Framework for Lifecycle enrichment of HPC Applications on Exascale Heterogeneous Architecture

Karan Sapra
Advisor: Melissa C. Smith
Committee Members: Frank A. Feltus, Richard R. Brooks, Walt B. Ligon III

Abstract

Recently, High Performance Computing (HPC) has experienced advancements with the introduction of multi-core, many-core, and massively parallel architectures such as Intel's Xeon-Phi coprocessors, General Purpose Graphical Processing Units (GPGPUs), and Field Programmable Gate Arrays (FPGAs). Although the aforementioned architectures can provide significant speedup, the performance greatly depends on the optimal choice of architecture for the given application. Furthermore, stepping into the exascale-computing era, heterogeneous supercomputers are gaining popularity because they can provide levels of performance that cannot be achieved on a traditional multi-core system alone. In this research we utilize a qualitative approach for enrichment and acceleration of HPC application. The enrichment of an HPC application is achieved in two phases: Development Phase using our Application-to-Architecture (A2A) followed by utilizing Functional Partitioning (FP) framework during runtime. We evaluate the accuracy of our framework using diverse representative micro-benchmark applications that belong to different regions in the application space. These micro-benchmark applications encompass a broad spectrum of application behavior, making them highly suitable for HPC studies.

Motivation

Recently, high-performance computing (HPC) has achieved new performance milestones with the introduction of multi-core and many-core architectures such as the General Purpose Graphical Processing Units (GPGPUs) and Intel's Xeon-Phi coprocessors. These architectures have leveraged significant performance improvements for several scientific applications that have performance limited by choice of the architecture on traditional processors using task- and thread-level parallelism. Although the aforementioned architectures can provide substantial speedup, the performance greatly depends on the choice of architecture for the given application, assuming a certain quality of implementation, which is beyond the scope of this project. Grozea et al. [1] and Bhuiyan [2] show that significant performance improvement is achieved when a given application is mapped to an appropriate architecture. They used an intrusion detection system and Spiking Neural Network (SNN) models: Hodgkin–Huxley [3], Izhikevich [4], Morris–Lecar [5], and Wilson [6], respectively to draw the above conclusion. Furthermore, Heterogeneous systems provide substantial potential of increase in performance factors for applications; however, much of their computing resources are often under utilized due to non-optimal Application-to-Architecture (A2A) mapping. This inefficiency leads to poor application speed-up, sub-optimal scaling efficiency, long queue delays, and increased power consumption. Bharathi et. al. [7] show that a scientific application is sum of multiple sub modular algorithms. Thus, by utilizing appropriate heterogeneous supercomputer architectures we accelerate the performance of applications. Some of current and upcoming supercomputers such as Stampede [8] and Cori [9] tend to provide accelerator heterogeneity to allow more optimal utilization of resources in scientific applications and algorithms. The first step for optimizing a complex scientific application is to create an optimal A2A mapping followed by optimizing performance of the overall application by management of subtasks. Our three major contributions are:

- Qualitative Application to Architecture framework for finding optimal architecture for a given application using A2A
- Framework for run-time maintenance of Application and Subtask in heterogeneous architecture using Functional Partitioning (FP)
- Use of A2A to determine optimal mapping of application task at runtime for optimal performance of overall application using A2A with FP

Framework Overview

Our framework is a two-stage process beginning at the start of application development. A2A Framework is qualitative mapping and the FP Framework is runtime-mapping framework. The A2A framework is used to identify optimality of architecture for application/task. These application/task can be either whole-or sub-part of the main application or node local post-processing that can be done in-situ. For complex applications like Climate Earth Science Model (CESM) [22] these post processing tasks include visualization, IO write, feature extraction and code data verification. The runtime framework FP is used to map these application-to-architecture for optimal performance.

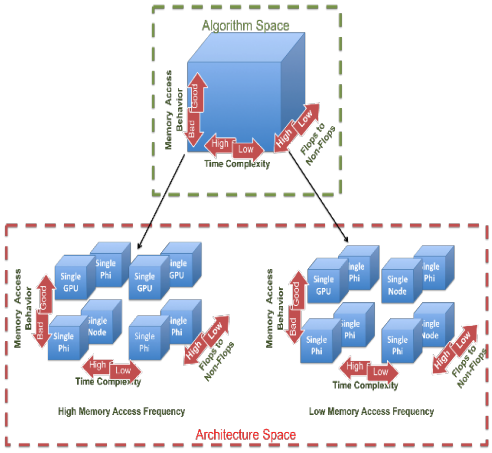


Figure 1: Tesseract Application to Architecture Mapping

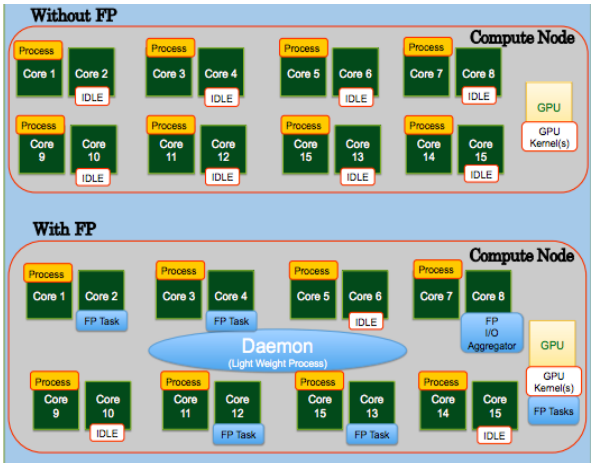


Figure 2: A Overview of FP on Heterogeneous Compute Node

A2A Mapping Framework

We propose an Application-to-Architecture (A2A) framework as shown in Figure 1 that allows for mapping for an application to most optimal architecture. Using qualitative analysis we develop a 4-dimensional taxonomy cube named Tesseract. Tesseract is based on 4 major application performance factors dimensions including: Time complexity, Floating-point operations (FLOPs) to Non-FLOPs ratio, Memory Access Behavior, and Memory Access Frequency. Memory Access Behavior represents type of memory being accessed L1-, L2-, L3-, Shared-, Main-memory and NUMA. Memory Access Frequency represents accesses count per compute operations. The above mentioned factors can be calculated using application such Vtune[17], NVProf[18], Mitos[19], Papi[20], etc. Furthermore, in early stages of development, a pseudo-code can also indicate the aforementioned classification by an informed user.

Tesseract provides an optimal A2A mapping by projecting an algorithm from the algorithm space to architecture in the architecture space. Using the algorithm classification and architecture specification along with the Tesseract, we create a unique one-to-one mapping using qualitative classification of algorithmic factors and architecture functionality. For example, if an application has a high time complexity, good memory access behavior, low memory access frequency and high FLOPs-to-Non-FLOPs ratio, then based on the Tesseract the application would perform optimally on a Single GPU. Therefore, once the algorithm specifications are identified, they are mapped to the Tesseract sub-cubes, which refer the user to an appropriate architecture.

Algorithm	Time Complexity	FLOPs-Non-FLOPs Ratio	Memory Access Behaviors	Memory Access Frequency	Result based Mapping	Tesseract based Mapping
GEM	High	High	Bad	High	$((P,N)toG)$	(P)
LAVAMD	High	High	Good	Low	$((G,P)toG)$	(G)
SWAT	High	Low	Bad	High	(N)	(N)
NW	High	Low	Bad	High	(N)	(N)
SRAD	High	Low	Good	High	(G)	(G)
CFD	Low	High	Bad	High	(G,N)	(P)
HMM (Vary Observation)	High	Low	Good to Bad	High	$((G,N)toP)$	$(GtoP)$
K-means (Vary Data Size)	High	High	Good	High	$((G,P)toG)$	(G)
K-means (Vary Max Cluster)	High	High	Good	High	$((G,P)toG)$	(G)
K-means (Vary Dimension)	High	High	Bad	High	(G,P)	(P)
LUD	High	Low	Good	High	(G,N)	(G)
SPMV (Vary Data Size)	High	High	Good	Low	$((G,N)toG)$	(G)
Triad	Low	High	Good	High	(G,P)	(G)
Reduction	Low	High to Low	Bad	Low	(G)	(P)
FFT	High	High	Good	Low	$((G,P)toG)$	(G)
SCAN	High	Low	Good	High	$((G,P)toG)$	(G)
Stencil2D	Low	High	Good	High	(G)	(G)
TDM	High	Low	Bad	High	(N)	(N)
CRC	Low	Low	Good	Low	(N)	(N)
Radix Sort	High	High	Bad	High	(G)	(P)
BFS	High	Low	Bad	High	$((G,N)toG)$	(N)
A*	High	Low	Bad	High	$((N,P),N)$	(N)

Table 1: Algorithm Classification and Mapping to Single (G)PU, Single (P)hi and Single (N)ode

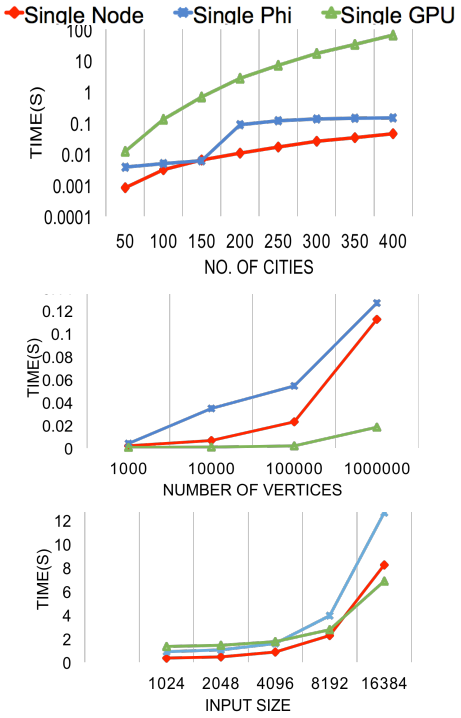


Figure 3: Evaluation results for 3 of 20 applications [Astar(top), Graph-traversal(mid) and Needleman-wunsch (bottom)] applications on Single Node, Single GPU and Single Phi.

FP Framework

FP is a C based framework designed to utilize heterogeneous architectures efficiently by effective resource management. Once user in the user space library defines the appropriate architecture for a scientific application, FP allows for seamless switching to the optimum architecture (if its infrastructure exists) while performing data streaming. Thus FP allows for utilizing resources as per their availability with minimum intervention from the main application. The user can also specify multiple post-processing tasks, along with their priorities and dependencies, where the FP daemon launches these post-processing tasks based on availability of resources. In addition to standard post-processing services (called FP services), the user can provide more specialized FP services that can be a part of a complex work-flow. While using the library, the application simply provides the data structures that need to be operated on by these services and the FP daemon detects the optimal time and available compute resources to work on those services. FP has three major calls, FP_Init() to initialize a node local daemon, FP_Post() and FP_Pull() to push and pull a requested data to daemon and FP_Finalize() to clean up. In order to transfer data between application and node local FP daemon we utilize Common Communication Interface(CCI) [23] which abstracts communication protocol and provides a simple, portable and scalable API for high performance communication. For current node local communication CCI utilizes the shared memory protocol to deliver high bandwidth and low latency communication. We plan to extend our framework to include cross node communication, which allows us to utilize transport layer abstraction for guaranteed high throughput.

A2A Results

In order to map a user application based on the Tesseract, we examine the algorithm characteristics for a given problem set. This characterization is performed by classification of the given application based on four dimensions: time complexity, FLOPs-to-Non-FLOPs memory access behavior, and memory access frequency. We evaluate the performance of Tesseract using 22 algorithmic micro- benchmarks belonging to different classes of algorithms. We utilize these algorithms from various sources such as OpenDwarf

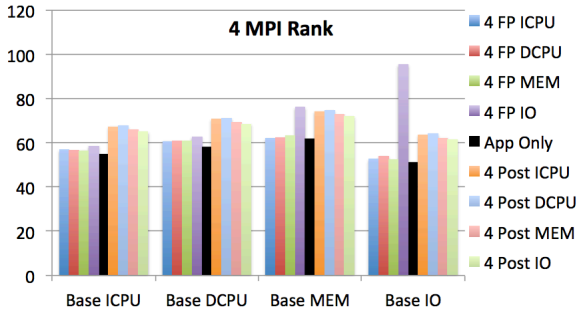
Benchmark [12], Rodinia Benchmark [13] and Scalable Heterogeneous Computing Benchmark Suite (SHOC) [14]. These benchmark belong to following categories: Linear Algebra, Divide and Conqueror, Grid, N-Body, Logic, Dynamic Programming, Branch and Bound, Graph Models, Graph Traversal, and Sorting.

We evaluate these algorithms on 3 accelerator-based supercomputers: Titan at Oak Ridge National Lab, Stampede at The University of Texas at Austin [8], and Palmetto at Clemson University [10]. We use Nvidia GPUs, Xeon-Phis, and Intel E5 on each of respective supercomputer to evaluate Tesseract. Figure 3 shows three of the many application execution on different accelerators.

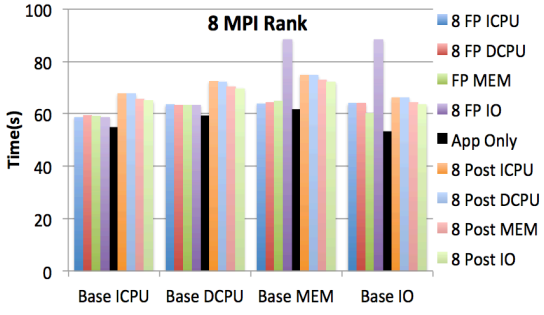
In order to evaluate Tesseract we perform classification of applications as shown in Table I. We define classification based on input parameter and data size. Thus for an increasing size the classification may change. After performing the classification for a given application based on the dimension for a given problem, the algorithm will lie in algorithm space, and can be mapped by referring to Tesseract shown in Figure 1, which is referred to Tesseract based mapping in Table 1. Result based mapping is obtained by evaluating the benchmark applications on different architectures in order to compare experimental results to Teseract based mapping. Some of the experimental results are shown in Figure 3. For classifications that contain transition due to varying input data, we imply 'to' for indicating transition. G, N and P represent Single-GPU, Single-Node and Single-Phi respectively.

FP Results

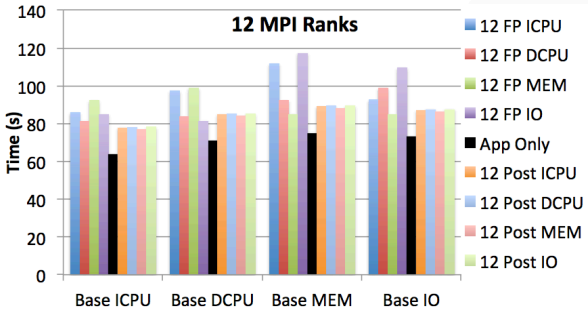
The key research challenge addressed in this work is the minimization of performance variability/jitter for the main application while still achieving high resource utilization using the FP framework. Figure 4, 5, 6 shows the runtime overview of heterogeneous compute node with and without FP on Titan SuperComputer [11], where we run our experiments. Each Titan node has 16 cores and 1 NVIDIA K20 and 32 GB RAM with no on node storage. The 16 cores share 8 FPU units between two cores. We experiment varying number of MPI ranks and also consider combinations of various micro-benchmarks as described below.



(a) 4 MPI Ranks per node



(b) 8 MPI Ranks per node



(c) 12 MPI Ranks per node

Figure 4(Top-left(a), Top-right(b), Bottom-left(c)): Runtime for number of ranks with range of base application (ICPU, DCPU, MEM, IO) and post-processing micro-benchmarks.

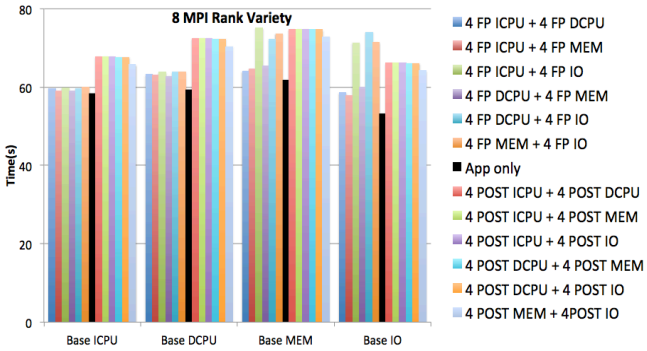


Figure 5(Bottom-right): Runtime for 8 MPI ranks with 4 different client applications (ICPU, DCPU, MEM, IO micro-benchmark) and combination of post-processing micro-benchmarks.

We evaluate the efficiency of the FP framework on the Titan supercomputer by quantifying performance jitter and propose mitigation strategies. We created 4 different types of client and post-processing micro-benchmarks for our experiment: Integer CPU Intensive (ICPU), Double Floating point Intensive (DCPU), Memory Intensive (MEM), and I/O Intensive (IO). The client (baseline) micro-benchmarks are 5 times more time consuming than their corresponding post-processing micro-benchmarks. The baseline ICPU and DCPU micro-benchmark perform 14 billion arithmetic and floating point operations per process. The baseline memory micro-benchmark performs 750 million random read and writes to memory, and the base I/O micro-benchmark performs 225 million write operations to a file.

Figure 4a, 4b, and 4c show the effect of rank scaling per node for various different types of micro-benchmarks combined with different type of clients. Each rank launches at a maximum of one FP task or Post-processing task. We observe for all scenarios i.e. 4, 8, 12 ranks, the run time for application using FP is less than performing corresponding post-processing without FP; except in case of I/O and memory intensive clients. We believe that this is due to the fact that currently all FP tasks are performed in Pthreads, which decreases the performance of I/O and memory reads/writes. Furthermore, due to absence of local node storage, there is an increase in network and memory activity to buffer small writes before writing to disk. Thus, performance of I/O is dependent on network bandwidth and the memory bandwidth of the node, which can be limited when shared with the client application.

In Figure 4c, we observe degradation in performance due to over-subscription. Since, there are only 16 core, we are launching 12 additional processes to the 12 base processes. There are 24 processes spread across the 8 non-binded core leading to degradation in performance due to oversubscription. The performance is worse for cases involving I/O and MEM for the same reason as stated above. Furthermore, we conduct another set of experiments that consisted of a combination of micro-benchmarks on a node. The base application consisted of 8 MPI ranks depicted by Figure 5. We observe that even in the combination experiment FP performs better than post-processing except for the micro-benchmarks involving I/O in either as base or post-processing application for the same reason as mentioned above.

Knowledge Implementation on Scientific Application

We utilize the knowledge from our frameworks for the development of three different scientific tools: Global GPU-enabled Gene Network Alignment (G3NA) [15], HPC Enabled Real-Time Remote Processing of Laparoscopic Surgery [21] and Climate Earth Science Model (CESM) [22]. G3NA is a tool for alignment of two co-expression network and visualization using forced directed layout and is implemented on various accelerators and correctly mapped to our prediction of optimal architecture implementation of GPUs. For the second scientific application, HPC Enabled Real-Time Remote Processing of Laparoscopic Surgery, optimality is of utmost important due to time constraint posed by the processing each framework within 30msec, we are using FP framework to offload algorithms to architecture and since most of our algorithms are image processing and A2A predicts the optimal architecture for image processing being GPUs, we utilize the palmetto cluster for optimal performance with the dual GPU node setup. Lastly, we integrated CESM with FP task of re-gridding, we found 5% decrease in overall runtime using FP. The result of CESM with FP is shown below in Figure 6. We see that CESM + FP performs better than CESM + Post-processing task.

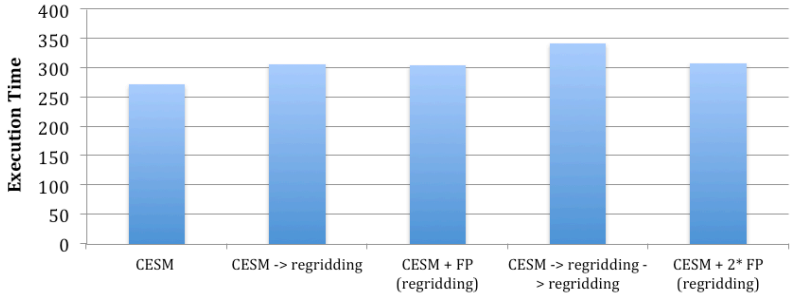


Figure 6: Runtime of CESM on 16 nodes on Titan Supercomputer with FP

Future Work

We are currently working on extending the A2A mapping to a quantitative mapping and qualitative for Multi-node implementation. The qualitative mapping can allow for new algorithms, pseudo code, and applications while the quantitative can provide more accurate mapping for existing applications. The FP Framework is currently being tested with OpenCL, OpenACC, and OpenMP to enable accelerator mapping of task by identification of optimal architecture using A2A mapping.

Conclusion

In conclusion, we present a framework to enable exascale heterogeneous computing using A2A and Functional Partitioning Framework. The A2A framework allows for qualitative mapping for applications to optimal architecture. This optimal architecture can then be used by FP framework for optimal mapping of task to accelerator on heterogeneous computer.

References

[1] C. Grozea and Z. Bankovic and P. Laskov (2011). FPGA vs. Multi-Core CPUs vs. GPUs: Hands-on Experience with a Sorting Application, In Book: Facing Multicore Performance Programming; Elsevier Science & Technology Books, 2013.

[2] M. Bhuiyan (2011). Performance Analysis and Fitness of GPGPU and Multi-Core Architectures for Scientific Applications. Ph.D. Dissertation, Clemson University

[3] Srivastava, R.; Tiwari, T.; Singh, S., "Bidirectional Expansion - Insertion Algorithm for Sorting," Emerging Trends in Engineering and Technology (ICETET), 2009 2nd International Conference on , vol., no., pp.59,62, 16-18 Dec. 2009

[4] Izhikevich EM. Simple model of spiking neurons. IEEE Transactions on Neural Networks 2003; 14(6):15691572.

[5] Morris C, Lecar H. Voltage oscillations in the barnacle giant muscle fiber. Biophysics Journal 1981; 35:193213.

[6] Wilson HR. Simplified dynamics of human and mammalian neocortical neurons. Journal of Theoretical Biology 1999; 200:375388.

[7]G. Juve, A. et.al, "Characterizing and profiling scientific work-flows," Future Generation Computer Systems, vol. 29, no. 3, pp. 682–692, 2013.

[8] Stampede Supercomputer, http://www.tacc.utexas.edu

[9] "Coricrayxc30,"https://www.nersc.gov/users/computational-systems/nersc-8-system-cori/.

[10] Palmetto Supercomputer, http://citi.clemson.edu/palmetto

[11] Titan Super Computer, http://titan.ornl.ccs.gov

[12] "OpenDwarfs benchmark ," https://github.com/opendwarfs.

[13] "Rodinia benchmark," http://www.cs.virginia.edu/ skadron/wiki/rodinia/index.php/Rodinia.

[14] "Scalable Heterogeneous Computing Benchmark Suite (SHOC) benchmark," http://keeneland.gatech.edu/software/keeneland/shoc

[15] G3NA, http://network.genome.clemson.edu/

[16] HPC Enabled Real-Time Remote Processing of Laparoscopic Surgery, Submitted SC15 Poster

[17] Intel Vtune, https://software.intel.com/en-us/intel-vtune-amplifier-xe

[18] NVProf, http://docs.nvidia.com/cuda/profiler-users-guide

[19] Mitos, A. Gim´enez, T. et. al. Dissecting on-node memory access performance: A semantic approach. In International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2014, New Orleans, LA, USA, November 16-21, 2014, pages 166–176, 2014.

[20] Papi, http://icl.cs.utk.edu/papi/

[21]K. Sapra, Z. Ronaghi, R. Izard, David M. Kwartowitz, MC Smith, HPC Enabled Real-Time Remote Processing of Laparoscopic Surgery, In SuperComputing(SC),2015, 20-25 Nov,2015

[22] K. Sapra, S. Gupta, R. Miller, V. Anantharaj, S. Atchley, S. S. Vazhkudai, D. Tiwari, Melissa C. Smith, "End-to-End Computing using Functional Partitioning: A Community Earth System Model (CESM) Case Study", Smoky Mountain Conference, Gatlinburg, TN, Sept 2014

[23] Common Communication Interface, CCI, https://www.olcf.ornl.gov/center-projects/common-communication-interface/