Accelerating Simulation Codes through the GeMTC Framework

Digvijay Singh Gahlot[*], Scott Krieder[*], Ioan Raicu[*+]
[*]Department of Computer Science, Illinois Institute of Technology, Chicago IL, USA
[+]Mathematics and Computer Science Division, Argonne National Laboratory, Argonne IL, USA
dgahlot@hawk.iit.edu, skrieder@iit.edu, iraicu@cs.iit.edu

ABSTRACT

"GPU Computing "utilizes high level language to run sequential part of the code on the CPU as well as speeds up parallel part via running it on GPUs but GPUs are SIMD by default which means they can run only single instruction on multiple data. The introduction of GEMTC framework [1] addresses these limitations by providing an efficient middleware through which tasks are submitted to a common task queue to the device and workers (warp which represent the lowest possible level of control on device) take out the tasks, execute them and put them back on the result queue. This work explores porting and evaluation of real world applications into GEMTC framework. I choose Imogen [2] advanced astrophysical simulation tool and SciColSim [3] which simulates scientific discovery. I was able to port pure fluid kernels from Imogen and expensive functions of SciColSim to GEMTC. The evaluation resulted in performance up to 200 plus tasks/sec for kernel with moderate size data inputs. The results were compared with the CPU equivalent code and GEMTC was able to outperform CPU code for moderate size data inputs.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]; H.3.4[**Systems and Software**]

General Terms

Performance, Experimentation

Keywords

GPGPU, MTC, CUDA, Matlab, Accelerator, Swift.

1. INTRODUCTION

This work involves porting of real work applications to GEMTC framework. GEMTC provides framework to use a GPU for MTC applications. The evaluation of GEMTC earlier was done using sleep jobs. The work discusses the applications Imogen and SciColSim which were chosen to be ported on GEMTC. Imogen originally was written in Matlab and CUDA. SciColSim is a C++ code using swift for launching parallel tasks.

Imogen solves fluid dynamics and Ideal Magneto Hydro Dynamics equations using GPU. This project discusses the effort involved in porting, including the analysis, porting challenges encountered. Also discusses the problems and bugs encountered during the development cycle. The work also explore design implementation strategies and methodologies which came handy and had considerable impact on performance. It provides tips and guidance for developing new applications on GEMTC. This will answer question like what approach will work easily? What will not work? What are the shortcuts? Furthermore the work discuses performance evaluation of the ported kernels in GEMTC. Plots and analyses of the same has been discussed in detail giving focus on similarities and dissimilarities found between different kernels. This work also coded C equivalent host code for all the ported kernels of Imogen as original code is all in MATLAB. Also benchmarking code was developed for performance evaluation. During the development cycle memory bugs in the GEMTC framework were also encountered and a basic code to quickly trigger this error has been discussed in this paper.

The discussion on porting effort for SciColSim is discussed and here the focus is on the challenges for porting. I have provided the reason that why porting the complete application itself was not feasible. I have also discussed the various approaches attempted for migration and what I was able to achieve for SciColSim. This application involved lot of tool-chaining effort all of which has been discussed and also detailed understanding of compiling shared libraries and combining them together. I have also discussed why Imogen is different from SciColSim and why a design of porting used for Imogen doesn't work for SciColSim.

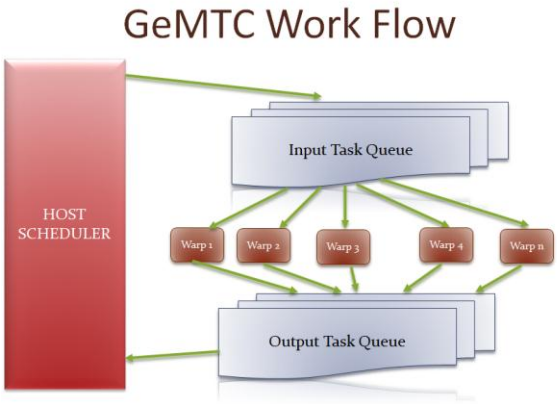
Discussion on future work gives insight on what more can be done. I also identified some applications which will show very high efficiency in GEMTC. Reason for why they will show high efficiency on GEMTC has been provided. I have also discussed the thought process for choosing an application to be ported be ported on GEMTC, this can be used as reference in future porting efforts.

2. BACKGROUND

The GPUs started as specific fixed function pipelining in their early era. With the bent of exploiting usage of GPU for scientific applications

they went enhancement to include support for floating point operations and were termed General Purpose GPUs. The main hurdle in extensive usage of GPGPUs was their programmability. Hence programming model to extend C for data-parallel constructs was adopted and CUDA was born. It is a parallel computing platform and programming model enabling increase in computing performance by using GPUs. NVIDIA claims acceleration of two order in magnitude for data parallel applications. It provides and abstraction and hence micro-kernels/application need not to be re-written with change in GPU architecture. The transfer rates between GPU and CPU have been claimed to be 6 GB/sec on PCI 2.0. All NVIDIA GPUs support 32 bit integer and single precision floating point arithmetic. NVIDIA GeForce 6 series support MIMD branching in vertex processors. GEMTC framework has been designed to utilize GPU as a MIMD resource by enabling different warps to run different micro-kernels. It accomplishes this by running a superkernel on the device. This kernel and CPU component communicate via shared memory on GPU. Device maintains two types of queues in it, one is the task queue on which the host submits task to device and the other is the result queue on which the results are placed by warps (workers) once the task computation is complete. Each task has a task description with the help of which the information about which kernel to be executed is communicated to the warp. Hence forming a MIMD architecture.

The major components included in GEMTC are task descriptions which is responsible for encoding key information for MTC workloads on host and device. TaskID in task descriptor differentiates one task from another. Tasktype refers to pre-complied micro-kernel. The author has mentioned a big list of microkernel GEMTC supports and numThreads indicates the number of threads of worker which will be requires to accomplish the task. GEMTC allows working at wrap level which is the lowest possible level of control on the device. It is these separate wraps which gives the MIMD behavior to the device. To improve malloc's and reduce the cost of allocating device memory GEMTC framework also includes a sub-allocator [10] which uses the existing CUDA malloc to allocate large contiguous pieces of memory, allocating more as needed. The pointers of these free chunks and their sizes are stored in a circular linked list on the CPU. This list is ordered by increasing device address to allow easy memory coalescing. The performance of the sub-allocator has been compared to CUDA and the sub-allocator shows efficient behavior with execution time on same order of magnitude as memory transfer to/from device. Comparison has been made of sleep task with varying time and it was notices that GEMTC achieves very high efficiency with tasks of size > 5000 micro seconds.



3. MOTIVATION

GEMTC framework [1] evaluated all its performance and efficiency using sleep tasks but to the is strong reason to test and benchmark results for real work applications. This is because sleep job don't perform any computation, it too simple in the sense that it can neither evaluate impact of heavy computation tasks on GEMTC nor evaluate impact of ***data intensive*** tasks on GEMTC. Heavy computation job involves evaluating result of a complex mathematical function, predicting probabilities etc. whereas data intensive task will involve too many read/writes for example weather forecasting using data for sensors. The real work applications will range from being totally compute intensive to totally data intensive. Also for making any framework suitable for wide acceptance firstly the framework must have solid results on real world applications and must be able to address large variety of problems falling under its category. This motivated migration of real world applications like Imogen which solves Fluid dynamics equations. Similarly SciColSim was chosen for porting because this application attempts to understand the process of discovery by modeling knowledge as graph of concepts and then tries to simulate different graph exploration strategies.

4. PROPOSED SOLUTION

The work as such consisted of migrating real world applications to GEMTC framework. Nature of project is implementation of real system. The first goal consisted of identifying a real world application for the project. Imogen was the first candidate chosen for porting. This application is an advanced astrophysical

simulation tool which uses MPI-parallel code to solve FD and MHD equations using GPUs. It uses Matlab to control management functionalities and the heavy duty processing is kept at GPU level. The core functions are compiled Mex files. The second application was SciColSim (Simulating Scientific Discovery). This is ongoing research at University of Chicago. Objective is to understand process of scientific discovery. Like "How do scientists select hypotheses to work upon", what are the most effective strategies". This can be explored with simulation by "modeling knowledge as graph of concepts. Then simulate different graph exploration strategies. There are computational characteristics associated with this application. Each simulation implemented with sequential C++ code. The application is floating point intensive, many probability calculations are involved in it. The second goal consists of writing micro-kernels for some applications from first goal. I was able to successfully write GEMTC kernels corresponding to CUDA kernels of Imogen. The first kernel ArrayAtomic solves setting array elements having value less than a certain threshold to that specific threshold, setting array elements having a value greater than certain threshold to that specific threshold, or setting array elements with values Not a Number to zero or some specified value. The second kernel ArrayRotate solves the problems of martrix transpose and also is capable of performing array exchange in Y and Z dimension for 3-D data. The third kernel FluidW calculates a first order accurate half-step of the conserved transport part of fluid equations CFD which is used as predictor input to the matching TVD function. Only pure hydro kernel was ported implying the magnetic variables are all zero. The fourth kernel freezeAndPtot is used to derive pressure and freeze parameters to enforce minimum pressure. The fifth kernel FluidTVD takes a single forward-time step, CFD of the conserved-transport part of the fluid equations using a total variation diminishing scheme to perform a non-oscillatory update. I migrated only purehyro kernel, hence magnetic parameters are all 0. For SciColSim I wrote kernel for its expensive function. The third goal consists of writing test cases for applications. The benchmarking code for all the kernels were written and tested for results. The fourth goal was comparative analysis of CPU with GPU version of the code. I tried to plot as many comparison graph marking impact of data size on performance, impact of submitting bunch of tasks together to GEMTC, time taken by single task in kernel. For all the kernels their respective CPU only code was also written. With that only the benchmarking and performance analysis was made.

There is big list of techniques which were used for this project. C/CUDA programming was the most utilized programming languages used. Apart from this Imogen required additional effort of learning Matlab code, this was because there were few Matlab library calls which were absent inside C/CUDA, I have struggled to find C equivalents and 1-2 times even implemented them. Also at the time of project development I didn't have any algorithm to start with and the comments present inside Imogen were too poor to get any idea what the kernel was solving. I will specify that the equations itself were absent on top of the kernel. There were lot of grep and search on Imogen 's base to find what actually is happening. (Just to mark it the equations were updated 11 days back only in Imogen). Hence apart from porting it was a reverse engineering effort (I guess it should also be counted as technique) and I didn't have any previous experience on Matlab. So what was done is usage of an online Matlab simulator to figure out the outcome of code at various stages, especially the unit-testing code of Imogen. The benchmarking code has been adopted from unit-testing code of Imogen. Imogen's implicit grid, block and thread logic in its CUDA kernel was broken for data processing by 1 single warp inside GEMTC. In Imogen we move in all three dimensions and data to process is implicitly determined by grids, block indices but GEMTC has only 32 threads. Another time consuming effort was on packing and unpacking of input, output parameters for submitting tasks into GEMTC and getting them back. All Imogen kernels required passing of 5-6 3-dimensional parameters, 3-4 2-dimensional parameters, and few single variables into GEMTC. This was one among the calculation intensive work. About 75% "Segmentation Fault" bugs were due to incorrect pointer arithmetic. High optimization for copying relevant parameters only back from GEMTC was accomplished. Imogen's unit testing code generated inputs itself, this inbuilt input generation code has also been written. The data provided was used for 1 kernel only "FluidTVD.cu" so for that kernel file read code was used. I also encountered memory bug inside GEMTC which was earlier thought to be a pointer arithmetic bug but I was able to re-create the bug without GEMTC kernel only via plain data copy to and fro from GEMTC. I have written a small code to re-create the issue. This is where we can clearly see that why we need to migrate and test real world application into GEMTC, sleep jobs will not encounter these bugs.

Each of ported Imogen's kernel incurred some challenges. "ArrayAtomic" kernel though looks simple had an issue, Imogen's kernel didn't have to care about processing the entire data with single warp, it was to be achieved by us with 1 warp.

Imogen's "ArrayRotate" relied on shared memory to rotate the array but if GEMTC starts using shared memory we will be limited on the maximum size array which can be rotated as only 8192 bytes are available per warp in GEMTC. As the data matrix is double the number of elements will be 1024 or in other terms 32*32 matrix of float. Imogen's "FluidW", "FluidW" and "freezeAndPtot" had lack of code comments, the unit-testing code required Matlab's library call hence had to be re-implemented in C.

The application SciColSim required even more techniques like tool-chaining. Making SciColSim itself to work on Jarvis it took 3 days due to tool-chain issues. The main was figuring out the proper set of tools with which SciColSim worked. This went as deep as finding out the date when SciColSim was developed and downloading, installing tools with version corresponding to that time period. SciColSim requires STC, Turbine, MPICH, TCL, Boost, Swift. Path variable settings (as I wasn't able to use default tools on Jarvis because SciColSim didn't work with them). After the tool-chain effort another effort was on installation of callgrind to figure out the order in which call are happening when the application is being run. Initially I attempted migrating the complete application itself in GEMTC but it failed because 1) the code piece is too larger involving C++ class. 2) The input to provided is a filename from which graph gets constructed in the constructor. 3) It involve bunch of global static variables and functions which don't belong to class itself (have to figure out where they will fit). 4) Porting will mean entirely moving the whole computation including the object creation, destruction in the GeMTC kernel. 5) There are Boost objects (Have to think about compiling and using them in GeMTC), I searched using boost in CUDA and found that there is no RTT (Run Time Type) support in CUDA. (Hence the idea of migrating the complete application to GEMTC was dropped). Also there is no Boost equivalent library with graph support in C. 6) The graph construction requires set of dynamically allocated edges, vertices, state, probability double dimensional pointers. I also noted that there is lot of sequential processing happening inside SciColSim. For example when we make a decision based on graph state it used to depend on previously computed values, hence there was a directly data dependency. This type of code can't be made parallel anyhow. So once I had the output of callgrind and notes of SciColSim it clearly brought out the most time consuming functions inside SciColSim and I implemented the corresponding GEMTC kernel for the same. Further there was more struggle to integrate the application into GEMTC due to shared library dependency. As per the work accomplished I implemented 1 warp kernel for processing and 1 thread only processing kernel for SciColSim. During the integration work it was found that SciColSim will launch multiple workers and hence my application kept crashing. I found it latter that it was due multiple calls to gemtcSetup() and the only way to work around this was to move this call inside swift code itself before parallel jobs are launched. Inspite of multiple attempts I wasn't able to make multiple worker code work it kept crashing. Hence benchmarking was done with single worker only.

As per benchmarks done for Imogen I also anticipate that even launching multiple workers will not let GEMTC beat out CPU only code implementation. This is because there is sequential code inside the application which is run by each worker along with the code I parallelized. The number workers which can be launched by swift will be dependent on the core on CPUs. Hence number of tasks inside GEMTC queue will be limited to the number of CPU cores (one-to-one correspondence with workers launched by swift) this will have direct implications on GEMTC performance. For evaluation I used Jarvis cluster and workstations with accelerators. The project required Swift/T scripting also. Software requirement include CUDA C compiler, screen/Tmux application for multiple session saving, Git repository for code development, TCL, STC, swift, turbine, mpich, matlab, boost etc.

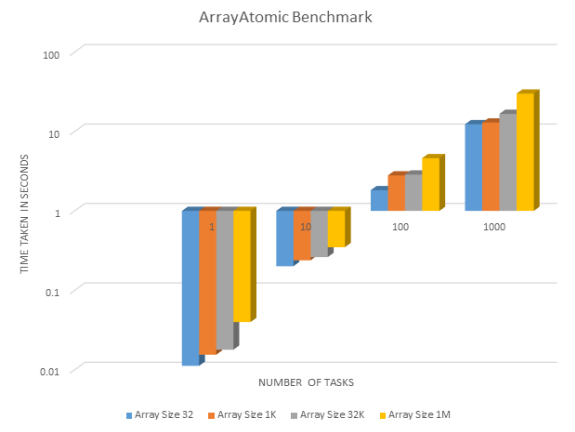
The code for Imogen is about 4500 lines with proper code comments. This has been created as separate directory under GEMTC https://github.com/skrieder-datasys/gemtc/tree/master/Tests/Imogen (the README file inside provide all details on code organization, building and executing, the input data is generated by the code itself, "FluidTVD" kernel uses the data present in the data directory).

The code for SciColSim kernels is about 600 lines and few modified files inside the SciColSim application itself. This has been created as a separate directory under GEMTC https://github.com/skrieder-datasys/gemtc/tree/master/Tests/SciColSim (the README file inside provide all details on code organization, compiling and executing).

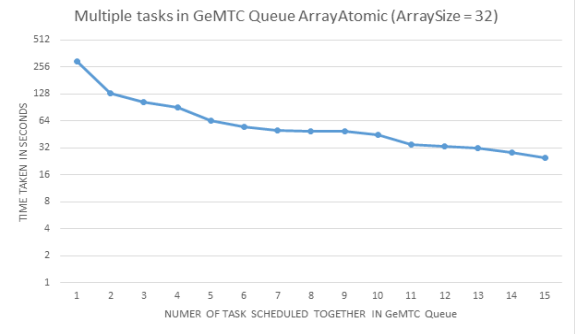
5. EVALUATION

The benchmarking for Imogen and SciColSim migrated kernels was done on per kernel basis. The tests performed plot curves depicting effect of data-size on application, impact of submitting multiple tasks together, comparison with equivalent host code on varying data sets, kernel processing time with Host processing time for CPU only application. I also

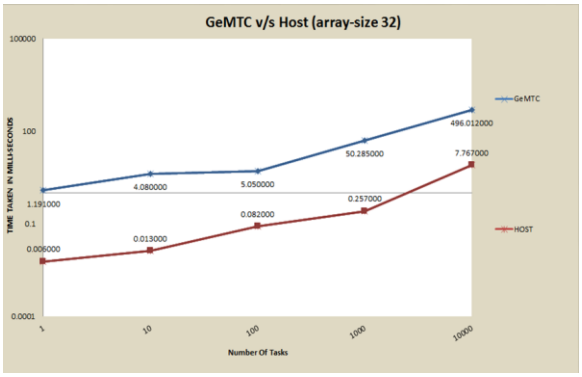
present best tasks/sec achieved by each kernel and the corresponding data set. Plot for SciColSim depicts performance on Host only code v/s GEMTC 1-thread code and GEMTC 32-thread code. The kernel and CPU processing time is also compared.



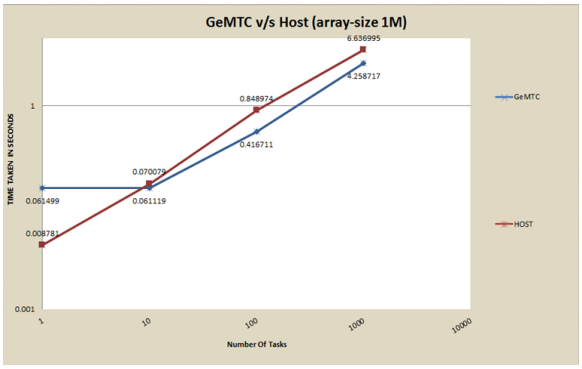
ArrayAtomic plot for varying data-size and its impact on turnaround time for completing a given number of tasks. Please note that it is for comparing data-size only hence there is only 1 task on GEMTC queue at a time. We see linear rise in time taken to complete jobs with increment in number of tasks also in each set the time taken by larger data-size array is more when compared to task with less data-size. This is all expected behavior.



Here we see the impact of submitting multiple tasks together into GEMTC queue the performance increases as the total turnaround time for completion is decrementing. The graph behaves linear then suddenly seems saturating when number of tasks reaches 12. This is because there is no control logic by which we can instruct SM to executed particular task. It is up to the warp scheduler to execute warp. There is 1 more thing, we are under-utilizing GEMTC, because at each step we submit 15 tasks, then poll for them to complete and again submit 15 tasks. Now what happens is when we are submitting the 11th task at that time warp executing 1st and 2nd tasks become free, but only 1 can take the task (or it may be some other warp, in that case also these 2 will keep waiting till submission of next bundle of tasks). This idle time gives the above plot. So the fundamental is to schedule as much job as possible in each iteration.



This plot tells us that CPU only code beats the GEMTC code in performance. This is attributed to the fact that the data-size is too small because of this the GEMTC overhead of moving data Host to device and back dominates the processing advantage given by CUDA cores.



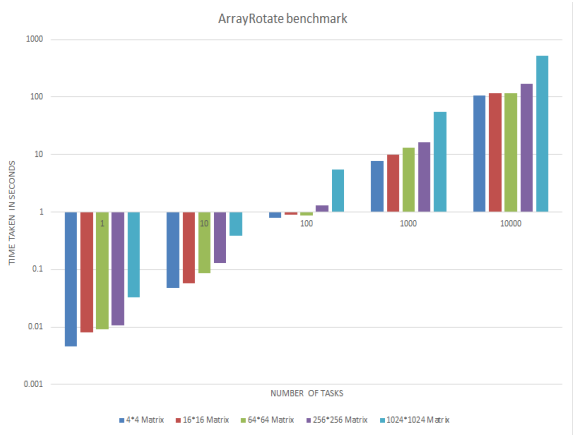
Here GEMTC outperforms CPU, because the data-size is large GEMTC overhead is less than time spend in data processing inside the kernel. The interesting part to see is that the performance of GeMTC is improving when number of tasks are 10 to 100 and but seems decrementing when number of tasks is 1000. This sounded odd for a while and I found latter that this is attributed to the fact that "GTX 480" has 1.6 GB of memory and Array-size with 1 M entries will occupy 8*1MB memory (8 because each entry is double). Now there is no way to schedule 8*1000MB (8 GB) anyhow. Hence I scheduled 100 batch jobs at a time, which gave the above behavior and we can see that time taken at 1000 tasks is 10 times the time taken at 100 tasks for GeMTC. In the table below we see that kernel itself doesn't take much time to execute for lower data

size. But at higher data size HOST takes over, this is because the shader clock rate of GPU is lower than CPU's clock rate, which has direct implications on Instructions per Second.

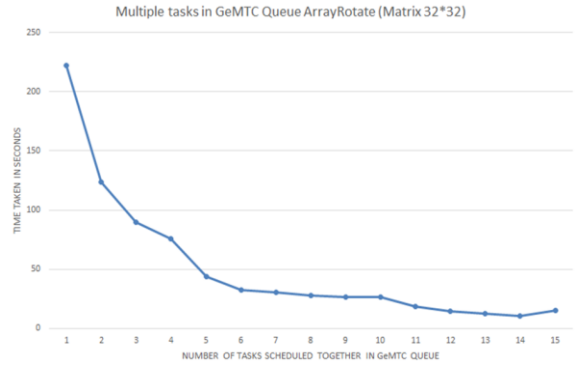
ArrayAtomic kernel benchmark inside CUDA.

Array Size	Kernel Time milli-seconds	Host Time milli-seconds
32	0.000634	0.006
1K	0.010601	0.032
32K	0.472029	0.941
1M	15.106884	8.898

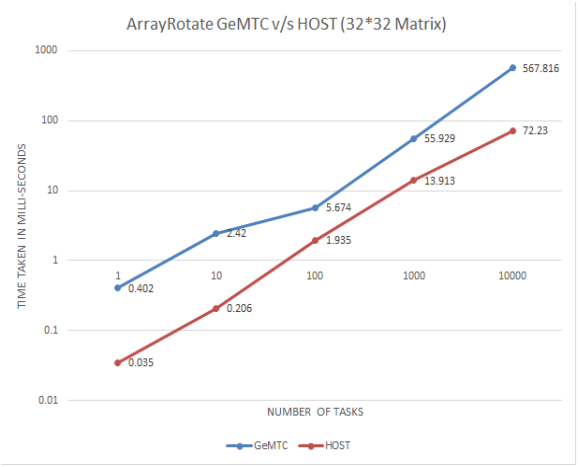
The plot of ArrayRotate with varying data-size.



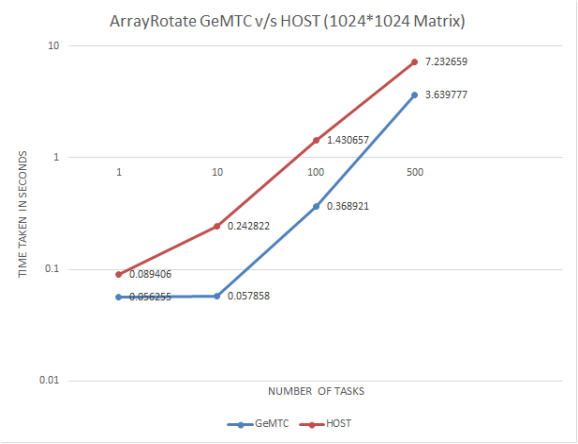
For comparing data-size only hence there is only 1 task on GEMTC queue at a time. We see linear rise in time taken to complete jobs with increment in number of tasks also in each set the time taken by larger data-size array is more when compared to task with less data-size. This is all expected behavior.



The behavior and its explanation is similar to that of ArrayAtomic benchmark.



This plot tells us that CPU only code beats the GEMTC code in performance. This is attributed to the fact that the data-size is too small because of this the GEMTC overhead of moving data Host to device and back dominates the processing advantage given by CUDA cores.

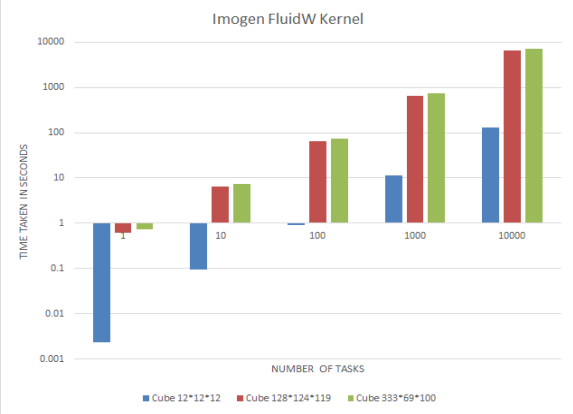


Here we see that GeMTC out-performs HOST when 10 to 500 tasks are submitted together in GeMTC queue. The interesting part to see is that the performance of GeMTC is improving when number of tasks are 1 to 500. For this big data size 1024*1024 we can't schedule more than 85 tasks at a time in GeMTC queue."GTX 480" has 1.6 GB of memory and Matrix-size with 1024*1024 will occupy 8*1MB memory (8 because each entry is double). Also there are 2 matrices which are actually transferred to the GeMTC kernel. Now there is no way to schedule (2*8*1 MB * 85) (1.42 GB), there will be GeMTC overhead of tasks descriptors, queues also. At 500 tasks we start hitting memory leak bug if we try to schedule more than 85 tasks per iteration.

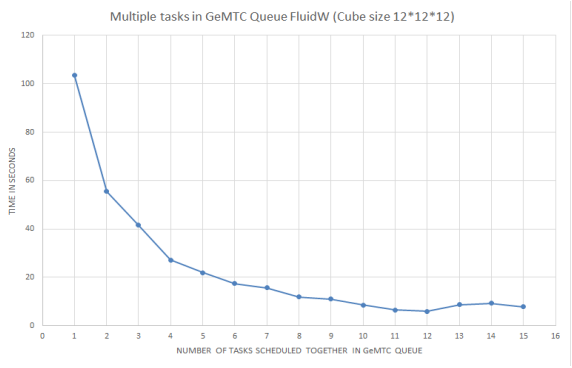
The behavior of time taken in kernel v/s time taken by host is almost dominated by host from 64*64 size data. The shader clock rate of GPU is lower than CPU's clock rate, which has direct implications on Instructions per Second.

Matrix	Kernel Time milli-seconds	Host Time milli-seconds
4*4	0.001490	0.003
16*16	0.023091	0.035
64*64	0.092381	0.043
256*256	1.759171	0.822
1024*1024	28.543989	21.157

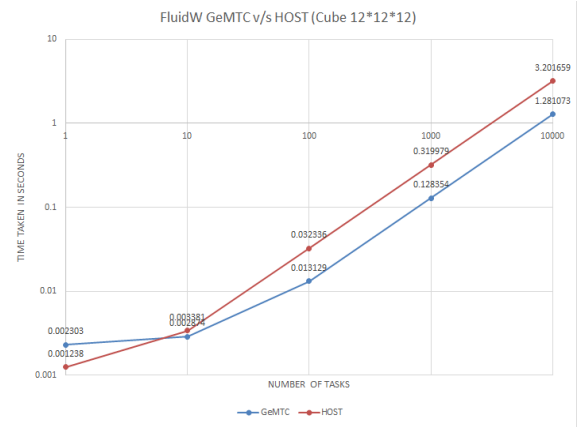
The plot of FluidW with varying data-size.



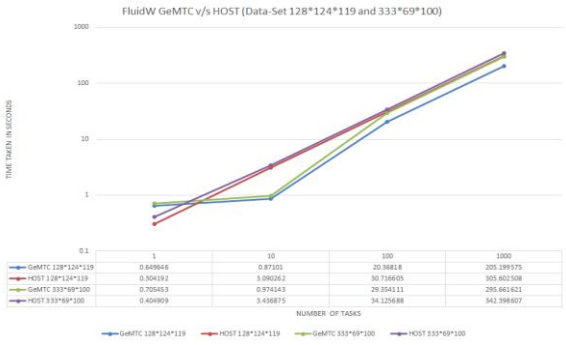
The above graph plots the time taken to complete Imogen FluidW tasks by GeMTC. As usual the size of data has direct impact on the time taken to complete the number of tasks. We also observe that there is almost linear increase in time taken to complete tasks with rise in number of tasks. Also to be noted that data size provided to GeMTC is 181.439 MB for 128*124*119 (because the number of equation parameters are large), 220.634 MB for 333*69*100, 167KB for 12*12*12. Remember that this weird data-set odd looking dimensions come from testing program of FluidW kernel of Imogen.



The behavior and its explanation is similar to that of ArrayAtomic benchmark.



Here we see that GeMTC out-performs HOST when 10 to 10000 tasks are submitted together in GeMTC queue. For 10000 set of tasks 1000 tasks are submitted in batch together in GeMTC queue.



We see that GeMTC outperforms HOST in respective data set benchmarks. Points to note 1) For data-set 128*124*119 at best we can schedule 7 tasks at a time in GeMTC queue because each task requires

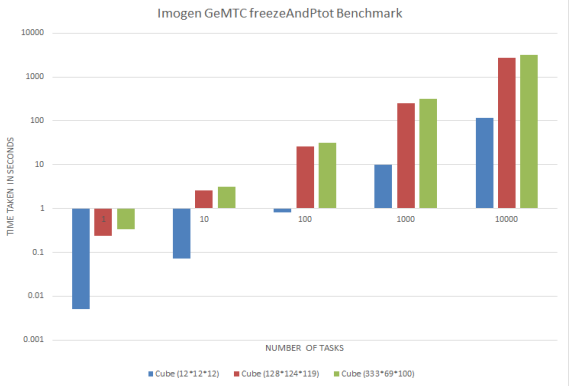
181439824 bytes of data to be submitted to GeMTC. 2) For 100 and 1000 tasks only 4 jobs at a time were scheduled to avoid GeMTC memory leaks. 3) For data-set 333*69*100 at best we can schedule 6 tasks at a time in GeMTC queue because each task requires 220634448 bytes of data to be submitted to GeMTC. 4) For 100 and 1000 tasks only 3 jobs at a time were scheduled to avoid GeMTC memory leaks. 5) Due large data set per kernel it is wasting CUDA cores by keeping SMs idle. And we are not happy about it.

It becomes a notable point that if we are scheduling a very heavy task in GEMTC it will under-utilize processing power, the best case will be to sub-divide this heavy task process it with GEMTC which will let us high the sweet spot for performance gain and utilization.

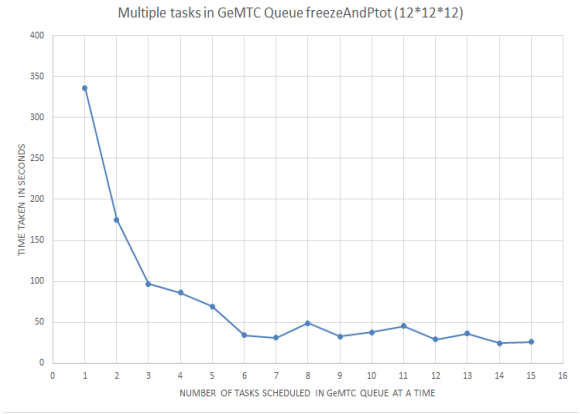
FluidW kernel benchmark from insde CUDA kernel.

Task size	Kernel Time milli-seconds	Host Time milli-seconds
12*12*12	1.027894	1.220
128*124*119	464.883484	303.851
333*69*100	527.947815	341.746

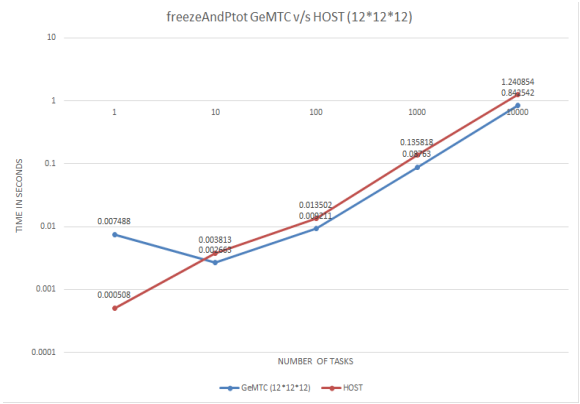
freezeAndPtot benchmark



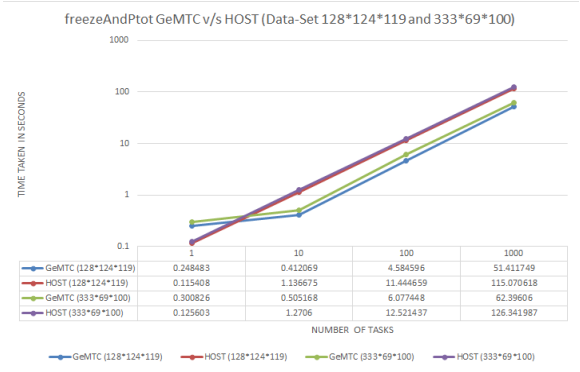
The above graph plots the time taken to complete Imogen freezeAndPtot tasks by GeMTC. As usual the size of data has direct impact on the time taken to complete the number of tasks. We also observe that there is almost linear increase in time taken to complete tasks with rise in number of tasks. Also to be noted that data size provided to GeMTC is 90.778 MB for 128*124*119 (because the number of equation parameters are large). 110.344 MB for 333*69*100 and 84.136KB for 12*12*12.



This benchmark analyses impact of scheduling multiple freezeAndPtot tasks in GeMTC queue. The total number of tasks scheduled is 20000.



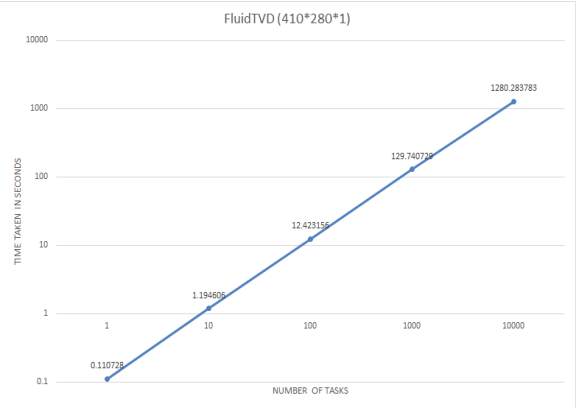
Here we see that GeMTC out-performs HOST when 10 to 10000 tasks are submitted together in GeMTC queue.



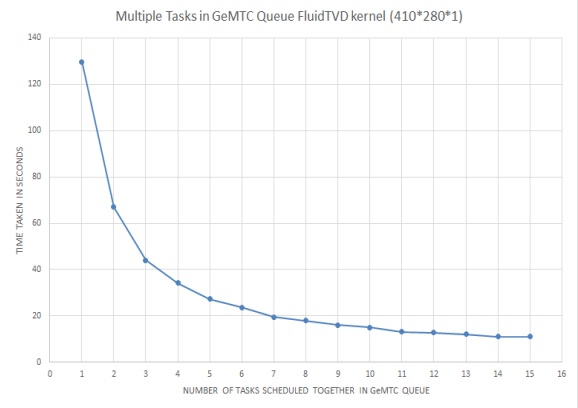
We see that GeMTC outperforms HOST in respective data set benchmarks. Points to note are 1) For data-set 128*124*119 at best we can schedule 10 tasks at a time in GeMTC queue because each task requires

90778952 bytes of data to be submitted to GeMTC. 2) For 100 and 1000 tasks only 8 and 7 jobs at a time were scheduled respectively to avoid GeMTC memory leaks. 3) For data-set 333*69*100 at best we can schedule 10 tasks at a time in GeMTC queue because each task requires 110344840 bytes of data to be submitted to GeMTC. 4) For 100 and 1000 tasks only 7 jobs at a time were scheduled to avoid GeMTC memory leaks. 5) Due large data set per kernel it is wasting CUDA cores by keeping SMs idle. Table for kernel time v/s host only time.

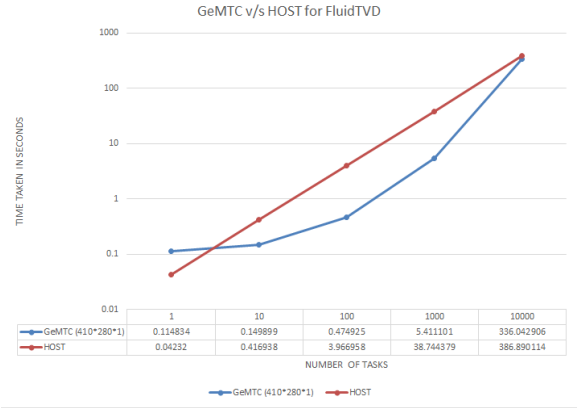
Task size	Kernel Time milli-seconds	Host Time milli-seconds
12*12*12	0.602769	0.166
128*124*119	197.565033	114.431
333*69*100	236.838699	133.920



The above graph plots the time taken to complete Imogen FluidTVD tasks by GeMTC. As usual the size of data has direct impact on the time taken to complete the number of tasks. We also observe that there is almost linear increase in time taken to complete tasks with rise in number of tasks.Also to be noted that data size provided to GeMTC is 10.104688 MB for 410*280*1 (because the number of equation parameters are large). This was the data set provided by Erik the original writer of Imogen.



This benchmark analyses impact of scheduling multiple FluidTVD tasks in GeMTC queue. The total number of tasks scheduled is 1000, the explanation is same as of ArrayRotate.



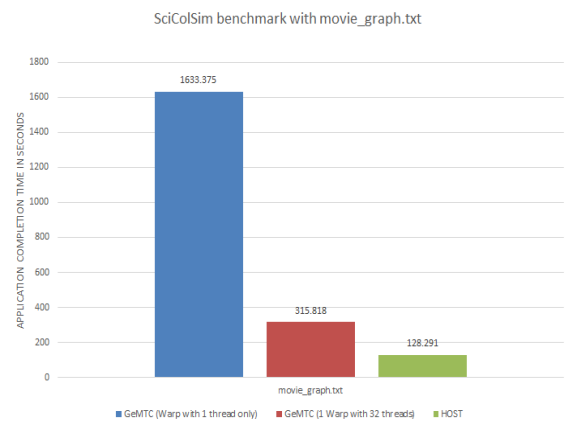
	1	10	100	1000	10000
GeMTC (410*280*1)	0.114834	0.149899	0.474925	5.411101	336.042906
HOST	0.04232	0.416938	3.966958	38.744379	386.890114

Here we see that GeMTC out-performs HOST when 10 to 10000 tasks are submitted together in GeMTC queue. At 1000 we were only able to submit 50 jobs together in GeMTC queue (memory bug becomes dominant). At 10000 we were only able to submit 4 jobs together in GeMTC queue (memory bug becomes dominant).

Task size	Kernel Time milli-seconds	Host Time milli-seconds
410*280*1	86.353767	42.628

This is benchmark for CUDA kernel time taken. Explanation is same as for ArrayRotate.

Here are the set of performance evaluation of SciColSim



Time taken by GeMTC kernel is more than HOST code itself. But this has always been there in Imogen also. GeMTC gets performance by scheduling multiple tasks in GeMTC queue together, which are in parallel taken up by workers and executed. This is not happening for SciColSim. Why this can't be achieved ? (remember there is only 1 GeMTC kernel for an expensive function but application launch as such has good amount of sequential code also. Multiple GeMTC tasks can be submitted only when there are multiple launch of SciColSim via Swift script. (But how many launch can work in parallel directly depends on host processor.

1-Thread Warp Simulation Time (milliseconds)	32-Thread Warp Simulation Time(milliseconds)	Host Time milli-seconds
967.148804	95.298195	10.000000

6. RELATED WORK

Exploiting MIMD control flow on SIMD GPUs has been explored in [4] where a stack is added to allow SIMD processing elements to execute distinct program path post occurrence of a branching instruction. It proposes dynamic warp formation for regrouping of processing elements of individual SIMD warps for efficient branch handling. MIND Interpretation on GPU[5] discusses compiler, assembler and interpreter which will not only allow MIMD execution model but also supports message passing, shared memory communication etc. [6] has analyzed CUDA workloads using a GPU simulator. The paper establishes that non-graphics applications are more sensitive to bisection bandwidth than latency and many times it is better to reduce the number of threads to avoid content on memory resources. [7] Summarizes tools and techniques in GPU computing. [8] Discusses implementation and design of SIMD-MIMD GPU architecture. [14] Discusses the problem statement of integrating data flow driven parallel programming systems and hardware accelerators. The work aimed to enable Swift to efficiently use accelerators to further accelerator wide range of applications, on a growing portion of high end systems. [15] Discusses static batch FIFO scheduler which sits between Swift and GPU handles multiple inputs from Swift and condenses these into single GPU calls.

7. CONCLUSION

5 Imogen purehydro kernels were successfully ported to GeMTC and were benchmarked. Also Host only code corresponding for these kernels was written and compared with GeMTC. It was observed that there is sweet space between the varying data sizes for each kernel where it was outperforming HOST only code. Expensive function from SciColSim was successfully ported and benchmarked. A good amount of time was spend in feasibility analysis for complete SciColSim migration to GeMTC kernel.

Future work on Imogen includes 1) Achieve better performance by breaking kernels into smaller pieces and submit these smaller pieces as tasks to GeMTC. Migrate magnetic kernels also (as of now I migrated fluid kernels only). All these kernels have to be integrated together to get Imogen's complete behavior. Future work of SciColSim include best case to be to find equivalent or implement 1 library just for all graph features being used by SciColSim. I still have doubt on whether this will still result in better performance as lot of other functions in SciColSim are sequentially dependent. This can have serious implication as it will result in 1 CUDA core running this sequential code which will be too slow. Enable support for launching multiple workers using GeMTC kernel.

As part of the project I also identified class of applications which when migrated to GeMTC will show very high efficiency. Monte Carlo methods are broad class of computational algorithms that rely on repeated random sampling to obtain numerical results. There is a large class of application which utilize this method for obtaining results ranging from physical sciences, engineering, computational biology, computer graphics, applied statistics etc. As a start I wrote a program to calculate value of PI using GeMTC. It relied on sending random values to GeMTC and calculating PI inside the kernel. This can be avoided by directly generating random values inside the GPU kernel itself. These classes of applications will benefit a lot for GeMTC because the amount of data to be moved to and fro between CPU and CPU will be less which will directly result in significant performance improvement. Hence

migrating these classes of application to GeMTC will also be attempted.

8. REFERENCES

[1] Scott Krieder, Ioan Raicu. "GEMTC: GPU Enabled Many-Task Computing", Illinois Institute of Technology, Department of Computer Science, PhD Oral Qualifier, 2013

[2] Erik Keever, James N. Imamura "Imogen: A Parallel 3D Fluid and MHD Code for GPUs", University of Oregon, OR, USA, 27th International ACM conference on International conference on supercomputing, 2013

[3] Justin M. Wozniak, Timothy G. Armstrong, Ketan Maheshweri, Ewing L. Lusk, Daniel S. Katz, Michael Wilde, Ian T. Foster "Turbine: A distributed-memory dataflow engine for extreme many-task applications", Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies, 2012

[4] Wilson W. L. Fung, Ivan Sham, George Yuan, Tor M. Aamodt, "Dynamic Warp Formation: Exploiting Thread Scheduling for Efficient MIMD Control Flow on SIMD Graphics Hardware", Department of Electrical and Computer Engineering, University of British Columbia, 40th IEEE/ACM International Symposium of Microarchitecture

[5] Henry G. Dietz, B. Dalton Young, "MIMD Interpretation on a GPU", Electrical and Computer Engineering, University of Kentucky, Proceedings of the 22nd international conference on Languages and Compilers for Parallel Computing, 2009

[6] Ali Bakhoda, George L. Yuan, Wilson W. L. Fung, Henry Wong, Tor M. Aamolt, "Analyzing CUDA Workloads Using a Detailed GPUSimulator ", Performance Analysis of Systems and Software, 2009, IEEE international Symposium

[7] John D. Owens, Mike Houston, David Luebke, Simon Green, John E. Stone, James C. Philips, "GPU Computing", University of California, Proceedings of IEEE Volume 96, Issue 5

[8] J. Lucas, S. Lal, M. Alvarez-Mesa, A. Alhossini, B. Juurlink, "Design and Implementation of SIMD-MIMD GPU architecture ", IISWC, 2010 IEEE International Symposium, 2010

[9] "NVIDIA's next generation CUDA computing architecture Fermi", Whitepaper

[10] Benjamin Grimmer, Scott Krieder, Ioan Raicu, "Enabling Dynamic Memory Management Support for MTC on NVIDIA GPUs", Illinois Institute of Technology, EuroSys 2013

[11] Dustin Shahidehpour, Scott Krieder, Ben Grimmer, Ioan Raicu. "Accelerating Scientific Workflow Applications with GPUs", 2nd Greater Chicago Area System Research Workshop (GCASR), 2013

[12] Scott J. Krieder, Benjamin Grimmer, Dustin Shahidehpour, Jeffrey Johnson, Justin M. Wozniaky, Michael Wildeyz, Ioan Raicu. "Towards Efficient Many-Task Computing on Accelerators in High-End Computing Systems", 2nd Greater Chicago Area System Research Workshop (GCASR), 2013

[13] Jeff Johnson, Scott Krieder, Benjamin Grimmer, Justin Wozniak, Michael Wilde, Ioan Raicu. "Understanding the Costs of Many-Task Computing Workloads on Intel Xeon Phi Coprocessors", 2nd Greater Chicago Area System Research Workshop (GCASR), 2013

[14] Scott Krieder, Ioan Raicu. "Towards the Support for Many-Task Computing on Many-Core Computing Platforms", Doctoral Showcase, IEEE/ACM Supercomputing/SC 2012

[15] Scott Krieder, Ben Grimmer, Ioan Raicu. "Early Experiences in running Many-Task Computing workloads on GPGPUs", XSEDE 2012

[16] Scott Krieder, Ioan Raicu. "An Overview of Current and Future Computing Accelerator Architectures", 1st Greater Chicago Area System Research Workshop, 2012

[17] A Primer on Eulerian Computational Fluid Dynamics for Astrophysics Publications of the Astronomical Society of the Pacific 115:303–321, 2003 March