A Multi-Tiered Optimization Framework for Heterogeneous Computing

Andrew Milluzzi, Justin Richardson, Alan George, Herman Lam
NSF Center for High-Performance Reconfigurable Computing (CHREC)
Department of Electrical and Computer Engineering
University of Florida
Gainesville, Florida 32611
Email: {milluzzi, richardson, george, hlam}@chrec.org

***Abstract*—Modern computing nodes often contain more than just a CPU. With the advent of GPU accelerators and Xeon Phi co-processors, there are many architectures available for data processing. However, it is difficult to understand which device is best for a given application. The issue of real-world performance originates in the lack of quantifiable data and method for analysis. This paper presents a novel, multi-tiered framework that leverages Pareto optimization to objectively construct the best processing node for a set of computational kernels. By deconstructing the optimization process into three distinct framework tiers (kernel, device, and system), the system designer is able to understand how the various computational variables impact device choices. We show how we leverage a combination of metrics and benchmarking to form various Pareto sets. Moving through the tiers, these Pareto sets are combined to identify the various combinations that enable maximum performance.**

I. Introduction

The last several years have seen rise to a new breed of hardware accelerators in computational nodes. The rise of GPU and many-core CPU accelerators potentially enables improved system computational performance and higher energy-efficency. Each architecture provides a range of advantages, from the flexibility of a CPU to the massive parallelism of a GPU. However, with all these choices for system construction, it can be a daunting task to obtain the best performance. While device vendors provide case studies of achieved performance, it can be often difficult to understand how an application maps to a given device.

In order to better compare devices, Williams et al. [1] and Richardson et al. [2] present a methodology of comparing theoretical device performance. While these metrics create a fair comparison for maximum performance, not all applications can map well to a given architecture. Targeted benchmarking can be used to better analyze each device; however, this approach is slow and requires access to the device. Furthermore, common synthetic and micro benchmarking rarely portrays how various kernel-level benchmarks will function in a more complex application. These assumptions of ideal execution common in micro benchmarks leave significant room for speculation on application-level and computational system-level performance. To more effectively aid in analysis, this paper will present a three-tiered analysis framework. Each tier of the framework provides analysis at key abstraction points: kernel, device, and system. By relating implementation performance, the kernel level can be used to more clearly demonstrate individual device architectures. The device level presents a series of tradeoffs in order to select a device that best serves a set of computational kernels. Lastly, the system level shows the high-level tradeoffs on application performance. The framework leverages a combination of metrics and benchmarking to form various Pareto sets. Moving through the tiers, these Pareto sets are combined to identify the various combinations that enable maximum performance.

978-1-4799-6233-4/14/$31.00 © 2014 IEEE

II. Related Research

The framework builds on previous work leveraging the combination of Pareto-optimal sets. Pareto optimization is one form of multi-objective optimization that forms a front of points that are the optimal in respect to all constraints. In a Pareto front, points may favor all or a subset of constraints, such that it could potentially include the extremes of fully satisfying one objective while completely failing another. [3] The Pareto optimal form of analysis is well established in device comparisons as outlined in Wulf et al. [4]. Note that in [4] the authors make special point of the complexities of system design. This approach is of particular interest as the combination of these fronts is a fundamental aspect of the proposed framework. Wettergren et al. [5] addresses this issue directly in the tradeoff of Pareto fronts for undersea sensors with respect to cost, type, and performance.

When a system introduces multiple constraints that must work together, Wettergren et al. [5] outlines an approach to methodically combine these Pareto fronts to achieve better performance. A review [5] shows that while a Pareto front is easily calculated for both long-range and short-range sensors, it can vary greatly with the addition of just one parameter. This analysis provides the foundation of the proposed framework, evaluating how the addition of various devices can impact the system Pareto front.

The driving factor used in this framework is benchmarking data. However, benchmarking is a labor-intensive process. In order to expand the applicability of this framework beyond existing devices and kernels in our benchmarking database, we will supplement benchmarking data with device metrics analysis. Williams et al. [1] introduces a set of theoretical metrics that can be used to objectively evaluate disparate devices. These metrics are calculated with respect to clock frequency, power, and the device architecture. One such metric

TABLE I. CD OF DEVICES STUDIED [6]–[10]

Device	Int8 *(GOPS)*	Int16 *(GOPS)*	Int32 *(GOPS)*	SPFP *(GOPS)*	DPFP *(GOPS)*
Intel Xeon E5-2670	998.40	499.20	249.60	332.80	166.4
Intel Xeon Phi 5110P	1074.06	1074.06	1074.06	1074.06	568.62
NVIDIA K20	1762.18	1762.18	1762.18	1762.18	587.39
NVIDIA K20X	1967.60	1967.60	1967.60	1967.60	655.87
NVIDIA K40	2145.60	2145.60	2145.60	2145.60	715.20

is Computational Density (CD) which is presented in giga-operations per second (GOPS) and looks at device, not mathematical, operations. For example, a fused multiply-accumulate operation, commonly found on modern processors, is only 1 operation according to CD, while mathematically it is 2. CD assumes that all data required for computation is available to functional units. These metrics do not consider the time it takes for memory to move data around the processor or to external memory. [2] Table I contains the 8-bit integer (Int8), 16-bit integer (Int16), 32-bit integer (Int32), single-precision floating point (SPFP), and double-precision floating point (DPFP) CD for the devices studied in this paper: Intel Xeon Phi 5110P, Intel Xeon E5-2670, and NVIDIA K20, K20X, and K40.

These metrics provide a first-order analysis of the devices of interest. For example, from Table I we see that the NVIDIA K40 is the highest-performing device. While in many cases, the K40 will outperform other devices, much of the K40's GOPS are a result of the 2880 CUDA cores. It, like the Intel Xeon Phi 5110P, suffers from a slower clock rate than the Intel Xeon E5-2670 CPU. This difference in clock speed would suggest that an application with limited parallelism could perform worse on the NVIDIA K20X than the Intel X5-2670 CPU.

Building on the theoretical nature of CD, Richardson, et al. [11] presents the concept of Realizable Utilization (RU). Like CD, RU looks at computational operations; however, it seeks to expand on CD by analyzing the actual amount of computation performed by the device. Memory overhead and other operations can detract from theoretical system performance, making it extremely difficult to achieve the CD. These real-world results create the RU score that seeks to find what percentage of the device's computational power is actually used for a given application. Richardson et al. [11] draws specific attention to how the RU score can vary with a developer's understanding of an application as well as implementation. For example, a poor implementation, while computationally correct, will likely result in less than optimal performance.

III. APPROACH

In order to address the complexity involved in a computing system, a clear scope must be established. Figure 1 presents a concept diagram for processing constraints of the proposed framework. In order to provide flexibility and effective comparisons, the framework is comprised of three tiers: System Configuration, Device Performance, and Kernel Implementation. The goal of each tier is to provide the system designer with the ability to control constraints while better understanding their impact on the system. As shown in Figure 1, each tier consolidates the data from the tier beneath it. One issue in presenting the final Pareto set is abstraction from device data. By constructing a three-tiered framework, the various levels of optimization can be more transparent. This approach enables the fine-grained control required in designing a computing system, while presenting the data in an organized manner: each tier presents a logical break in order to enable system analysis. In addition to building on lower tiers, the System Configuration and Device Performance tiers also add mission constraints into consideration.

A. Kernel Implementation

Code implementation and optimizations can have a large impact on device performance. From basic C math library implementations to advanced tuning libraries, such as the Fastest Fourier Transform in the West (FFTW). Finding optimal implementation performance is the foundation of the framework. This analysis happens in the Kernel Implementation tier. Inputs to the tier consists of a database of benchmarking data for a sampling device. The Kernel Implementation tier outputs a Pareto front of the optimal implementations for each kernel of interest on each device studied as seen in the callout in Figure 1. In order to compare various device implementations, this tier looks at the Pareto front of benchmarking for multiple implementations of a given computational kernel. The Pareto front is maximized towards performance.

To illustrate this tier, consider the FFT implementation data presented in Figure 2. Each point on graph represents the average of 1000 trials at each dataset size. FFTW, a library that tunes for the architecture, results in higher performance than the Intel Math Kernel Library (MKL); however, it does not use the same implementation at each point. In some cases, serial FFTW code performs better than FFTW code leveraging OpenMP parallelism. Therefore, the Pareto front for this dataset would be a mix of serial and OpenMP parallel FFTW FFT implementations over this range in dataset size.

Note that for this example (for simplicity), the Pareto front is a special case that optimizes only one objective (performance). In general, multiple objectives can be optimized at the Kernel Implementation tier; e.g. non-recurring engineering (NRE) cost and memory overhead. For example, a high NRE investment could result in an innovative approach to computational kernels as seen in Barhen et al. [12] with respect to corner turns in FFTs on the IBM Cell. While currently the Kernel Implementation tier is focused on performance, it can be further expanded to account for development time in the form of NRE cost. These constraints will enable a system designer to consider both implementation and development time while finding the optimal implementation for their needs.

B. Device Performance

Device parallelism and clock speed are two of the greatest factors of CD performance. Most applications see performance boosts in clock speed, while only parallel applications see a boost as core count increases. However, there can be a limit to the benefits of device parallelism. If you have a small dataset, the overhead of communication can cause performance to suffer. In addition, for a small dataset, you might not saturate the device. For example, the NVIDIA K20X GPU has 2688

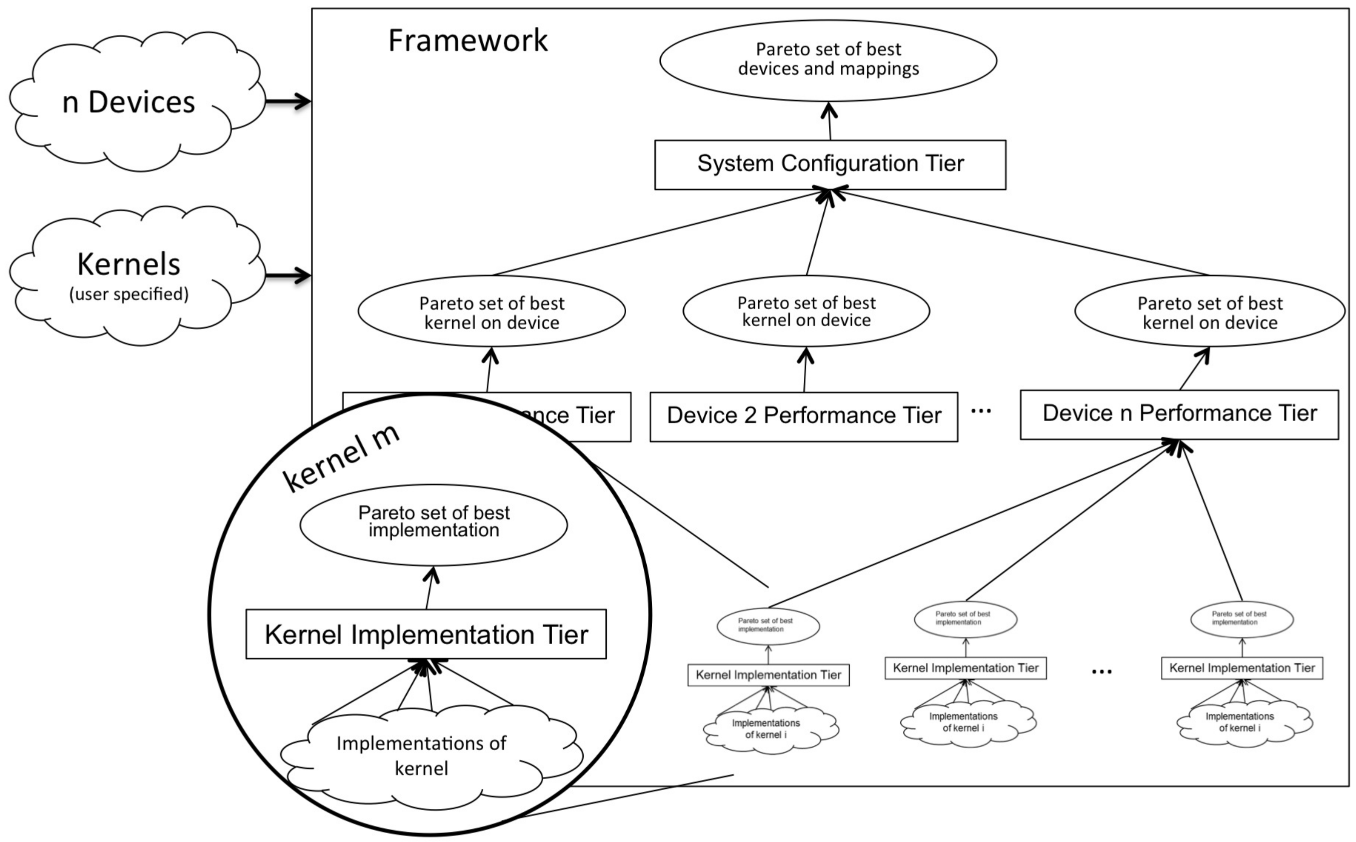
n Devices
Kernels
(user specified)
Framework
Pareto set of best devices and mappings
System Configuration Tier
Pareto set of best kernel on device
Pareto set of best kernel on device
Pareto set of best kernel on device
nce Tier
Device 2 Performance Tier
...
Device n Performance Tier
kernel m
Pareto set of best implementation
Kernel Implementation Tier
Implementations of kernel
Pareto set of best implementation
Kernel Implementation Tier
Implementations of kernel i
Pareto set of best implementation
Kernel Implementation Tier
Implementations of kernel i
...
Pareto set of best implementation
Kernel Implementation Tier
Implementations of kernel i

Fig. 1. Concept Diagram showcasing the three-tiered approach to optimization

CUDA cores. [9] Consider a square matrix multiply on a dataset of only 256 elements. The limited parallelism and overhead in moving the memory to the GPU might make it less efficient that just using the parallelism of the CPU and any SIMD units it might have. The Device Performance tier seeks to address this issue by finding the optimal kernel at a given data set size while also considering other applications constraints such as power and reliability.

While the Kernel Implementation tier filters the out the non-optimal implementations of each kernel, the Device Performance tier seeks to figure out which kernel is best for a given device. This analysis can be seen in the middle level of Figure 1, building from the data (Pareto sets) from the Kernel Implementation tier. The input to the Device Performance tier is the Pareto set of implementation for each kernel and the tier outputs the Pareto set of optimal performing kernels at each dataset size. For example, consider the various Kernel Implementation DPFP outputs for the NVIDIA K20X presented in Figure 3. In comparing the performance of these kernels, it becomes evident that the K20X GPU shows its best performance for 1D FFT, 2D FFT, and Matrix Multiplication. In order to gain optimal efficiency from our system, it is key for each device to run its optimal-performing kernel.

Again, although this example demonstrates optimization based on only performance, this tier can be expanded to include power, reliability and other constraints. For example, by leveraging another metric from [1], Computational Density per Watt (CD/W) and any power data gathered during benchmarking, this tier can enable a more in-depth analysis on performance per Watt. Additionally, the CD/W metric can be used in place of the Kernel Implementation tier data when benchmarking data is not available. Should a level of fault-tolerance be required, this tier can also serve as one point to analyze the impacts of different software approaches. With a performance estimate by either benchmarking or metric analysis data, we can gain insight to added computational redundancy or encoding of data. However, a hardware approach to fault-tolerance, such as triple modular redundancy (TMR), would be factored into the System Configuration.

C. System Configuration

As shown as child nodes of the System Configuration tier in Figure 1, the inputs to the System Configuration tier are

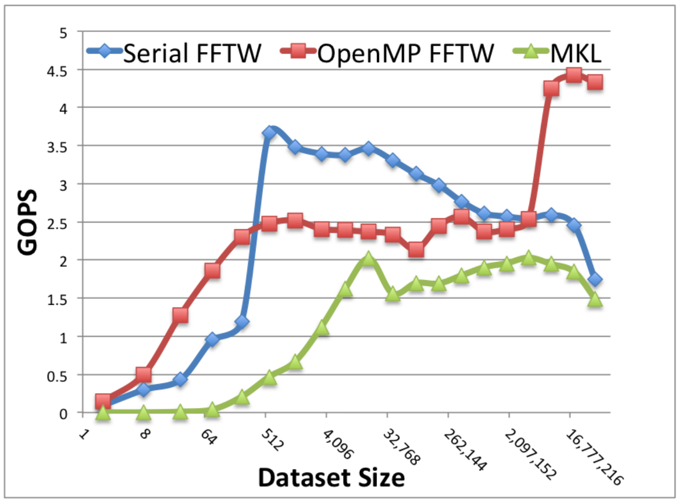

Fig. 2. DPFP 1D FFT benchmarking results on Intel Xeon E5-2670

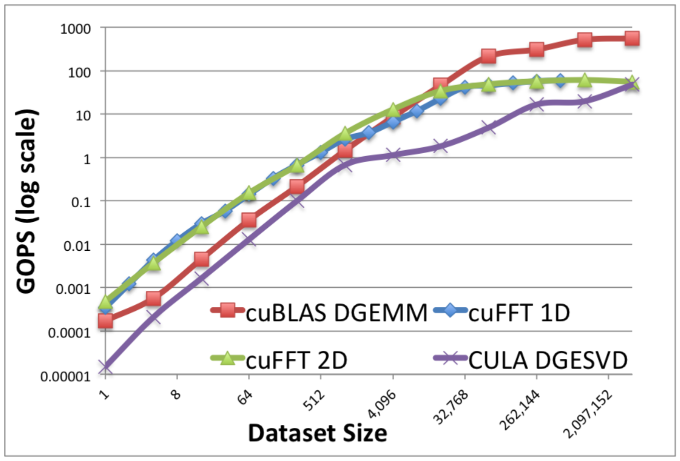

Fig. 3. DPFP FFT, Matrix Multiply, and SVD implementation Pareto fronts on NVIDIA K20X GPU

both the optimal kernel implementations from the Kernel Implementation tier and optimal kernel mappings from the Device Performance tier. The System Configuration tier generates an optimal computational system and kernel mappings. This top tier is the most abstracted from implementation details and gives a high-level overview of system construction. Much of the data for this system-level Pareto front come from the lower tiers. The System Configuration tier is also the most flexible, allowing for data to be based on RU-extrapolated device metrics, or the results of Device Performance tier. In examining its function and goal, the System Configuration tier is similar to the processing framework proposed by Wulf et al. [4] with different optimization constraints.

In order to perform the analysis on the system, the System Configuration tier must process the Pareto fronts generated from the Device Performance and Kernel Implementation tiers. Figure 4 is an example of the Pareto fronts generated from the various kernel implementations presented in Figure 3. By comparing system execution goals to Pareto-fronts, the framework can find the most computationally efficient configuration.

The System Configuration tier requires the data from all devices being considered. However, in many cases it is impractical to have access to a full family of devices. Applying the concept of RU presented by Richardson et al. [11], we can extrapolate performance levels for additional devices. For example, the NVIDIA Kepler GK110 architecture can exist in several forms. [9] These can range from a single SMX as found in the Tegra K1, to multiple SMX units found in the K20, K20X, and K40 accelerators. Since the architectures only computationally vary by degree of parallelism and clock speed, we can extrapolate performance between devices. This estimation is key in expanding the framework to the large set of devices. Figure 5 shows the extrapolated Pareto fronts for two other members of the NVIDIA Kepler family of accelerators, K20 and K40. This graph is formed by calculating the RU of each kernel at each size and applying that to the CD for the other devices in the family. These projected values give much more detail than just raw CD, but still contain some error. In some cases, the difference is small, as seen in the smaller dataset sizes presented in Figure 5 and error is negligible. For larger dataset sizes, there is more variance between the projected and estimated values. The multi-tiered approach to this framework allows the designer to go back and analyze this and decide the impact for their own system.

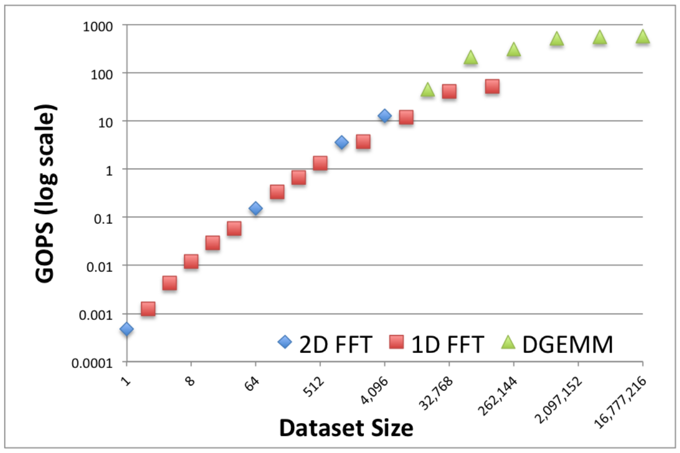

Fig. 4. NVIDIA K20X Device Performance teir Pareto fronts

Sometimes an application requires a kernel with a dataset size that is never on the Pareto front for a device. In these cases, the System Configuration tier directly compares the outputs of the Kernel Implementation tier and available system resources. Figures 6 and 7 present DPFP SVD performance and DPFP square matrix multiplication across all 3 devices of study, respectively. For almost every dataset size in Figure 6, the Intel Xeon E5-2670 performs the best. However, there is a point where the NVIDIA K20X shows slightly better performance. Depending on application requirements and device availability, the System Configuration tier may map the SVD kernel to either the Xeon CPU or K20X GPU. Figure 7 presents a very different picture, showing that each device is the optimal performer at different sizes. Like all the other graphs in this paper, each point represents 1000 trials at the given size. This average is key in ensuring the framework is computing repeatable results.

In working with Pareto fronts, it is possible to have an optimal combination of hardware accelerators for a device

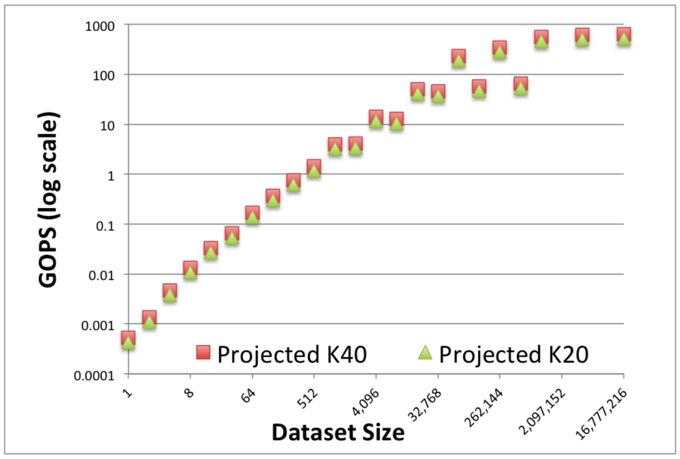

Fig. 5. Projected NVIDIA Kepler family of Device Performance teir Pareto fronts

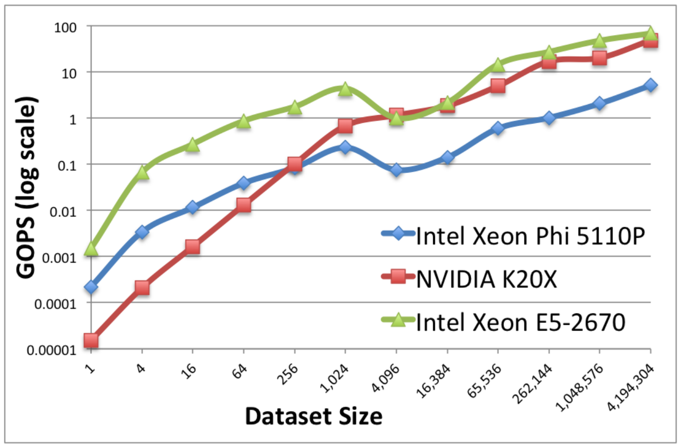

Fig. 6. DPFP SVD kernel perofrmance on Intel Xeon E5-3670, Intel Xeon Phi 5110P, and NVIDIA K20X.

without a host. This strict approach poses a unique challenge and most accelerators require a host in order to operate. In addition to finding the Pareto-optimal front of system devices, this tier also ensures that a device capable of being a host CPU is in the various system combinations.

IV. Case Study

This multi-tiered framework relies on two main sets of data: device metrics and benchmarking results. Device metrics are easy to obtain and only require a datasheet or vendor tools. [2] Benchmarking allows for much deeper analysis, but requires a significant time investment. This case study demonstrates how we leverage a combination of metrics and benchmarking to form various Pareto sets. Moving through the tiers, these Pareto sets are combined to identify the various combinations that enable maximum performance.

In this case study, assume that we want to design a system to perform signal processing in the frequency domain. Based on project constraints, the candidate set of devices are Intel Xeon E5-2670 CPUs, NVIDIA K20, K20X, or K40 GPUs, and Intel Xeon Phi 5110P accelerators. The application does many 4096-element 2D FFTs and SVDs. The application also does

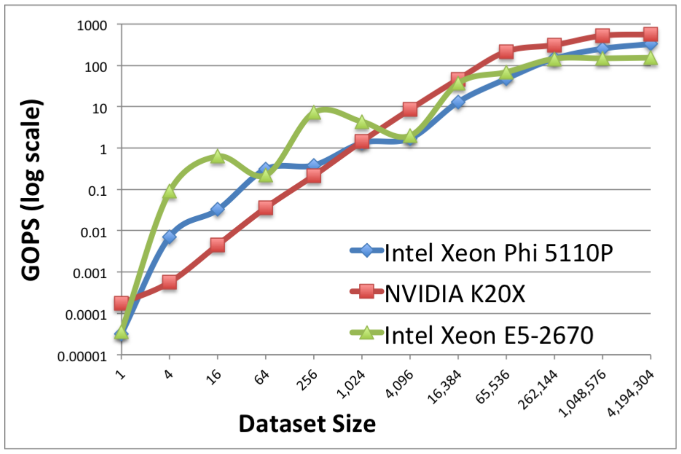

Fig. 7. DPFP Matrix Multiply kernel perofrmance on Intel Xeon E5-3670, Intel Xeon Phi 5110P, and NVIDIA K20X.

1024-element square matrix multiplication. All these kernels are pipelined and can execute in parallel. Given these inputs, the result of the System Configuration tier Pareto front (of the two top solutions) is shown in Tables II and III. These tables provide the important values for the system designer: devices, quantity, and kernel mappings. However, as shown in the tables, much of the details of this implementation are not visible at the System Configuration tier. However, because of the multi-tier design of this framework, the system designer can obtain these details by examining in a transparent manner the results of the Device Performance and Kernel Implementation tiers.

In this case, at first glance at Tables II and III the results seem strange. GPUs are typically known for their matrix multiplication performance. It is odd that neither output set shows matrix multiplication on the GPU. To investigate this counter intuitive result, the Device Performance tier enables review of underlying kernel performance data. Figure 6 shows that in most cases, SVD should be executed on the CPU; however, this is one of the few cases that the NVIDIA K20X, and by proxy the K40, showed better performance. The matrix multiplication kernel is also of concern. GPU accelerators are known for their matrix-multiplication performance; however as you can see in Figure 7, Xeon Phi and Xeon E5-2670 devices outperforms the K20X on dataset sizes smaller than 1024 elements. Upon comparing these results, the output of the

TABLE II. Top Performing System Configuration tier Output

Device	Kernel and Dataset Size
Intel Xeon E5-2670	Matrix Multiply (1024)
NVIDIA K40	2D FFT (4096) SVD (4096)

TABLE III. 2nd Highest Performing System Configuration tier Output

Device	Kernel and Dataset Size
Intel Xeon E5-2670	Matrix Multiply (1024)
NVIDIA K20X	2D FFT (4096) SVD (4096)

System Configuration tier is clear. Even if the performance of the K40 was evaluated, 109% of the K20X, the performance still falls short of the Intel Xeon E5-2670. Furthermore, these comparisons only consider execution time; they do not consider the overhead of moving the data to and from the accelerator, giving the CPU an even greater performance gap.

With multiple outputs and an understanding of each step of the process outlined in Figure 1, the system designer can make choices based on other factors. For example, if performance is your only concern, the designer might consider the results in Table II. However, there is a significant price difference between NVIDIA GPUs, and giving up a bit of performance, you might consider the more economical results in Table III.

This example shows the basic operation of the framework with respect to device performance. The optimizations at each tier present a single dimension of analysis. However, from this example it is clear how power, development time (NRE), device cost, size, memory overhead, and weight will fit into the framework. The logical decomposition of analysis presented with this example enables a system designer to look at the impact various constraints have on the system. Furthermore, this example shows the power of RU and CD in extrapolating metric results to a family of devices, a key requirement in effectively evaluating system configuration options.

V. Conclusions

Designing the optimal computational node is a balancing act between many constraints. It is often hard to truly understand how a given application will map onto various devices. This paper presented a novel framework that leverages performance metrics and benchmarking data to achieve the optimal system configuration with respect to computational performance, and can be extended to size, NRE, reliability, and power.

The example presented in this paper explores the workings of the framework in the construction of a system. It begins with looking at optimal-performing implementations. Leveraging implementation performance data, the framework finds the most efficient use of system devices. The framework then brings the optimal sets into the scope of application goals, presenting the highest performing kernel mappings and system configurations.

Once an application is decomposed into key computational kernels, this multi-tiered framework provides a method of analysis that seeks to show quantitative results at all levels of system configuration. The Kernel Implementation Tier provides the insight into optimal coding style and implementation details for a given computational kernel. By combining and evaluating performance of various kernels on a given device, the Device Performance Tier identifies the strength of each processor. This data, combined with the implementation results, can improve design choices over metric data alone. Finally the Device Performance and Kernel Implementation Tiers all work to achieve optimal performance in the System Configuration Tier.

Future work would include expanding computational kernels and devices. While CD can fill in some gaps, expanded benchmarking is key to gaining a more complete framework at the node level. As a result of staying within the CD domain, existing benchmarking data does not contain the time required to copy memory from the host to accelerator. Future work could leverage the memory metrics presented by Richardson et al. [2] and integrate additional memory benchmarking. Memory and network data could augment the mapping of kernels by comparing if the performance gain outweighs the added complexity and overhead of moving to a coprocessor or GPU. This idea of data-transfer speed would also be instrumental in analyzing cluster configurations that leverage Gigabit Ethernet, InfiniBand, PCIe, Thunderbolt, Fiber Channel or other future interconnects. These additional considerations are key in expanding the usability and effectiveness of the current framework.

Acknowledgments

This work was supported in part by the I/UCRC Program of the National Science Foundation under Grant Nos. EEC-0642422 and IIP-1161022. The authors would like to thank Kevin Cheng, Aishwarya Dhandapani, and Ishan Dalal of the NSF Center for High-Performance Reconfigurable Computing at the University of Florida for assisting with the calculation and confirmation of the CD numbers presented.

References

[1] Williams, J., George, A., Richardson, J., Gosrani, K.,Massie, C., and Lam, H., *Characterization of Fixed and Reconfigurable Multi-Core Devices for Application Acceleration*, ACM Transactions on Reconfigurable Technology and Systems (TRETS), Vol. 3, No. 4, Nov. 2010, pp. 19:1-19:29

[2] Richardson, J., Fingulin, S., Raghunathan, D., Massie, C., George, A., and Lam, H., *Comparative Analysis of HPC and Accelerator Devices: Computation, Memory, I/O, and Power*, Proc. of High-Performance Reconfigurable Computing Technology and Applications Workshop (HPRCTA), at SC'10, New Orleans, LA, Nov. 14, 2010.

[3] Zitzler, E., Thiele, L., *Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach*, Evolutionary Computation, IEEE Transactions on, vol.3, no.4, pp.257-271, Nov 1999

[4] Wulf, N., George, A., and Gordon-Ross, A., *A Framework to Analyze, Compare, and Optimize High-Perofrmance, On-Board Processing Ssytems*, Proc. of IEEE Aerospace Conference, Big Sky, MT, Mar. 3-10, 2012

[5] Wettergren, T., Costa, R., Baylog, J., Grage, S., *Assessing Performance Tradeoffs in Undersea Distributed Sensor Networks*, OCEANS 2006, pp.1, 6, 18-21 Sept. 2006

[6] Intel Corporation *Intel Xeon Phi Coprocessor Vector Microarchitecture*, Apress, 2012

[7] Chrysos, G., *Intel Xeon Phi Coprocessor - the Architecture*, Intel Corporation, 2012

[8] Intel Corporation *Intel Xeon Phi Coprocessor Instruction Set Architecture Reference Manual*, September 2012

[9] NVIDIA Corporation *NVIDIA's Next Generation CUDA Compute Architecture: Kepler GK110*, Version 1.0, 2012

[10] Intel Corporation *Intel 64 and IA-32 Architecture Optimization Reference Manual*, April 2012

[11] Richardson, J., George, A., and Lam, H., *Performance Analysis of GPU Accelerators with Realizable Utilization of Computational Density*, Proc. of Symposium on Application Accelerators in High-Performance Computing (SAAHPC), Chicago, IL, July 10-11, 2012

[12] Barhen, J.; Humble, T.; Mitra, P.; Traweek, M., *Multi-FFT Vectorization for the Cell Multicore Processor*, Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on, pp. 780, 785, 17-20 May 2010