PARALLEL SYSTEMS PROJECT

CSC 548 HW6, under guidance of Dr. Frank Mueller

▪ Kaustubh Prabhu (ksprabhu) ▪ Narayanan Subramanian (nsubram) ▪ Ritesh Anand (ranand)

Assessing the benefits of CUDA accelerator on AMG2006 Benchmark of ASC Sequoia

Summary

This work attempts to assess the speedup obtained by employing parallelism capabilities of general purpose graphic processing unit in NVIDIA graphics cards, using CUDA, on AMG2006 which is a MPI C code parallel algebraic multi-grid solver for linear systems arising from problems on unstructured grids.

In our analysis we also found that AMG2006 is memory-access bound, doing only about 1-2 computations per memory access, so memory-access speeds will have a large impact on performance. This observation is also made in the *read me* of the benchmark. Knowing that this hardware accelerator in *GeForce GTX 280* is tuned towards data-parallel compute extensive tasks, like in graphics rendering, we do not expect this to be extracting much speedup on this benchmark. We expect it to actually show a fractional speedup.

The main reasons are -

1. Huge data structures (matrices, vectors, etc.) are to be worked upon by the Kernel.
2. Due to the size of data structure we cannot leverage spatial locality optimally as size of cached *constant* and *texture* memory supported is rather limited.
3. The accessing pattern of data structures is very irregular. Array indices are often stored as values in some other array, hence the strides in access very unpredictable. E.g. one of the array access inside kernel is a[b[i]] where a[i] and b[i] are both device arrays.
4. We fall back on global memory access, which are time expensive and on top of it, computation to I/O ratio is very unfavorably biased.

However, keeping in mind these limitations we have achieved –

1. Integration of CUDA kernel and AMG2006 application
2. Speedup in function execution time, which involves the computation kernel
3. A method which can be applied to port any computational kernel on CUDA

For this, we have used *GeForce GTX 280 NVIDIA* graphics cards available on the *osXX* machines having *AMD Athlon XP* processors. We have been able to completely port the benchmarking applications on CUDA. One of the identified compute intensive kernels has been successfully ported on CUDA. Currently, we can observe speedup in function execution (through gprof).

Solved Issues

Since our last reported progress, in which we had correctly identified the main performance bottlenecks which could be optimized, following are the main solved issues:

1. Compiling AMG2006 on CUDA cluster
2. Porting whole application to CUDA
3. Porting compute intensive performance bottlenecks to CUDA kernels
4. Running AMG2006 applications with CUDA (limited semantics)
5. Optimizing CUDA code for lesser problem size

Relevant details for these solved issues are explained in the ***detailed analysis*** section.

Results

This section presents the data comparison between function execution time with normal AMG2006 and AMG2006 with CUDA.

The following table represents the head to head execution time comparison with and without CUDA kernel execution. The concerned function is "*hypre_BoomerAMGRelax*".

Command	MPI nodes	% execution time and self call time - Normal	% execution time and self call time – With CUDA
mpirun -np 4 -machinefile hosts amg2006 -P 2 2 1 -r 2 1 1	4	**12.50%**	**Function not spotted**
mpirun -np 4 -machinefile hosts amg2006 -P 2 2 1 -r 2 2 1	4	**12.50%**	**Function not spotted**
mpirun -np 8 -machinefile hosts amg2006 -P 2 2 2	8	**13.04%, 0.04 sec**	**4.76%**
mpirun -np 8 -machinefile hosts amg2006 -P 2 2 2 -r 2 1 1	8	**15.00%, 0.04 sec**	**2.50%**
mpirun -np 8 -machinefile hosts amg2006 -P 2 2 2 -r 3 1 1	8	**14.29%, 0.06 sec**	**1.75%**

Key:

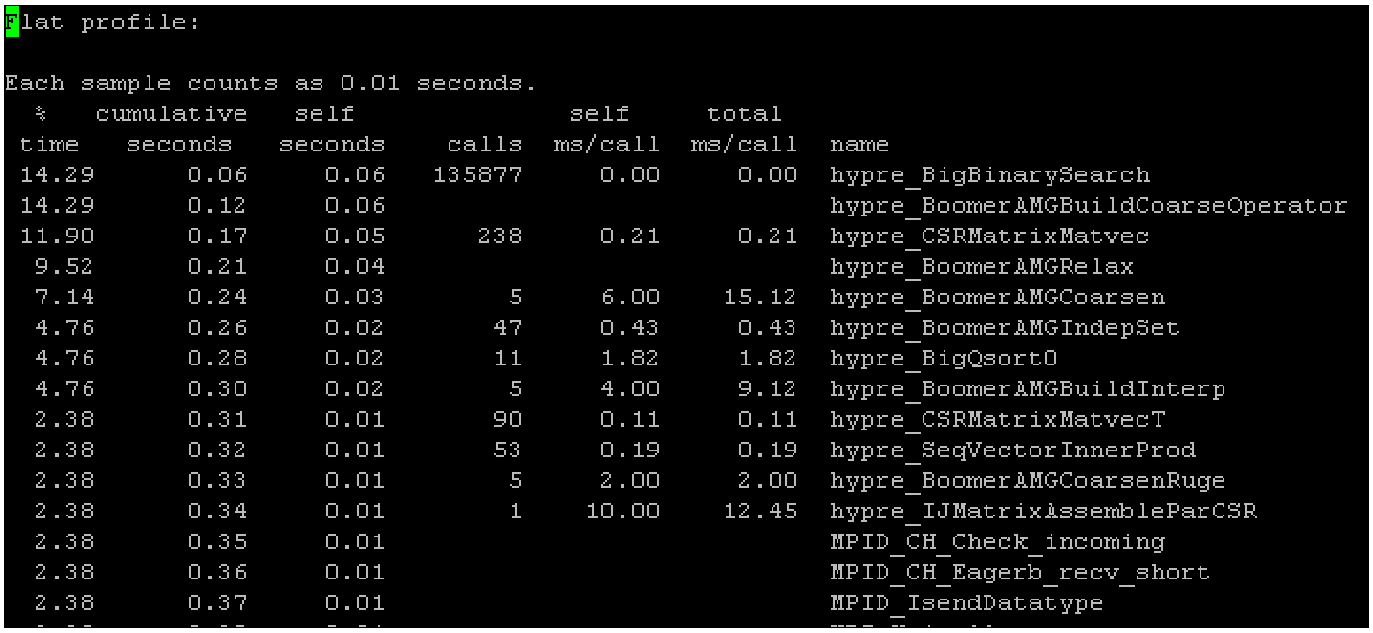
Command – these are the input parameters associated with the AMG2006 execution.

MPI nodes – represents number of nodes communicating.

% execution time and self time - % execution time is the percentage of total execution time. Self call time is the absolute time taken by the function.

Note - The varying command line argument (-r) represents the problem size.

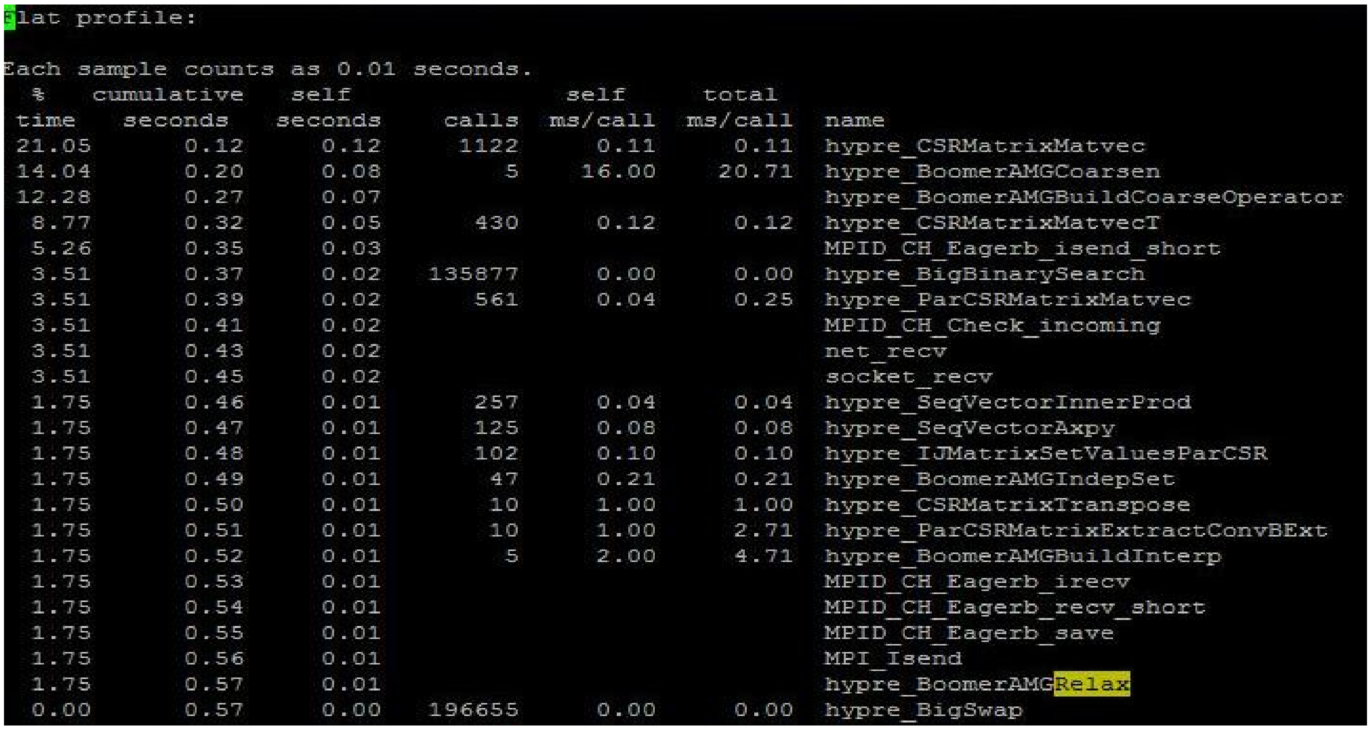
Sample Gprof Output -



```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 14.29      0.06     0.06   135877     0.00     0.00  hypre_BigBinarySearch
 14.29      0.12     0.06                             hypre_BoomerAMGBuildCoarseOperator
 11.90      0.17     0.05      238     0.21     0.21  hypre_CSRMatrixMatvec
  9.52      0.21     0.04                             hypre_BoomerAMGRelax
  7.14      0.24     0.03        5     6.00    15.12  hypre_BoomerAMGCoarsen
  4.76      0.26     0.02       47     0.43     0.43  hypre_BoomerAMGIndepSet
  4.76      0.28     0.02       11     1.82     1.82  hypre_BigQsort0
  4.76      0.30     0.02        5     4.00     9.12  hypre_BoomerAMGBuildInterp
  2.38      0.31     0.01       90     0.11     0.11  hypre_CSRMatrixMatvecT
  2.38      0.32     0.01       53     0.19     0.19  hypre_SeqVectorInnerProd
  2.38      0.33     0.01        5     2.00     2.00  hypre_BoomerAMGCoarsenRuge
  2.38      0.34     0.01        1    10.00    12.45  hypre_IJMatrixAssembleParCSR
  2.38      0.35     0.01                             MPID_CH_Check_incoming
  2.38      0.36     0.01                             MPID_CH_Eagerb_recv_short
  2.38      0.37     0.01                             MPID_IsendDatatype
```

Figure 1- Execution of AMG2006 for 8 nodes. "Hypre_BoomerAMGRelax" takes 9.52% of total execution time.



```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 21.05      0.12     0.12     1122     0.11     0.11  hypre_CSRMatrixMatvec
 14.04      0.20     0.08        5    16.00    20.71  hypre_BoomerAMGCoarsen
 12.28      0.27     0.07                             hypre_BoomerAMGBuildCoarseOperator
  8.77      0.32     0.05      430     0.12     0.12  hypre_CSRMatrixMatvecT
  5.26      0.35     0.03                             MPID_CH_Eagerb_isend_short
  3.51      0.37     0.02   135877     0.00     0.00  hypre_BigBinarySearch
  3.51      0.39     0.02      561     0.04     0.25  hypre_ParCSRMatrixMatvec
  3.51      0.41     0.02                             MPID_CH_Check_incoming
  3.51      0.43     0.02                             net_recv
  3.51      0.45     0.02                             socket_recv
  1.75      0.46     0.01      257     0.04     0.04  hypre_SeqVectorInnerProd
  1.75      0.47     0.01      125     0.08     0.08  hypre_SeqVectorAxpy
  1.75      0.48     0.01      102     0.10     0.10  hypre_IJMatrixSetValuesParCSR
  1.75      0.49     0.01       47     0.21     0.21  hypre_BoomerAMGIndepSet
  1.75      0.50     0.01       10     1.00     1.00  hypre_CSRMatrixTranspose
  1.75      0.51     0.01       10     1.00     2.71  hypre_ParCSRMatrixExtractConvBExt
  1.75      0.52     0.01        5     2.00     4.71  hypre_BoomerAMGBuildInterp
  1.75      0.53     0.01                             MPID_CH_Eagerb_irecv
  1.75      0.54     0.01                             MPID_CH_Eagerb_recv_short
  1.75      0.55     0.01                             MPID_CH_Eagerb_save
  1.75      0.56     0.01                             MPI_Isend
  1.75      0.57     0.01                             hypre_BoomerAMGRelax
  0.00      0.57     0.00   196655     0.00     0.00  hypre_BigSwap
```

Figure 2- Execution of AMG2006 with CUDA for 8 nodes. "Hypre_BoomerAMGRelax" takes 1.75% of total execution time

Detailed Analysis

We began the work from performance analysis of AMG2006 on regular *AMD Athlon XP* cluster. We could find the performance bottlenecks and hence the targets for optimization. This is aptly captured in our previous reports and on our project web page.

Below we describe the major tasks undertaken and how we solved it, with issue faced if any.

1 Compiling:

This was our first r0ad block. After 2 weeks of almost continuous effort we were able to successfully compile AMG2006 applications with CUDA.

Problems Faced:

Compiling the AMG2006 application involved multiple Makefiles. Integration of application specific makefiles and CUDA makefiles was main blockade. As our code was creating a custom library file, integration of makefile was difficult. The CUDA programming structure doesn't accept relative path for included library files.

What worked:

nvcc compiler provided by CUDA. nvcc allowed us to create object files for custom libraries.

2 Porting whole application:

Our approach to port to CUDA was:

(1) Identifying kernel regions (code sections to be executed on the GPU)

(2) Extracting kernel regions and transforming them into CUDA kernel functions, and

(3) Analyzing shared data that will be accessed by the GPU and inserting necessary memory transfer calls.

Problems Faced:

Identification of computational kernels which can be successfully ported to CUDA. Many of the computational kernels involved custom library calls like (HYPRE_CTAlloc) which can not be ported to CUDA. Some of the hot spots had to be ignored because of this device specific constraint.

What worked: Any loops with multiple entry-exit points were rejected. Any compute intensive code with a custom library call was rejected. The elimination method proved useful.

3 Porting Kernels:

Problems Faced:

A) Primary concern was memory constraints while creating device specific data structures. As the problem size of application increased, CUDA memory was insufficient.

B) The application accesses non uniform array patterns. This involves heavy usage of array[array[offset]] pattern. This creates limitation on the memory access.

C) The size of device specific arrays is hard to find. Furthermore, due to complex structure of the code, it was difficult to access the size of device specific arrays. This is still the Major obstacle.

What worked:

A) We limited our scope to optimize the performance of AMG2006 for reduced problem size.

B) Overprovision of memory is a simple approach we have used in this code. Another possible approach is to access the relevant code section, where the arrays are getting populated and return the size. This will involve changing the code.

4 Optimization:

By optimization we mean the performance improvement in execution time of the compute intensive kernel, and respective improvement in function execution time. We have observed that the function execution time reduces drastically. CUDA computes the kernel much faster but also involves copying of huge data structures.

Problems Faced: The memory constraints on the device specific data structures did not allow us to exploit CUDA kernels for large problem size.

We were able to solve the issue only to certain extent, given the scope of the project.

Conclusion

We have observed a performance improvement in function execution time when the application AMG2006 was ported to CUDA. Due to memory constraints we could not exploit this at higher problem size. Hence the observed speedup is limited for less problem size and relatively fewer numbers of nodes.

Future Work

Future work involves addition of more computational kernels in AMG2006 application, which were identified in earlier report. Also a scalable approach while accessing device specific data structures is required.

Individual Contributions

Compilation, Overprovisioning: Narayn.

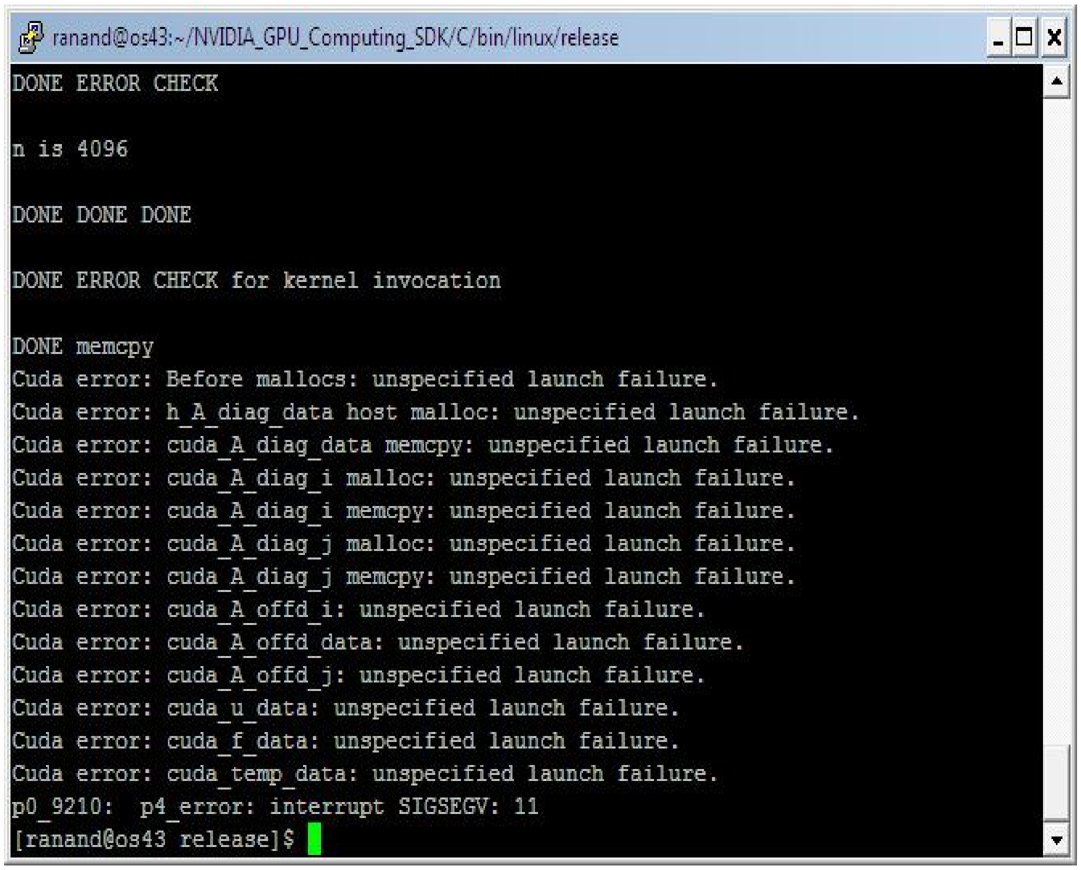
Hyper library, data sizes: Kaustubh.

Optimization issues for data transfer: Ritesh.

References

1. http://www.nvidia.com/object/product_geforce_gtx_280_us.html
2. http://classic.chem.msu.su/gran/gamess/gtx280.html

Appendix



ranand@os43:~/NVIDIA_GPU_Computing_SDK/C/bin/linux/release

```
DONE ERROR CHECK

n is 4096

DONE DONE DONE

DONE ERROR CHECK for kernel invocation

DONE memcpy
Cuda error: Before mallocs: unspecified launch failure.
Cuda error: h_A_diag_data host malloc: unspecified launch failure.
Cuda error: cuda_A_diag_data memcpy: unspecified launch failure.
Cuda error: cuda_A_diag_i malloc: unspecified launch failure.
Cuda error: cuda_A_diag_i memcpy: unspecified launch failure.
Cuda error: cuda_A_diag_j malloc: unspecified launch failure.
Cuda error: cuda_A_diag_j memcpy: unspecified launch failure.
Cuda error: cuda_A_offd_i: unspecified launch failure.
Cuda error: cuda_A_offd_data: unspecified launch failure.
Cuda error: cuda_A_offd_j: unspecified launch failure.
Cuda error: cuda_u_data: unspecified launch failure.
Cuda error: cuda_f_data: unspecified launch failure.
Cuda error: cuda_temp_data: unspecified launch failure.
p0_9210:  p4_error: interrupt SIGSEGV: 11
[ranand@os43 release]$
```

Figure 3 – Errors in accessing device data structures