High Performance Computing : Home Work 4

SACHIN SATISH BHARADWAJ
N16360220 (ssb638)
https://github.com/SachinSBharadwaj/hw4.git

April 19th, 2020

Question 1
Matrix Vector Operations on a GPU

PLEASE NOTE SOME OBSERVATIONS:

1. The code seems to run mainly on cuda{2-3}.cims.nyu edu, though there are certainly differences in performances.
2. The code while running on cuda{2-3}.cims.nyu.edu , if there are other jobs running, the code runs with a lower memory bandwidth compared to CPU, else GPU has higher memory bandwidth.
3. The code either doesn't run at all or gives erroneous results on cuda1, cuda4 and cuda5.cims.nyu.edu.
4. Though I tried changing the compilation flags and some other tweaks, it doesn't seem to work.
5. Thus, results on cuda{2-3}.cims.nyu.edu are relatively more trust worthy and functional as far as my code is concerned. Hence, will report here, the results of cuda2 and 3 servers.

(1) VECTOR-VECTOR INNER PRODUCT:

The first code, **gpuq1_bt.cu** performs the vector-vector inner product using GPU + OpenMP. The following reports the performance on the CIMS' cuda{2-3}.cims.nyu.edu GPU machines:

(1) **cuda2.cims.nyu.edu** :

CPU Bandwidth = 5.223871 GB/s
GPU Bandwidth = 12.940027 GB/s
Error = 0.000000

(2) **cuda3.cims.nyu.edu** :

CPU Bandwidth = 1.442759 GB/s
GPU Bandwidth = 2.208052 GB/s
Error = 0.000000

(2) MATRIX-VECTOR INNER PRODUCT:

The second code, **gpuq1_matvec2.cu** performs the vector-vector inner product using GPU + OpenMP. The following reports the performance on the CIMS' cuda{2-3}.cims.nyu.edu GPU machines:

(1) **cuda2.cims.nyu.edu** :

CPU Bandwidth = 5.446422 GB/s
GPU Bandwidth = 26.174479 GB/s
Error = 0.000000

(2) **cuda3.cims.nyu.edu** :

CPU Bandwidth = 1.864616 GB/s
GPU Bandwidth = 4.003667 GB/s
Error = 0.000000

Question 2
2D Jacobi Method on a GPU

The third code, **gpuq2_jc.cu** performs the 2D Jacobi calculation using GPU + OpenMP. The GPU code for higher matrix sizes performs extremely well and faster on cuda3.cims.nyu.edu than the CPU(compared to cuda2) , almost by an order of magnitude. The following reports the performance on the CIMS' cuda{2-3}.cims.nyu.edu GPU machines:

(1) **cuda2.cims.nyu.edu** :

DIMENSION	TIME
8	0.166211
16	0.089171
24	0.441556
32	1.428617
40	4.619995
48	12.988716
56	31.308242
64	69.749887
72	139.065301
80	215.530283

(2) **cuda3.cims.nyu.edu** :

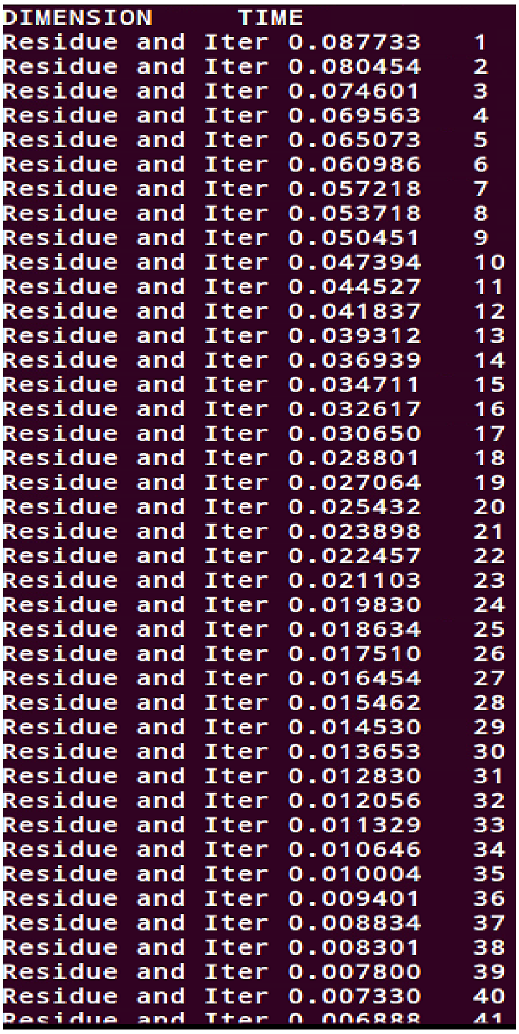
DIMENSION	TIME
8	0.336216
16	0.077725
24	0.186406
32	0.481197
40	1.140198
48	2.609419
56	5.672767
64	11.696975
72	22.739561
80	33.991669

Now in order to make sure that the GPU implementation is correct, for illustration, given a problem of dimension N=8, I compare the residues at each iteration step both on CPU and GPU and it turns out they match perfectly, demonstrating the GPU implementation is correct as shown in Figure 1 and 2. (residues vs iteration #):

Question 3
Final Project Update

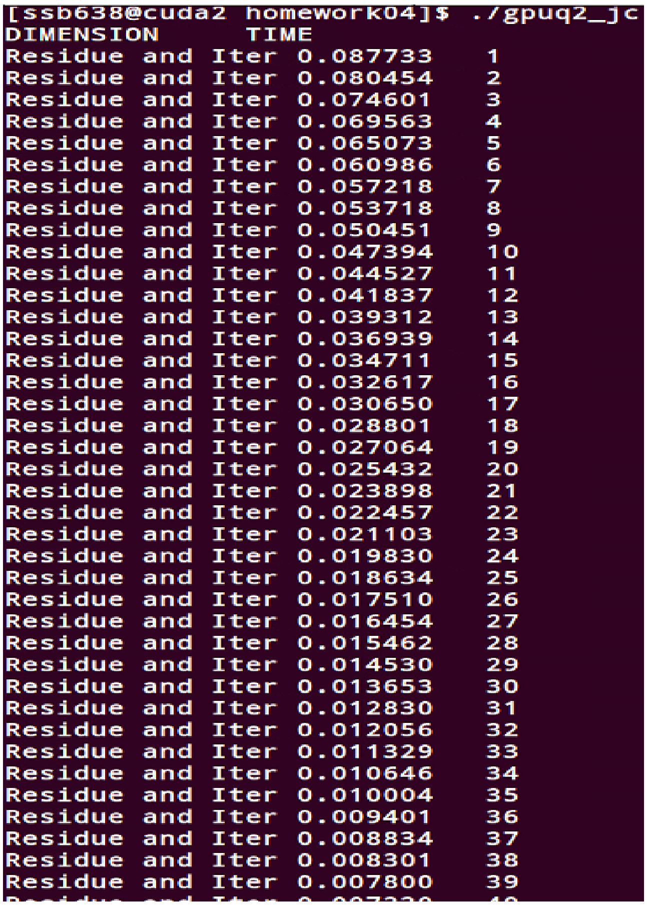
As far the final project is concerned, the following developments have taken place so far:

1. Developed the discretisation method for the Stokes Flow for 2D/1D.
2. Decided and developed the Ax=B matrix system required for the solving a Stokes flow problem.
3. Writing currently the Jacobi method for these matrices.
4. Will be doing the theoretical integration and vectorisation next.



```
DIMENSION      TIME
Residue and Iter 0.087733    1
Residue and Iter 0.080454    2
Residue and Iter 0.074601    3
Residue and Iter 0.069563    4
Residue and Iter 0.065073    5
Residue and Iter 0.060986    6
Residue and Iter 0.057218    7
Residue and Iter 0.053718    8
Residue and Iter 0.050451    9
Residue and Iter 0.047394    10
Residue and Iter 0.044527    11
Residue and Iter 0.041837    12
Residue and Iter 0.039312    13
Residue and Iter 0.036939    14
Residue and Iter 0.034711    15
Residue and Iter 0.032617    16
Residue and Iter 0.030650    17
Residue and Iter 0.028801    18
Residue and Iter 0.027064    19
Residue and Iter 0.025432    20
Residue and Iter 0.023898    21
Residue and Iter 0.022457    22
Residue and Iter 0.021103    23
Residue and Iter 0.019830    24
Residue and Iter 0.018634    25
Residue and Iter 0.017510    26
Residue and Iter 0.016454    27
Residue and Iter 0.015462    28
Residue and Iter 0.014530    29
Residue and Iter 0.013653    30
Residue and Iter 0.012830    31
Residue and Iter 0.012056    32
Residue and Iter 0.011329    33
Residue and Iter 0.010646    34
Residue and Iter 0.010004    35
Residue and Iter 0.009401    36
Residue and Iter 0.008834    37
Residue and Iter 0.008301    38
Residue and Iter 0.007800    39
Residue and Iter 0.007330    40
Residue and Iter 0.006888    41
```

Figure 1: **On the CPU, N=8**



```
[ssb638@cuda2 homework04]$ ./gpuq2_jc
DIMENSION      TIME
Residue and Iter 0.087733    1
Residue and Iter 0.080454    2
Residue and Iter 0.074601    3
Residue and Iter 0.069563    4
Residue and Iter 0.065073    5
Residue and Iter 0.060986    6
Residue and Iter 0.057218    7
Residue and Iter 0.053718    8
Residue and Iter 0.050451    9
Residue and Iter 0.047394    10
Residue and Iter 0.044527    11
Residue and Iter 0.041837    12
Residue and Iter 0.039312    13
Residue and Iter 0.036939    14
Residue and Iter 0.034711    15
Residue and Iter 0.032617    16
Residue and Iter 0.030650    17
Residue and Iter 0.028801    18
Residue and Iter 0.027064    19
Residue and Iter 0.025432    20
Residue and Iter 0.023898    21
Residue and Iter 0.022457    22
Residue and Iter 0.021103    23
Residue and Iter 0.019830    24
Residue and Iter 0.018634    25
Residue and Iter 0.017510    26
Residue and Iter 0.016454    27
Residue and Iter 0.015462    28
Residue and Iter 0.014530    29
Residue and Iter 0.013653    30
Residue and Iter 0.012830    31
Residue and Iter 0.012056    32
Residue and Iter 0.011329    33
Residue and Iter 0.010646    34
Residue and Iter 0.010004    35
Residue and Iter 0.009401    36
Residue and Iter 0.008834    37
Residue and Iter 0.008301    38
Residue and Iter 0.007800    39
Residue and Iter 0.007330    40
```

Figure 2: **On the GPU, N=8**