PARALLEL SYSTEMS PROJECT

CSC 548 HW5, under Dr. Frank Mueller

Submitted by: Kaustubh Prabhu (ksprabhu), Narayanan Subramanian (nsubram), Ritesh Anand (ranand).

Solved Issues:

1. Identification of target loop constructs to be optimized by porting the inner-most functionalities as CUDA kernels.
2. The task 1 has been done for three major bottleneck functions that were consistently taking major time across the four major applications in the benchmark.
3. Resolved many linking errors on attempting to port the benchmark application on to CUDA.

Detailed Analysis

- We initially realized that the profiling tool gprof was not giving consistent results for different runs. We realized that there wasn't enough computation for 64 nodes in parallel. So, we tried to get an overview of the application, mainly how to increase the problem size.

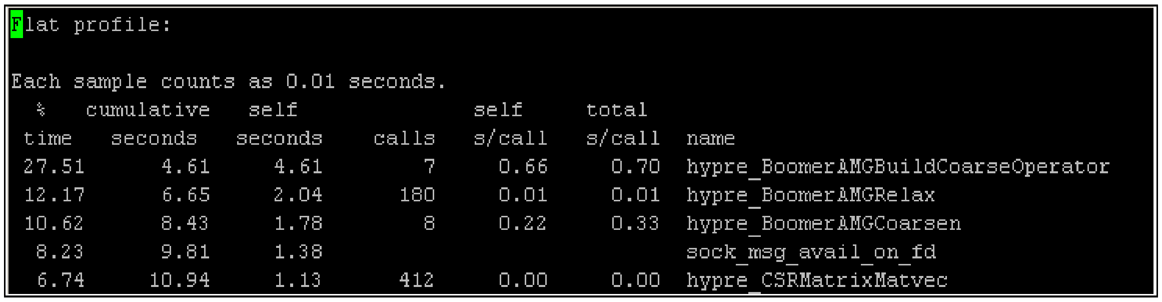
  We then found about the –r option which increases the problem size per processor.

  Running the application for larger problem sizes, we could now match the hotspots with the ones we had submitted in Report 1.

- Sample output from the profiler (gprof)

  Command line execution -

  mpirun –np 64 –machinefile hosts amg2006 –P 4 4 4 [-r 6 6 6] -laplace



```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 27.51      4.61     4.61        7     0.66     0.70  hypre_BoomerAMGBuildCoarseOperator
 12.17      6.65     2.04      180     0.01     0.01  hypre_BoomerAMGRelax
 10.62      8.43     1.78        8     0.22     0.33  hypre_BoomerAMGCoarsen
  8.23      9.81     1.38                             sock_msg_avail_on_fd
  6.74     10.94     1.13      412     0.00     0.00  hypre_CSRMatrixMatvec
```

Fig 1.1 gprof output

- Then we analyzed the following functions:
  1.) hypre_BoomerAMGBuildCoarseOperator
  2.) hypre_BoomerAMGRelax
  3.) hypre_BoomerAMGCoarsen

  Which have nested *for* loops are the taking longer time duration in every processing element. We analyzed the loops further using *printfs* and counters and MPI_Wtime () to see how much the nested for loops contributed to the total function time.

We identified performance bottleneck from each of these functions are described below –

hypre_BoomerAMGBuildCoarseOperator (AMG2006/parcsr_ls/par_rap.c)

Line No.	**Type of Loop**	**Nested**	**Level of nesting**	**Remarks**
541- 741	for	Yes	5	Type1
768-1021	for	Yes	5	Type1
1148- 1376	for	Yes	5	Type1
1427-1747	for	Yes	5	Type1

"Type": We are using this notation to classify *for* loops based on structure.

This function contains multiple bottlenecks. Each of the "for" loops is a 5-level nested for loop, involving computation on different arrays. These for loops are of a single type.

By type, we mean the context of "for" loop. The internal structure of all these loops is similar. Hence the performance improvement amongst all of them should be consistent.

. *hypre_BoomerAMGRelax (AMG2006/parcsr_ls/par_relax.c)*

Line No.	**Type of Loop**	**Nested**	**Level of nesting**	**Remarks**
189-211	For	Yes	2	Type 1
223-247	For	Yes	2	Type 1
303-324	For	Yes	2	Type 1
336-359	For	Yes	2	Type 1
425-464	For	Yes	3	Type 2
470-491	For	Yes	2	Type 1
511-552	For	Yes	3	Type 2
607-655	For	Yes	3	Type 2

As we can see, we have 2 types of *for* loops repeating at several places in this function. We are in a process to analyze all such loops by their category. Since *for* loops of a similar kind can be ported in a similar form of CUDA kernel, the repetition can be exploited.

hypre_BoomerAMGCoarsen (AMG2006/parcsr_ls/par_coursen.c)

Line No.	**Type of Loop**	**Nested**	**Level of nesting**	**Remarks**
628-827	For	Yes	3	Single bottleneck

As we can see, this function contains a single nested for loop, which is identified as the computational kernel for CUDA.

- We inferred that at the innermost *for* loop, not significant computation is done. Rather, data is being structured through these for loops.

- These are the same *for* loops for which *openmp* constructs were used in an effort to parallelize them. We thus realized that, we need to port these for loops to CUDA to get performance improvement.

<u>Compiling AMG2006 on CUDA</u>

- In an effort to start things on CUDA, we tried to compile the application through CUDA.
  We faced numerous problems because of existent directory structures and header file inclusions. We are in the process of modifying the make files and linking all the generated AMG specific libraries to somehow run the application.

- Once this is done, we will have a solid platform to build our CUDA code.

<u>Milestones Achieved:</u>

- Analysis of each function in detail – doing a code walkthrough to understand what each function does.

  Each member analyzed 2 *for* loops.

- Compiling the application using CUDA.(This activity is 75% complete)
  Currently we are not able to include the libraries to be linked. The CUDA build and the AMG build together is a very complex structure and hence we are spending more time on this to get the things resolved.

<u>Recent Concerns:</u>

- The nested *for* loops operate on many arrays which need to be copied to the CUDA device from the host. We fear that though we will optimize the computation involved, time will be spent on moving the data in and out of device.

- While running the gprof tools, we found that a big chunk of time is spent in communication between the nodes. This is validated from the functions like net_send, p4_sockets_ready, socket_recv which appear as the hotspots in the gmon log file. These functions are a part of some library and hence we cannot optimize them.

<u>Future Work:</u>

- Compile the application using CUDA.(resolve the pending issues)
- Start porting the *for* loops into CUDA.
- Analyze the result.
- Repeat the work if needed.