmarks:
  - Integer Sort
  - Conjugate Gradient
  - CFD
  - FFT
  - And others…
- Come in several flavors
  - Serial
  - OpenMP
  - Have been modified/update/added to by others (e.g. OpenCL)

NPB Benchmarks

- Each benchmark includes several different problem sizes called “CLASS”es – e.g. A (small), B (medium), C (large), etc.
- Some were originally written in Fortran (e.g. CG), some in C (e.g. IS)
- Source: http://www.nas.nasa.gov/publications/npb.html
- Original Whitepaper: http://www.nas.nasa.gov/assets/pdf/techreports/1994/rnr-94-007.pdf
- SNU-NPB (Seoul National University) Update:
  - All are re-written in C
  - Added some OpenCL versions
  - http://aces.snu.ac.kr/Center_for_Manycore_Programming/SNU_NPB_Suite.html

CG (Conjugate Gradient Solver) Benchmark

- "A conjugate gradient method is used to compute an approximation to the smallest eigenvalue of a large sparse symmetric positive definite matrix. This kernel is typical of unstructured grid computations in that it tests irregular long distance communication employing unstructured matrix vector multiplication."
- Uses a variety of dense vector algebra, and sparse matrix-dense vector algebra (SpMV)
- Original code written in Fortran, uses no libraries or other high level constructs. (We will work with the C translation created by SNU, there is no functional difference.)
- We will use CUBLAS for the dense vector algebra, and CUSPARSE for the sparse matrix – dense vector algebra.

What is CUBLAS?

- A Linear Algebra library which duplicates many functions from the well-known BLAS (Basic Linear Algebra Subprograms) libraries for performing dense vector and matrix algebra.
- Automatically uses the GPU, and (generally) requires that the data be explicitly managed: Data must be resident on the GPU before the CUBLAS function (e.g. DGEMM, DDOT) is invoked.
- Most vector or matrix results automatically remain on the GPU, and must be explicitly moved to the host if needed there.
- Some scalar results (e.g. DOT product) can be automatically returned to the host.
- Typical routine naming:
  - DAXPY= Double precision A times X plus Y (X, Y are vectors, A is scalar)
  - DDOT = Double precision DOT product
- Documentation: http://docs.nvidia.com/cuda/cublas/index.html

What is CUSPARSE?

- A set of linear algebra subroutines used for handling sparse matrices.
- Automatically uses the GPU, and (generally) requires that the data be explicitly managed: Data must be resident on the GPU before the CUSPARSE function (e.g. SpMV, SpMM) is invoked.
- Most vector or matrix results automatically remain on the GPU, and must be explicitly moved to the GPU if needed.
- Supports several different sparse matrix storage formats:
  - CSR - Compressed Sparse Row (data , row pointers, column indices)
  - COO - Coordinate Format (each data element has x,y coordinates)
  - CSC, ELL, HYB, BSR, etc.
- Typical naming
  - Dcsrspmv= Double precision CSR sparse matrix – dense vector multiply
- Documentation: http://docs.nvidia.com/cuda/cusparse/index.html

Why use libraries?

- Generally much quicker than writing your own routines.
- Tap into GPU experts for difficult problems (e.g. optimizing sparse matrix-vector multiply)
- Automatically handle many aspects of device management, and configuration
- Take advantage of performance increases as new (more optimized) library versions are released.
- Reduced code size.
- Higher level of abstraction/easier to port/maintain/update.

CG Benchmark - Main Loop

"Inverse Power Method"

Create initial estimate of x: $[1,1,1, \ldots, 1]^T$

DO it =1, niter (number of iterations of main loop – varies with problem size)

  Solve Az = x using CG method (next slide) and return ||r|| (residual)

  zeta = lambda + $1/(x^Tz)$

  Print it, ||r||, and zeta

  x = z/||z||

END DO

CG Benchmark - CG Loop

"The solution z to the linear system of equations Az = x is to be approximated using the conjugate gradient method"

r = x

rho = $r^T r$

p = r

DO it =1, 25

q = Ap (SpMV)

alpha = rho /($p^T q$)

z = z + (alpha)(p)

rho_0 = rho

r = r – (alpha)(q)

rho = $r^T r$

beta = rho/rho_0

P = r +(beta)(p)

END DO

||r|| = ||x – Az|| (another SpMV)

General Porting approach

- Identify main data components (A, x, p, r, z, etc.) which need to be resident on the GPU, and allocate GPU storage for them
- After the main data components are initially set up on the host, copy to GPU
- Identify key math operations in the code (dot product, matrix-vector multiply, etc.), and convert to appropriate CUBLAS or CUSPARSE function
- Leave most vector and matrix data exclusively on the GPU – no need to copy data back and forth.
- Actual results/convergence indicators (zeta, ||r||) are scalar in nature
- Leave most setup, control flow, and reporting functions unchanged

Summary

- Didn’t write a line of GPU “device code”
- Overall code size, complexity reduced, and easier to read
- Approximate results:
  - ~2x speedup vs. OpenCL version
  - ~3x speedup vs. OpenMP version (4 cores)
  - ~5x speedup vs. Serial version

Where to get help?

- Sign up as a registered developer: https://developer.nvidia.com/
- Access the NVIDIA community forums: https://devtalk.nvidia.com/
- OpenACC: http://www.openacc-standard.org/
- StackOverflow:
  - CUDA: http://stackoverflow.com/questions/tagged/cuda
  - Thrust: http://stackoverflow.com/questions/tagged/thrust
  - OpenACC: http://stackoverflow.com/questions/tagged/openacc

Questions?