General Project Update

Dr. Joshua Booth, Hayden V Estes

Project Background, Focus, and Purpose

Purpose:

Understanding the impact of data types on fault tolerance is crucial, particularly in modern heterogeneous systems with diverse processors.

Focus:

Our focus is exploring concurrent CPU-GPU computation with variable precision levels for improved fault tolerance and memory transfer rates in the preconditioned conjugate gradient algorithm.

Background:

This is a continuation of the work by Manu Shantharam, Sowmyalatha Srinivasmurthy, and Padma Raghavan in 2012. “Fault tolerant preconditioned conjugate gradient for sparse linear system solution”.

Technical Specifications

- ***<u>The program creates two CPU (p)threads</u>***: one for running the CPU CG implementation in C and another for making CUDA calls in C for the GPU CG implementation.

- ***<u>Preconditioners are generated beforehand</u>*** using Matlab/Octave. All matrices are converted to ***<u>compressed sparse row (CSR)</u>*** format, and the main matrices are reordered using **reverse *<u>Cuthill-McKee ordering (RCM)</u>***.

- Both functions support single and double precisions. The ***<u>GPU uses single precision</u>***, while the ***<u>CPU uses double precision</u>***.

```
Structure Data_CG:
  - matrix_count: number of matrices
  - files: array of matrix file paths
  - pfiles: array of preconditioner file paths
  - maxit: maximum number of iterations
  - tol: tolerance value

Function batch_CCG/batch_CuCG:
  - Input: arg (pointer to Data_CG structure)
  - Output: None
  - Open a results file for writing
  - Iterate over each matrix in the files array
        - Read the matrix from the file
        - Allocate arrays for x and b
        - Run the CCG/CuCG algorithm on the
              matrix and preconditioner (if available)
        - Write the results to the results file
        - Free memory
  - Close the results file

Function main:
  - Set initial values and variables
  - Read user options or command-line arguments
  - Find matrix and preconditioner files
  - Launch batch_CCG in a pthread
  - Run batch_CuCG on current thread
  - Wait for both CG's to finish
  - Clean up memory
  - Return 0
```

More Technical Specifications

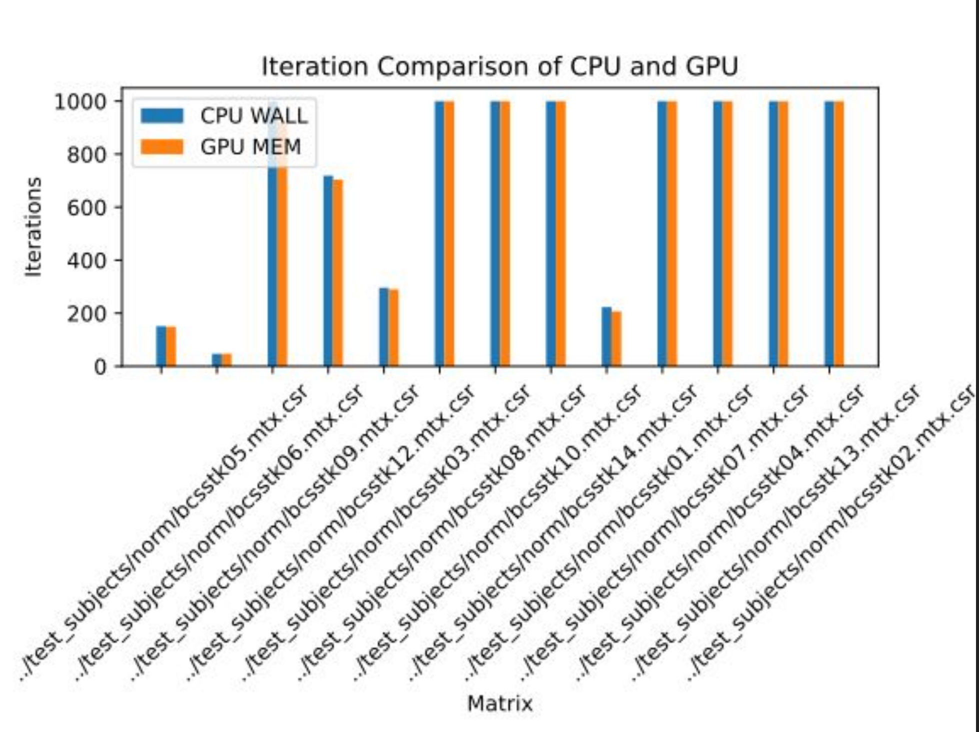
- The CUDA calculations utilize ***CUBLAS and CUSPARSE*** libraries, while the C calculations are performed using custom functions.

- In the GPU function, the answer vectors ***data is copied back to the host after every n iterations***. This step is specifically for ***fault-tolerance*** purposes and is ***timed separately***.

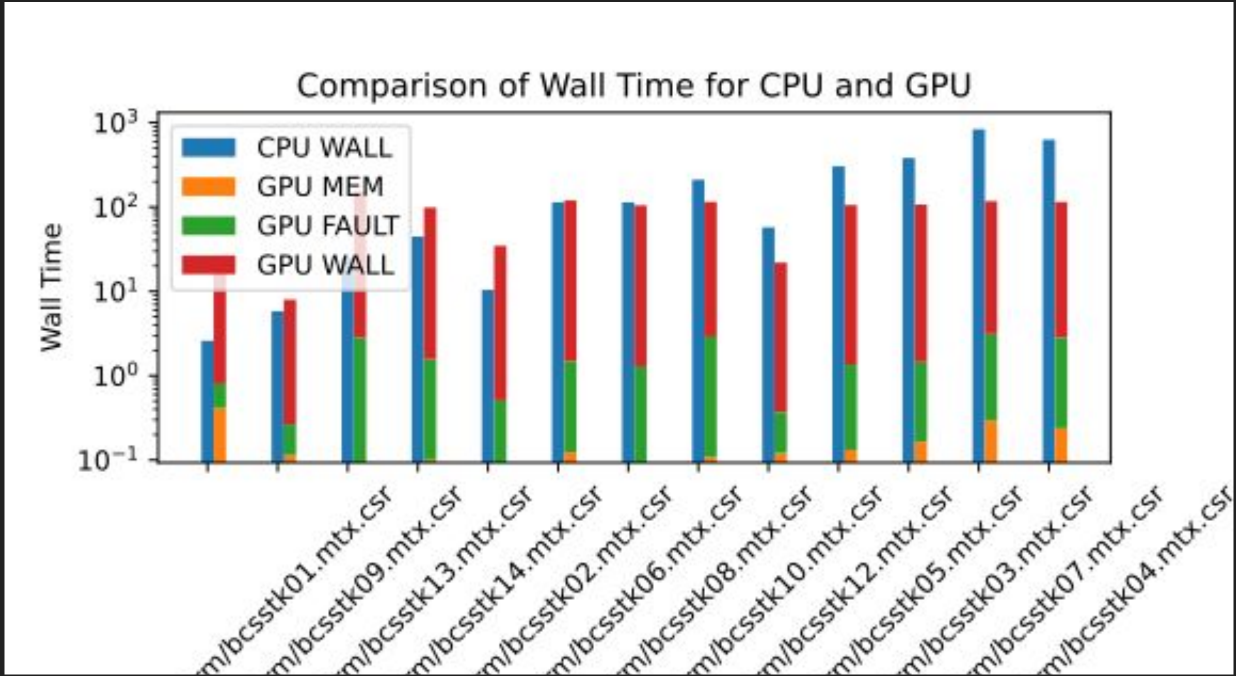
```
Read A Matrix
Read M Matrix

MEM OMP WALL TIME START

Malloc A_matrix
... M_matrix
... b_vec ... x_vec ... r_vec
... p_vec ... q_vec ... z_vec

cudaMallocPitch A_matrix
... M_matrix
... b_vec ... x_vec ... r_vec
... p_vec ... q_vec ... z_vec

cudaMemcpy(A_matrix)
... M_matrix
... b_vec ... x_vec ... r_vec
... p_vec ... q_vec ... z_vec

cudaDeviceSynchronize()

cusparseCreate...Vec(b_vec)
... x_vec ... r_vec ... p_vec
... q_vec ... z_vec
cusparseCreateCsr(A_matrix)
cusparseCreateCsr(M_matrix)

cudaDeviceSynchronize()

MEM OMP WALL TIME END

CuCG(A_matrix, M_matrix, b_vec, x_
```

```
CCG and CudaCG ( A, M, b, x max_iter, tolerance, ...
{
  r = A * x
  r = b - r
  z = m * r      (triangular solve)
  CG OMP WALL TIME START

  while iter < max_iter and ratio > tolerance
     iter = iter + 1
     FAULT OMP WALL TIME START
     If iter % n is 0, fault_check(x)
     FAULT OMP WALL TIME END
     z = m * r          ( triangular solve )
     Rho = r[j] * z[j]
     if itert == 1
        p = z
     else
        beta = Rho / (v + Tiny)
        p = z + (beta * p)
     q = A * p
     Rtmp = p * q
     v = r * z
     alpha = Rho / (Rtmp)
     x = x + (alpha * p)
     r = r - (alpha * q)
     res_norm = norm(n, r)
     ratio = res_norm / init_norm
     r = A * x
     r = b - r

  CG OMP WALL TIME END
  return
}
```

Device	Matrix	Precision	Iteration	Wall Time (%)	Mem Wall Time (%)	Fault Time (%)
GPU	../../test_subjects/norm/Andrews.mtx.csr	DOUBLE	30000	95.70%	0.03%	4.30%
GPU	../../test_subjects/norm/Dubcova3.mtx.csr	DOUBLE	171	93.09%	5.77%	6.91%
GPU	../../test_subjects/norm/boneS01.mtx.csr	DOUBLE	2483	92.42%	0.62%	7.58%
GPU	../../test_subjects/norm/cfd1.mtx.csr	DOUBLE	1743	93.82%	0.72%	6.18%
GPU	../../test_subjects/norm/cfd2.mtx.csr	DOUBLE	7411	92.78%	0.16%	7.22%
GPU	../../test_subjects/norm/consph.mtx.csr	DOUBLE	14798	92.50%	0.12%	7.50%
GPU	../../test_subjects/norm/crankseg_1.mtx.csr	DOUBLE	3897	91.58%	0.48%	8.42%
GPU	../../test_subjects/norm/cvxbqp1.mtx.csr	DOUBLE	9275	95.65%	0.07%	4.35%
GPU	../../test_subjects/norm/finan512.mtx.csr	DOUBLE	41	94.79%	19.42%	5.21%
GPU	../../test_subjects/norm/hood.mtx.csr	DOUBLE	27313	91.64%	0.06%	8.36%
GPU	../../test_subjects/norm/nasasrb.mtx.csr	DOUBLE	30000	93.55%	0.05%	6.45%
GPU	../../test_subjects/norm/offshore.mtx.csr	DOUBLE	30000	92.77%	0.04%	7.23%
GPU	../../test_subjects/norm/qa8fm.mtx.csr	DOUBLE	58	94.15%	17.20%	5.85%
GPU	../../test_subjects/norm/s3dkq4m2.mtx.csr	DOUBLE	30000	92.78%	0.05%	7.22%
GPU	../../test_subjects/norm/ship_003.mtx.csr	DOUBLE	30000	92.77%	0.05%	7.23%
GPU	../../test_subjects/norm/shipsec1.mtx.csr	DOUBLE	30000	92.83%	0.04%	7.17%
GPU	../../test_subjects/norm/shipsec5.mtx.csr	DOUBLE	30000	92.55%	0.05%	7.45%
GPU	../../test_subjects/norm/shipsec8.mtx.csr	DOUBLE	30000	92.89%	0.04%	7.11%
GPU	../../test_subjects/norm/thermal1.mtx.csr	DOUBLE	1391	94.61%	0.53%	5.39%
GPU	../../test_subjects/norm/thermomech_dM.mtx.csr	DOUBLE	87	93.37%	10.33%	6.63%



Iteration Comparison of CPU and GPU
CPU WALL
GPU MEM
Iterations
0
200
400
600
800
1000
../test_subjects/norm/bcsstk05.mtx.csr
../test_subjects/norm/bcsstk06.mtx.csr
../test_subjects/norm/bcsstk09.mtx.csr
../test_subjects/norm/bcsstk12.mtx.csr
../test_subjects/norm/bcsstk03.mtx.csr
../test_subjects/norm/bcsstk08.mtx.csr
../test_subjects/norm/bcsstk10.mtx.csr
../test_subjects/norm/bcsstk14.mtx.csr
../test_subjects/norm/bcsstk01.mtx.csr
../test_subjects/norm/bcsstk07.mtx.csr
../test_subjects/norm/bcsstk04.mtx.csr
../test_subjects/norm/bcsstk13.mtx.csr
../test_subjects/norm/bcsstk02.mtx.csr
Matrix


Comparison of Wall Time for CPU and GPU
CPU WALL
GPU MEM
GPU FAULT
GPU WALL
Wall Time
10^3
10^2
10^1
10^0
10^{-1}
m/bcsstk01.mtx.csr
m/bcsstk09.mtx.csr
m/bcsstk13.mtx.csr
m/bcsstk14.mtx.csr
m/bcsstk02.mtx.csr
m/bcsstk06.mtx.csr
m/bcsstk08.mtx.csr
m/bcsstk10.mtx.csr
m/bcsstk12.mtx.csr
m/bcsstk05.mtx.csr
m/bcsstk03.mtx.csr
m/bcsstk07.mtx.csr
m/bcsstk04.mtx.csr

Sources

Manu Shantharam, Sowmyalatha Srinivasmurthy, and Padma Raghavan. 2012. Fault tolerant preconditioned conjugate gradient for sparse linear system solution. In Proceedings of the 26th ACM international conference on Supercomputing (ICS '12). Association for Computing Machinery, New York, NY, USA, 69–78