Power Profiling of GeMTC Many Task Computing

Sean Wallace, Scott Krieder, and Ioan Raicu

Illinois Institute of Technology, Chicago, IL, USA
swallac6@iit.edu, skrieder@iit.edu, and iraicu@cs.iit.edu

***Abstract*—GeMTC allows for Many Task Computing (MTC) workloads to run on hardware accelerators allowing for advantages that come from the many-core architecture. However, presently GeMTC is only written to take advantage of NVIDIA GPUs. Another such hardware accelerator, the Intel Xeon Phi, is also an excellent candidate for MTC workloads. Therefore, the first goal of this project will be to add support to GeMTC to allow it to run on Xeon Phi. While there has been plenty of research on power consumption of hardware accelerators, MTC workloads are a significantly understudied research area. MonEQ, a power profiling library, was primarily developed to measure power consumption of the IBM Blue Gene/Q supercomputer, but has lately evolved to also include profiling of both NVIDIA GPUs as well as the Intel Xeon Phi. As a second goal, this project seeks to profile real MTC workloads running on both NVIDIA GPUs as well as on the Xeon Phi.**

I. Background Information

GeMTC [1], [2] enables Many Task Computing (MTC) workloads to run efficiently on NVIDIA GPUs. The standard CUDA framework can only support 16 kernels, one kernel for each streaming multiprocessor (SM). The biggest downside to this is that each kernel must be started and stopped at the same time, which leads to incredible inefficiencies in heterogeneous workloads which are commonplace for MTC. By working at the warp level, a trade can be made for local memory and concurrency which allows the execution of up to 200 concurrent kernels. Preliminary results show that GeMTC framework is able to achieve a higher level of efficiency for various MTC workloads.

MonEQ [3], [4] is a power monitoring library initially designed to measure the power consumption of scientific applications running on the IBM Blue Gene/Q supercomputer. Results show that extremely accurate, precise, and sub-second power data can be obtained with marginal overhead added. The Blue Gene series of supercomputers is homogeneous, however, so the recent evolution of MonEQ has enabled the profiling of applications running on both NVIDIA GPUs and Intel Xeon Phi accelerators (potentially at the same time) using the vendor supplied low-level APIs for data gathering again at sub-second intervals.

II. Problem Statement

This project would seek to solve two problems:

1) Expand GeMTC to also support the Intel Xeon Phi.
2) Integrate MonEQ into GeMTC to allow for the collection of power data of MTC workloads on both NVIDIA GPUs as well as the Intel Xeon Phi.

Towards the first, a "first version" framework has been actively developed for GeMTC supporting the Xeon Phi. This project seeks to improve upon this framework adding, most importantly, support for multi-node clusters/supercomputers.

Towards the second, at the start of this project MonEQ had very little support for hardware accelerators. The first step in profiling MTC workloads on hardware accelerators is to develop, test, and profile applications running on accelerators with MonEQ. The second would be to integrate this finished framework into GeMTC at the "super kernel" level allowing for profiling of any kernels which can be plugged into GeMTC on either a GPU or a Xeon Phi.

III. Proposed Solution

As discussed in Section II, there are two parts to this project. The GeMTC framework has already very mature support for GPUs and as such no further improvement was necessary there. The Xeon Phi implementation of GeMTC however, while functional, needed further improvement to support more diverse workloads. As an example, the previous version of the Xeon Phi GeMTC didn't have support for how many threads the chosen kernel should run with.

Both the CUDA optimized and the Xeon Phi optimized versions of GeMTC have a similar structure. The basic idea is that workloads are implemented into what are referred to as kernels. These kernels are launched by what's called the "super kernel". The super kernel has direct control over which tasks run, when they run, etc. A typical kernel should be thought of a what one task will need to compute. For example, in the case of a vector add, the kernel would execute the code necessary to compute say one row of the target matrix. Of course, there are a number of ways kernels can be designed and there are performance as well as power considerations that go into this, however these topics are left for future exploration and are not discussed in this work. In this work, for all kernels developed, the idea is to assign one unit of work to one task– the finest level of granularity possible. In this way, going back to the vector add example, one task is responsible for adding the applicable index of one of the source arrays to the result vector. Thus, if you had two arrays each with 10 values, there would be a total of 20 tasks.

Setting up GeMTC is a simple process with a single call to the library for both the CUDA and Xeon Phi versions. `gemtcSetup()` is the method responsible for this. It takes two arguments; in the CUDA version the first is the queue size, and the second is an overfill flag. Put simply, the queue

size determines the size of both of the queues in the GPU memory that hold pointers to task description structures as well as finished jobs. The overfill flag has two values:

- 0 - Launches enough warps to have a one-to-one mapping with 16 CUDA core groupings. This is high efficiency.
- 1 - Launches the maximum number of warps per SM giving the highest throughput.

In the Xeon Phi version however, the overfill flag is replaced for a number of workers flag. This number determines how many pthreads are dispatched onto the super kernel. This idea is similar to the CUDA implementation but since Xeon Phi's don't have SM's, the implementation is fundamentally different.

The CUDA and Xeon Phi components have a similar lower level structures. The basic idea is each task has some attributes/parameters/data/etc. that it must have in order to perform its computation. For both NVIDIA and Intel devices, there are explicit calls that can be used to allocate memory for device programs as well as copy the necessary data to the device. To allocate the memory on the device, a call to `gemtcMalloc()` will set aside the requested amount of memory and return a pointer to this memory in the device. Copying data to the device is is abstracted in GeMTC in the form of a call to `gemtcMemcpyHostToDevice()` where the arguments are the host's pointer to the data, the device's pointer to the allocated area of memory in the previous step, as well as how much data is to be copied.

After the necessary data has been copied to the device, the driver program begins to call `gemtcPush()`. This call takes as argument a pointer/reference to the kernel to be launched, the number of threads which should be used for each kernel launch, as well as the parameters to the program. Inside the library, GeMTC takes these pushes and adds them to the master queue where these jobs are subsequently launched when the necessary resources are available. The completion of jobs is checked with a call to `gemtcPoll()` which is a non-blocking call which will return the unique identifier for the launched kernel as well as it's return state. In this way, one can push some number of tasks and then poll for that same number of jobs. For example, one could push every task and subsequently poll for all the jobs. Or, one could push half the total tasks, poll for those tasks, then push and poll for the latter half of jobs. This idea of a collection of tasks is what is referred to as a task group, which will come up later in this report.

All of these features at the time of starting this project were only implemented for a single node. To add multi-node support, the principal remains the same, however a master process will coordinate which groups of task groups get sent to which node by using MPI. For example, if one had 100 tasks and 2 nodes, 50 tasks would be sent to the first node and 50 to the second. The task groups again could be whatever size desired.

As far as MonEQ was concerned, as previously discussed it did not have the necessary functionality to profile applications running on hardware accelerators. The way MonEQ works on all platforms is to utilize vendor supplied low-level APIs which poll data from sensors located in hardware. This data is maintained in a data structure which can either be output to flat file or the running application can ask for direct access.

Once accelerator support was added to MonEQ, the process was simple. The MonEQ library is activated by a single call, `MonEQ_Initialize()`. This call sets up the data structure and registers for a signal which it intercepts at predetermined times. After initialization, one need not do anything else, but if desired MonEQ allows for tagging of specific ares of code. To accomplish this, the block of code that is to be separated from the rest of the program is surrounded by two calls, `MonEQ_StartPowerTag()` and `MonEQ_EndPowerTag()`. These calls result in special markers being added to the data structure so eventually when `MonEQ_Finalize()` is called, separate files are created which contain the data for these tagged ares of code. The remainder of the program is still profiled and output.

IV. Evaluation

Experiments were run both on GPUs as well as the Xeon Phi. For both, sleep workloads (both with and without GeMTC integration) and vector add workloads (again, both with and without GeMTC) were profiled.

A. GPU Evaluation

Sleep workloads are an ideal starting point because they perform no useful computation and have no memory access, so the power consumption should be as near to idle as possible. Therefore, any increase in power consumption as a result of using GeMTC will be evident.

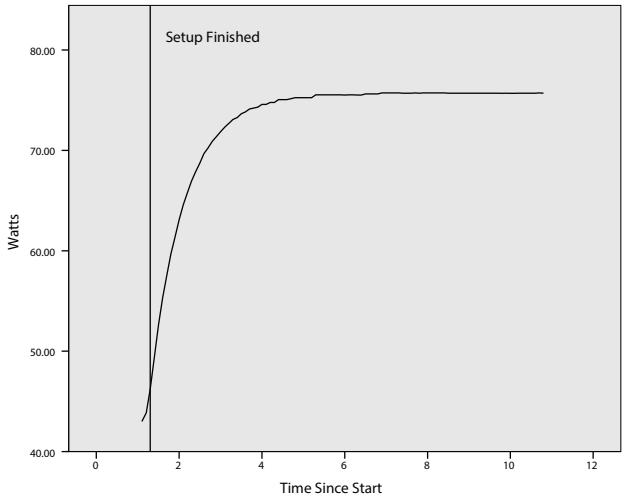

Fig. 1. Shows sleep MTC workload using GeMTC. After GeMTC is fully setup, the power consumption continues to slowly increase over the course of a few seconds leveling off near the 75 Watt mark.

Figure 1 shows the result of a sleep(0) (sleep for 0 seconds) workload using GeMTC with the point at which the GeMTC framework finished initializing marked. The workload was setup such that 100,000 tasks were dispatched to the GPU in blocks of 1,000 at a time. The figure shows that setup takes

very little time and power consumption steadily increases over a period of about 6 seconds until finally leveling off. This result shows a run of only 12 seconds, but this power curve is the same no matter if the run is significantly longer. That is to say that once the GPU gets "warmed" up, power consumption for this workload remains constant.

The very fact that the power consumption follows this increasing curve is interesting. While I can't say for certain exactly why this is, it's likely due to the fact that tasks in the GeMTC framework are launched incredibly quickly (this experiment showed that GeMTC on GPUs is capable of 10,000+ tasks/second), but also sequentially. Said another way, tasks are pushed onto the device one at a time. Once the entire GPU is occupied, the power consumption levels off because tasks which are completing are being replaced immediately by tasks which have not yet run.

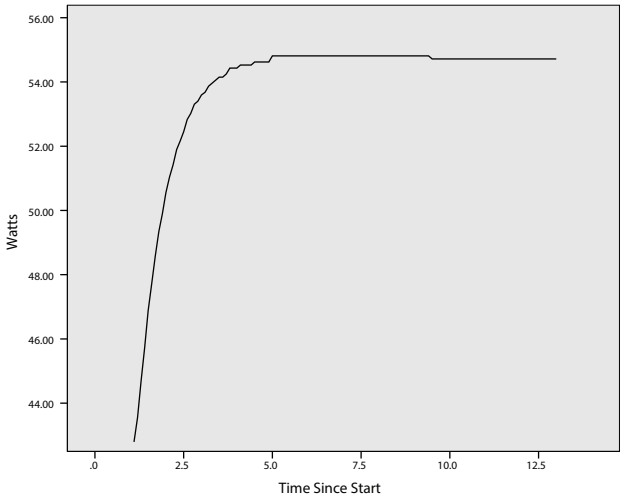

Fig. 2. Standard sleep (without using GeMTC) on GPU. Shows similar power curve as the GeMTC version but maximum power consumption is about 20 Watts less.

The next logical question is whether this increasing power curve phenomenon is also exhibited in a sleep workload which does not use GeMTC. Figure 2 shows just such a workload and interestingly also shows the same power curve. It's worth noting that once the power consumption did level off it did so at a level which was about 20 Watts less than the GeMTC sleep workload. Therefore, it's safe to say that the overhead of GeMTC is about 20 Watts with even the simplest of workloads.

Having evaluated a sleep workload which doesn't perform any interesting computation, the next step was to look at what an application that does perform real work looks like. Figure 3 shows the power consumption of a vector add workload with input array size of 134,217,728 (2^{27}) and 10 arrays of double precision floating point values. As can be seen from the graphic, the same power ramp up over the first 6 seconds or so is still visible. From there once the data has been generated we again see a steep increase in power consumption to about the 140 Watt level where it stays until the program eventually completes. The last little tail drop down is the point when the result data is copied from the device to the host.

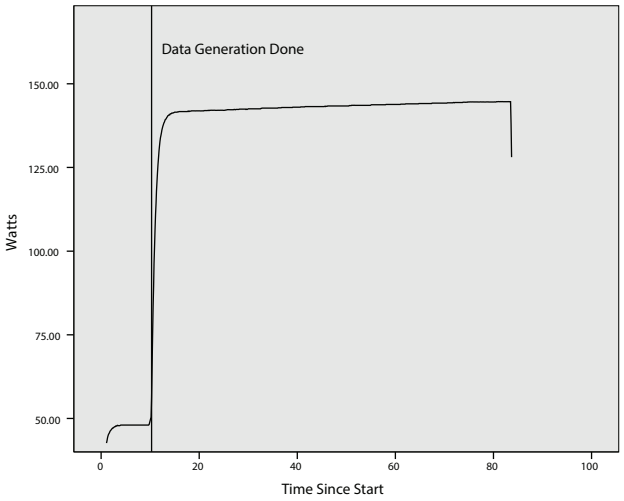

Fig. 3. Vector add implemented with vanilla CUDA. At beginning shows power ramp up present in both sleep tests. After data generation power rapidly increases to a level of about 140 Watts and stays there for the remainder of the computation.

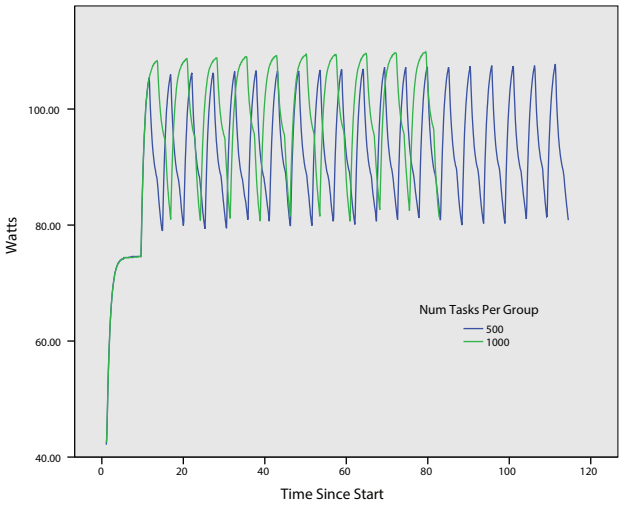

Fig. 4. Vector add implemented with GeMTC. Shows power consumption for two different workloads. For both we see that the power consumption follows a wave pattern increasing when the tasks are actually pushed to the device and decreasing as GeMTC waits for tasks to complete. 1000 tasks per group ultimately has a higher peak power consumption, but completes about 20 seconds quicker than 500 tasks per group.

The next experiment was to then to convert the vector add workload into a GeMTC kernel. In this case, each kernel call was responsible for computing one unit with the input sizes and number of input arrays remaining the same as in the previous experiment. As can be seen from Figure 4, this produces a much more interesting power consumption graph. As with all experiments, the initial ramp up of power is still, however, once the data had been generated and tasks were then pushed to the GPU, a much different power trend emerges.

This experiment was run with two task group sizes: 500 tasks per group and 1000 tasks per group. As can be seen from the figure, in both cases a sin wave type power consumption

trend is visible. This is likely due to the fact that as the tasks were pushed to the device power consumption greatly increased because the device was fully populated and working at full capacity. However, as tasks begin to complete the power consumption starts to drop off because more of the device is idle. Finally, once all of the tasks in a group have completed and the results gathered, the power consumption spikes again. This pattern continues until all of the tasks have executed.

Interestingly, two things are different between the 500 and 1000 tasks per group runs. First, in the 1000 tasks per group run, the peak power consumption is actually higher. This is likely because as there are more tasks the device is kept at peak for longer (having more to do for a given task group) resulting in an increase in power consumption. Second, the 1000 tasks per group run completes about 20 seconds sooner than the 500 tasks per group run. This makes intuitive sense as there is going to be more overhead associated with dispatching and then polling more times. However, this trend of shorter run times does not continue on all the way to dispatching all of the tasks simultaneously. At about the 1200 tasks per group level the run time stops decreasing and above that actually starts to increase again.

B. Xeon Phi Evaluation

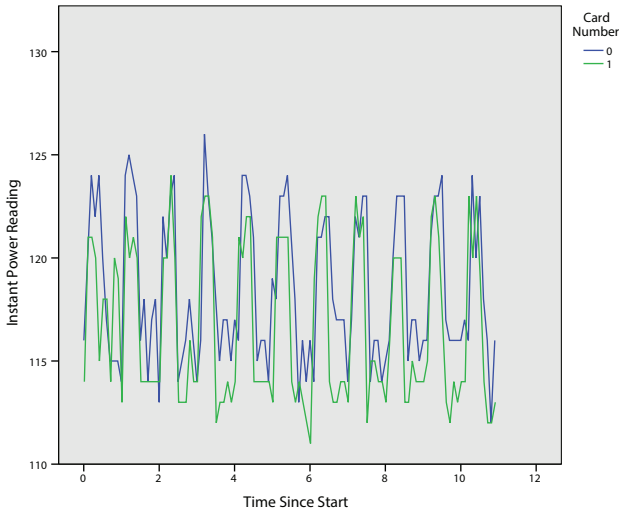

Fig. 5. Sleep workload on Xeon Phi without GeMTC. Shows chaotic power consumption for both cards installed in machine. Range is about 10 Watts between highest and lowest power consumption.

Beginning again with sleep workloads, Figure 5 shows a standard sleep(0) workload without GeMTC. As can be seen from that figure, the power consumption here is much more chaotic bouncing between about 115 Watts and 125 Watts. This experiment was run on a system with two Xeon Phi cards installed. Also evident is that both cards exhibit this trend for the entirety of the execution.

When the sleep workload is converted into a GeMTC kernel as shown in Figure 6, while the power consumption is in fact less chaotic, it still bounces up and down from about the same levels as the previous experiment. There is

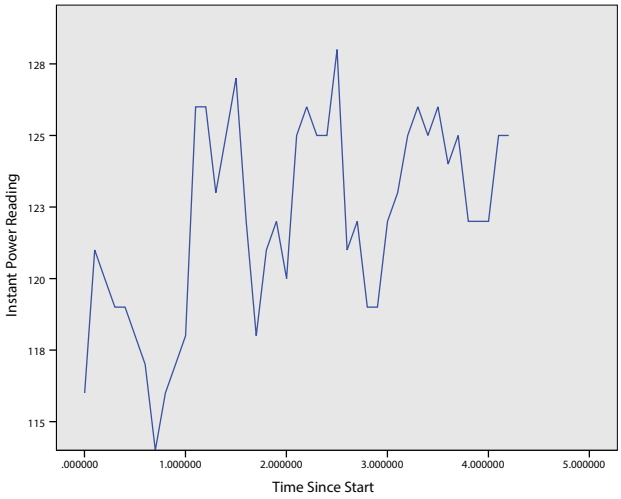

Fig. 6. Sleep workload on Xeon Phi utilizing GeMTC. Shows less chaotic pattern as opposed to directly offloading sleep instruction to Mic, however the power consumption still has a range of about 15 Watts.

however one interesting fact that can be taken from this. In the GPU sleep workloads, the GeMTC kernel version exhibited a greater power consumption than did the standard CUDA implementation. Moving over to the Xeon Phi however, while tough to say conclusively due to the range of consumption during the experiment, the average over the whole run isn't any higher than the standard offloaded code.

It's tough to say exactly why this is, but one theory is that the Xeon Phi actually has a Linux operating system running on it. As in a traditional computer, it's responsible for scheduling, swapping out processes, etc. This overhead very well might contribute both to the chaotic power consumption as well as the GeMTC workload not consuming more power.

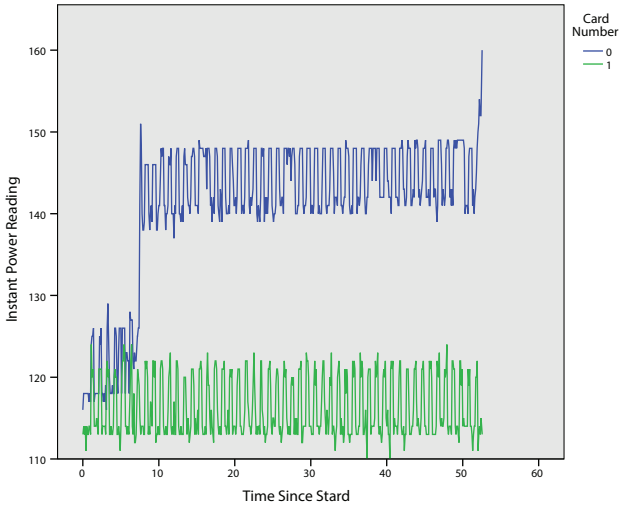

Fig. 7. Vector add workload on Xeon Phi without GeMTC. Shows power consumption of two cards. The workload was only offloaded to card number 0, the other card's data was left just to show that the up-down power curve exhibited in the sleep workloads is still present. After data generation is completed near the 10 second mark, power consumption quickly increases to the 140-150 Watt range staying there for the remainder of the computation.

Again keeping with the vector add workload, Figure 7 shows what a standard OpenMP optimized vector add workload looks like in terms of power consumption on the Xeon Phi. First thing worth noting is that the up/down power consumption seen in the sleep workloads is still present. However, after data generation completes at about the 10 second mark, there is a clear jump in power consumption to about the 140-150 Watt range where it remains for the remainder of the computation. The second card, while not used for useful computation, was left in to show that in fact even when the Xeon Phi isn't tasked with anything at all this up/down power consumption trend continues.

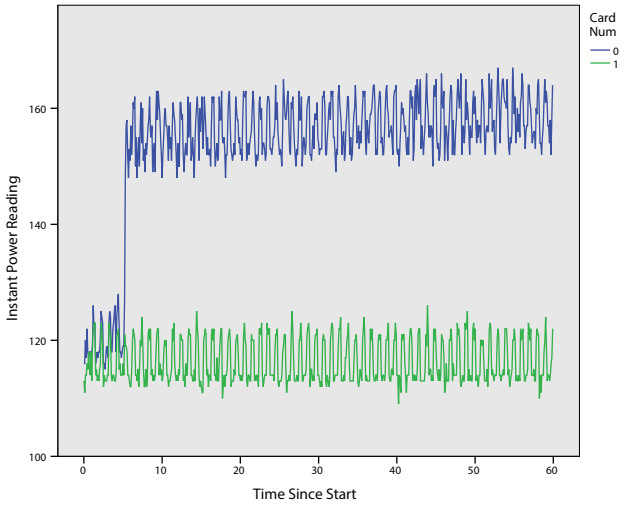

Fig. 8. Vector add workload on Xeon Phi utilizing GeMTC. Shows similar pattern to directly offloading the workload and still exhibits up-down power consumption pattern. Total power consumption is about 10 Watts higher than with traditional offloading and the total runtime is about 10 seconds longer.

In the next experiment shown in Figure 8, the vector add kernel was written for the Xeon Phi version of GeMTC. Again we see the up/down power consumption for both cards throughout the entire execution and again we see a sharp increase in power consumption once data generation has completed and computation has begun.

The first thing worth noting from this result is that the range of power consumption during which time the device was occupied with tasks is about 10 Watts higher than the standard offloaded version. Perhaps consequently, this also resulted in about 10 seconds less of total computation time. Most interesting though is that the number of tasks per group (which in this experiment was tested exactly the same as in the GPU version) doesn't show a sin wave type of power consumption curve. There are a number of factors which might play into this, but it's entirely possible this trend is still there, but impossible to discern from the data because the power consumption is already jumping up and down.

V. Related Work

Given the relatively new nature of hardware accelerators, only a handful of research studies have been conducted to date which have studied the power consumption of HPC applications running on them. Wang et al. have published heavily on the subject of power aware software solutions on heterogeneous systems with GPUs [5], [6], [7], [8], [9]. Others have looked at accurately predicting power consumption of a GPU based on the type of workload [10], [11], [12]. The Xeon Phi however (likely because it's only just become a commonly available piece of hardware) has had very little publicized on its power consumption. To my knowledge, there exists no research studies which have also investigated power consumption of MTC workloads run on hardware accelerators.

Chen at al. [13] provided a high-level GPU power consumption model using sophisticated tree-based random forest methods which could correlate the power consumption with a set of independent performance variables. This model was able to predict the GPU runtime power consumption as well as provide insight for understanding the dependence between the GPU runtime power consumption and the individual performance metrics. This evaluation was done on a GTX 280 GPU, but the authors note that the approach can be applied to any other CUDA based GPU.

Kaceli et al. [14] compared the power and performance of two designs in the many-core processor domain. In this study, the authors compared the XMT general-purpose processor to a NVIDIA GPU. The XMT provides runtime advantage on irregular parallel programs (such as graph algorithms) while the NVIDIA GPU excelled at regular parallel program which required high processing capability. In general they found that the power consumption for both models is similar for many of the workloads they studied.

Richardson et al. [15] looked at the computation, I/O, power, and memory interfacing of various accelerator devices. In this study, they compared both RMC and FMC devices measuring the performance per watt. While they did not look at the Intel Xeon Phi, they did perform experiments on several NVIDIA GPUs for both integers and floating point data structures. The authors note that while GPUs tend to perform well in most categories, but they really stand out in floating-point calculations due to their high clock rates and the sheer number of cores.

Totoni et al. [16] came the closest to part of the proposed project. They compared power consumption of various Intel Single-Chip Cloud Computer chips as well as various NVIDIA GPUs. The authors note that the Intel SCC offer a decent balance between power and performance as they consume lower power than heavy-weight multi-cores but are faster than low-power processors and do not have the portability issues of GPGPUs. They also note that GPGPUs are exceptionally powerful for many applications in speed, power and energy, however their lack of sophisticated architecture prevents them from executing complex and irregular applications efficiently.

VI. Conclusion

In this project I have evaluated the power consumption of the GeMTC library for enabling MTC workloads on hardware accelerators. The GPU component of GeMTC was already

well written and tested and did not require any further work on my behalf. The Xeon Phi component on the other hand, while an excellent start, did require tweaking to enable running tens of thousands of tasks on multiple devices.

As has been shown in the results, GeMTC doesn't come for free in terms of power consumption. In fact, for all of the experiments I ran it showed an increase in power consumption. However, in certain instances, this increase in power consumption was negated by the fact the total time to completion was actually lower than the standard method of simply offloading the whole task as one big chunk. I have also shown that on GPUs at least, the number of tasks per group makes a big impact. While this might have also been the case on the Xeon Phi, it is impossible to tell because of the already chaotic power consumption data that is obtained when no computation is being performed. Finally, regardless of workload or the use of GeMTC, GPUs when used for computation seem to have a slowly increasing power consumption curve which lasts for several seconds before leveling off and staying constant for the remainder of the computation.

In terms of the project on the whole, I have to say that it was a success. MonEQ has now been updated to support the two most common hardware accelerators, NVIDIA GPUs as well as the Xeon Phi. What's more, useful data was obtained showing that power consumption of GeMTC workloads do differ from their standard offloaded counterparts.

Looking forward, the next most obvious step is to profile more applications, both using GeMTC kernels and not, on these two platforms. Additionally, further investigation into the implications these results have at scale would be prudent. What's more, the results obtained from this work are to be view as initial and not final products. My kernel implementations were naive and simple, and there is likely much optimization possible in them.

REFERENCES

[1] S. Krieder and I. Raicu, "GeMTC: GPU enabled many-task computing," Illinois Institute of Technology, Department of Computer Science, PhD Oral Qualifier, 2013.

[2] S. Krieder and I. Raicu, "Towards the support for Many-Task Computing on many-core computing platforms," IEEE/ACM Supercomputing/SC, Doctoral Showcase, 2012.

[3] S. Wallace, V. Vishwanath, S. Coghlan, Z. Lan, and M. Papka, "Measuring power consumption on IBM Blue Gene/Q," in *The Ninth Workshop on High-Performance, Power-Aware Computing, 2013 (HPPAC'13)*, Boston, USA, May 2013.

[4] S. Wallace, V. Vishwanath, S. Coghlan, J. Tramm, Z. Lan, and M. E. Papka, "Application power profiling on IBM Blue Gene/Q," in *IEEE International Conference on Cluster Computing 2013*, Indianapolis, USA, September 2013.

[5] G. Wang and Y. Lin, "Heterogeneity-aware peak power management for accelerator-based systems," in *Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on*, 2011, pp. 396–403.

[6] G. Wang, "Power analysis and optimizations for gpu architecture using a power simulator," in *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on*, vol. 1, 2010, pp. V1–619–V1–623.

[7] G. Wang and X. Ren, "Power-efficient work distribution method for cpu-gpu heterogeneous system," in *Parallel and Distributed Processing with Applications (ISPA), 2010 International Symposium on*, 2010, pp. 122–129.

[8] G. Wang and W. Song, "Communication-aware task partition and voltage scaling for energy minimization on heterogeneous parallel systems," in *Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2011 12th International Conference on*, 2011, pp. 327–333.

[9] G. Wang, Y. Lin, and W. Yi, "Kernel fusion: An effective method for better power efficiency on multithreaded gpu," in *Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int'l Conference on Int'l Conference on Cyber, Physical and Social Computing (CPSCom)*, 2010, pp. 344–350.

[10] Q. Xie, T. Huang, Z. Zou, L. Xia, Y. Zhu, and J. Jiang, "An accurate power model for gpu processors," in *Computing and Convergence Technology (ICCCT), 2012 7th International Conference on*, 2012, pp. 1141–1146.

[11] S. Song, C. Su, B. Rountree, and K. Cameron, "A simplified and accurate model of power-performance efficiency on emergent gpu architectures," in *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, 2013, pp. 673–686.

[12] R. Suda and D. Q. Ren, "Accurate measurements and precise modeling of power dissipation of cuda kernels toward power optimized high performance cpu-gpu computing," in *Parallel and Distributed Computing, Applications and Technologies, 2009 International Conference on*, 2009, pp. 432–438.

[13] J. Chen, B. Li, Y. Zhang, L. Peng, and J.-K. Peir, "Statistical gpu power analysis using tree-based methods," in *Green Computing Conference and Workshops (IGCC), 2011 International*, 2011, pp. 1–6.

[14] F. Keceli, T. Moreshet, and U. Vishkin, "Power-performance comparison of single-task driven many-cores," in *Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on*, 2011, pp. 348–355.

[15] J. Richardson, S. Fingulin, D. Raghunathan, C. Massie, A. George, and H. Lam, "Comparative analysis of HPC and accelerator devices: Computation, memory, i/o, and power," in *High-Performance Reconfigurable Computing Technology and Applications (HPRCTA), 2010 Fourth International Workshop on*, 2010, pp. 1–10.

[16] E. Totoni, B. Behzad, S. Ghike, and J. Torrellas, "Comparing the power and performance of Intel's SCC to state-of-the-art CPUs and GPUs," in *Performance Analysis of Systems and Software (ISPASS), 2012 IEEE International Symposium on*, 2012, pp. 78–87.