Modeling and Predicting Performance of High Performance Computing Applications on Hardware Accelerators

Mitesh R. Meswani
UCSD/SDSC
La Jolla, CA, USA
mitesh@sdsc.edu

Laura Carrington
UCSD/SDSC
La Jolla, CA, USA
lcarring@sdsc.edu

Didem Unat
UCSD
La Jolla, CA, USA
dunat@cs.ucsd.edu

Allan Snavely
UCSD/SDSC
La Jolla, CA, USA
allans@sdsc.edu

Scott Baden
UCSD
La Jolla, CA, USA
baden@ucsd.edu

Stephen Poole
ORNL
Oak Ridge, TN, USA
Stephen.w.poole@ugov.gov

Abstract— **Computers with hardware accelerators, also referred to as hybrid-core systems, speedup applications by offloading certain compute operations that can run faster on accelerators. Thus, it is not surprising that many of top500 supercomputers use accelerators. However, in addition to procurement cost, significant programming and porting effort is required to realize the potential benefit of such accelerators. Hence, before building such a system it is prudent to answer the question 'what is the projected performance benefit from accelerators for the workloads of interest?'. We address this question by way of a performance-modeling framework that predicts realizable application performance on accelerators rapidly and accurately without going to the considerable effort of porting and tuning.**

The modeling framework first automatically identifies commonly found compute patterns in scientific applications which we term idioms, which may benefit by accelerator technology. Next the framework models the predicted speedup of those idioms if they were to be ported to and run on hardware accelerators. As a proof of concept we characterize two kinds of accelerators 1) the FPGA accelerators on a Convey HC-1 system and 2) an NVIDIA FERMI GPU accelerator. We model performance of the idioms gather/scatter and stream and our predictions show that where these occur in two full-scale HPC applications, Milc and HYCOM, gather/scatter speeds up by as much as 15X, and stream by as much as 14X, whereas the overall compute time of Milc improves by 3.4% and HYCOM by 20%.

Keywords-accelerators;GPU;FPGA;performance prediction; performance modeling;benchmarking;HPC

I. INTRODUCTION

Limitations of traditional general processor architectures have resulted in the emergence of hybrid core computers. These computer systems integrate specialized hardware, called accelerators; to augment the generic CPU and can provide fast computations for certain classes of compute operations. The CPU offloads operations to the hardware accelerators such as GPUs or the FPGAs to perform certain operations that may run faster on these, and this in turn improves application performance.

Accelerators have gained popularity recently and indeed many of the worlds fastest supercomputers listed in the TOP500 list [15] have hardware accelerators. For example, Tianhe-1A supercomputer has 7,168 NVIDIA Tesla M2050 general purpose GPUs.

One of the challenges of hybrid core architecture is the effort required to port existing applications to these new system architectures. Large HPC applications typically have code sizes that exceed 100,000 lines, and porting them can take years of effort. Moreover, while in general an accelerator may benefit an application, the choice of the particular best accelerator device may not be readily apparent. Hence, before undertaking procurement, or time consuming reprogramming effort, one should explore the projected benefits from a given set of accelerator devices on the workload of interest. We show that performance models have a useful part to play in this evaluation and can help answer these questions accurately and speedily.

To predict speedup and evaluate choice of accelerator device, one must first identify sections of existing source code that are potential candidates to run on accelerators. For identification of code sections we can leverage the fact that many HPC applications' computational and data access patterns can be expressed by a small set of commonly occurring idioms [41]; examples of idioms include reduction, transpose, stencil and the like. Hence by identifying idioms one can identify instances of idioms that may run faster on the accelerator. Porting can then simply replace the selected idiom instances in source code to run on the selected accelerator.

In this paper we present our modeling methodology that can predict idiom performance on a given accelerator, and these models are then used to identify and choose idiom instances and accelerator device for porting. To develop prediction models, we first characterize how a particular accelerator device performs idiom operations over a range of data sizes by means of simple benchmarks. Given this information we develop a model that can predict performance of an idiom operation when it occurs in an application.

In particular, in this paper we present characterization studies of two accelerator devices: 1) FPGAs on a Convey HC-1 system [3] and 2) NVIDIA FERMI GPUs [4]. For these two devices we characterize their ability to perform reduction, stream, transpose, stencil, and gather/scatter idioms. Furthermore we develop performance models of gather/scatter and stream idioms for both accelerator devices. We also present two validation experiments to validate the performance models. In the first validation experiment we predict runtimes, with 10% average error, at fine-grained level/loop level of single loops with an idiom in the HMMER [5] application running on the accelerators. In the second validation experiment we predict runtimes, with 1.5% error, of graph500 [8] application when all its idioms that benefit from accelerators are ported to the accelerator device.

Using the performance models we project how two HPC applications, Milc and HYCOM, would benefit running on a hypothetical system like, Jaguar, a Cray XT5, but with the addition of the FPGA and GPU attached to each node. Our predictions show that for Milc and HYCOM, gather/scatter speeds up by as much as 15X, stream by as much as 14X, whereas the overall compute time of Milc improves by 3.4% and HYCOM by 20%.

Identification of idioms and modeling large-scale applications can be labor intensive and error prone. Hence, we also briefly describe the performance modeling framework and the set of tools that can automate the prediction process.

The outline of the rest of the paper is as follows: Section II gives an overview of idioms that are used to characterize applications. Section III describes the two hardware accelerators that were modeled, and their capabilities to compute idioms are shown in Section IV. Section V describes the tools and process used to identify idioms in applications. The PMaC modeling framework is used for performance predictions and this is described in Section VI. Validation of our models and a projection study to calculate speedup of applications on a HPC machine with hardware accelerators in described in Section VII. Finally we discuss related work in Section VIII and conclusions and future work is discussed in Section IX.

II. Idioms

An idiom is a pattern of computation and memory access that may occur within an application. Given a sufficient covering set one may express an application in terms of its constituent idioms. For example, a stream idiom, shown in Figure 1, is used to define a pattern where memory is read from an array and copied to another array. Specifically, the data is read and written sequentially. A stream may arise from the presence of the statement A[i] = B[i] within a loop over i.

```
values[c] = constants[c];
for( i = 0; i < 10; ++i ) {
    item = source_array[i];
    dest_array[i] = item; }
```

Figure 1. Sample Stream Code

Idioms are useful for describing patterns of computation that have the potential to be optimized or sped up, for example, by loading the piece of code to a coprocessor or accelerator.

We have found the following idioms to be common in HPC applications. All of the code samples given here are assumed to be part of a loop, i (and j) are loop induction variables.

- Stream: A[i] = A[i] + B[i]

The stream idiom includes accesses that step through arrays. In the above example two arrays are being stepped through simultaneously, but the stream idiom is not limited to this case. Stepping through any array in a loop where the index is determined by a loop induction variable is considered a stream.

- Transpose: A[i][j] = B[j][i]

The transpose idiom involves a matrix transpose, essentially reordering an array using the loop induction variable.

- Gather: A[i] = B[C[i]]

The gather idiom includes gathering data from a potentially random access area in memory to a sequential array. In this example the random accesses are created using an index array, C.

- Scatter: A[B[i]] = C[i]

The scatter idiom is essentially symmetric to gather idiom. Values are read from a sequential area of memory and saved to an area accessed in a potentially random manner.

- Reduction: s = s + A[i]

A reduction can be formed from a stream, as in the working example, or a gather. It implies that the values returned from the read portion of the idiom are folded into a temporary variable.

- Stencil: A[i] = A[i-1] + A[i+1]

A stencil idiom involves accessing an array in a fixed pattern more complex than a simple sequential manner, including a dependency between iterations of the loop.

- Matrix Matrix Multiply: C[i][j] = A[i][k]*B[k][j]

This is the familiar operation of multiplying two matrices.

- Matrix Vector Multiply: C[i] = A[i][j]*B[j]

This is another common operation describing product of a matrix with a vector.

Defining a complete and usefully distinguishing covering set of idioms for a broad class of applications is an open research problem but the just-described set is useful as shown in this paper.

III. Hardware Accelerators

We study two popular hardware accelerator technologies: FPGAs and GPUs. A Convey HC-1 system with FPGAs is described in Section III.A and a machine with NVIDIA FERMI GPUs is described in Section III.B

A. FPGA - Convey HC-1

The Convey HC-1[3], shown in Figure 2, is a hybrid core computer that closely couples the FPGA based reconfigurable coprocessor with an Intel Xeon Woodcrest processor (Intel 5138). The coprocessor and the host processor share memory and thus reduce the cost of data transfer. The coprocessor can be targeted for different workloads by reloading them with different instruction sets called Convey personalities. The personalities can be written in low level FPGA specific language or use the vendor supplied personalities. These personalities can then be called from C/C++/Fortran source code. In this paper we used the vendor-supplied personalities.

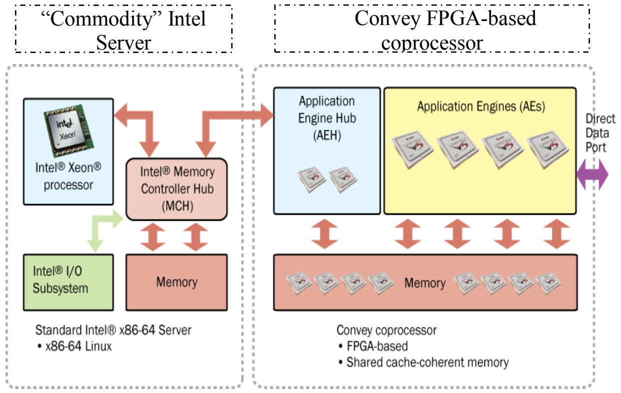

"Commodity" Intel Server
Convey FPGA-based coprocessor
Intel® Xeon® processor
Intel® Memory Controller Hub (MCH)
Intel® I/O Subsystem
Memory
Standard Intel® x86-64 Server
• x86-64 Linux
Application Engine Hub (AEH)
Application Engines (AEs)
Direct Data Port
Memory
Convey coprocessor
• FPGA-based
• Shared cache-coherent memory

Figure 2. Convey HC-1

B. *GPU – NVIDIA FERMI*

The second accelerator we studied was a NVIDIA C2070 (FERMI) GPU [4]. The system studied was composed of three GPUs attached to a machine with dual hex-core Intel Xeon processors (X5680). Shown in Figure 3 is a NVIDIA FERMI GPU. FERMI has 512 cores that are arranged in 16 Streaming Multiprocessors (SM). The GPU is connected to the host machine by PCI express interface and has its own DRAM. In this paper the NVIDIA supplied compiler was used to compile code for the GPU.

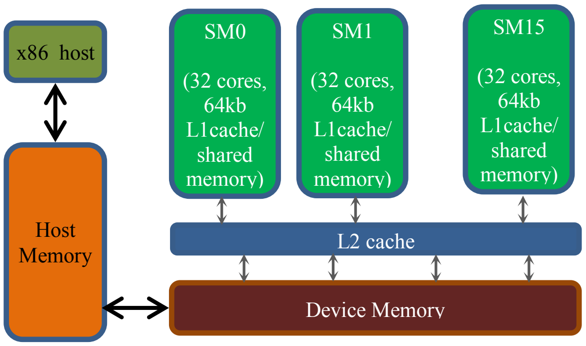
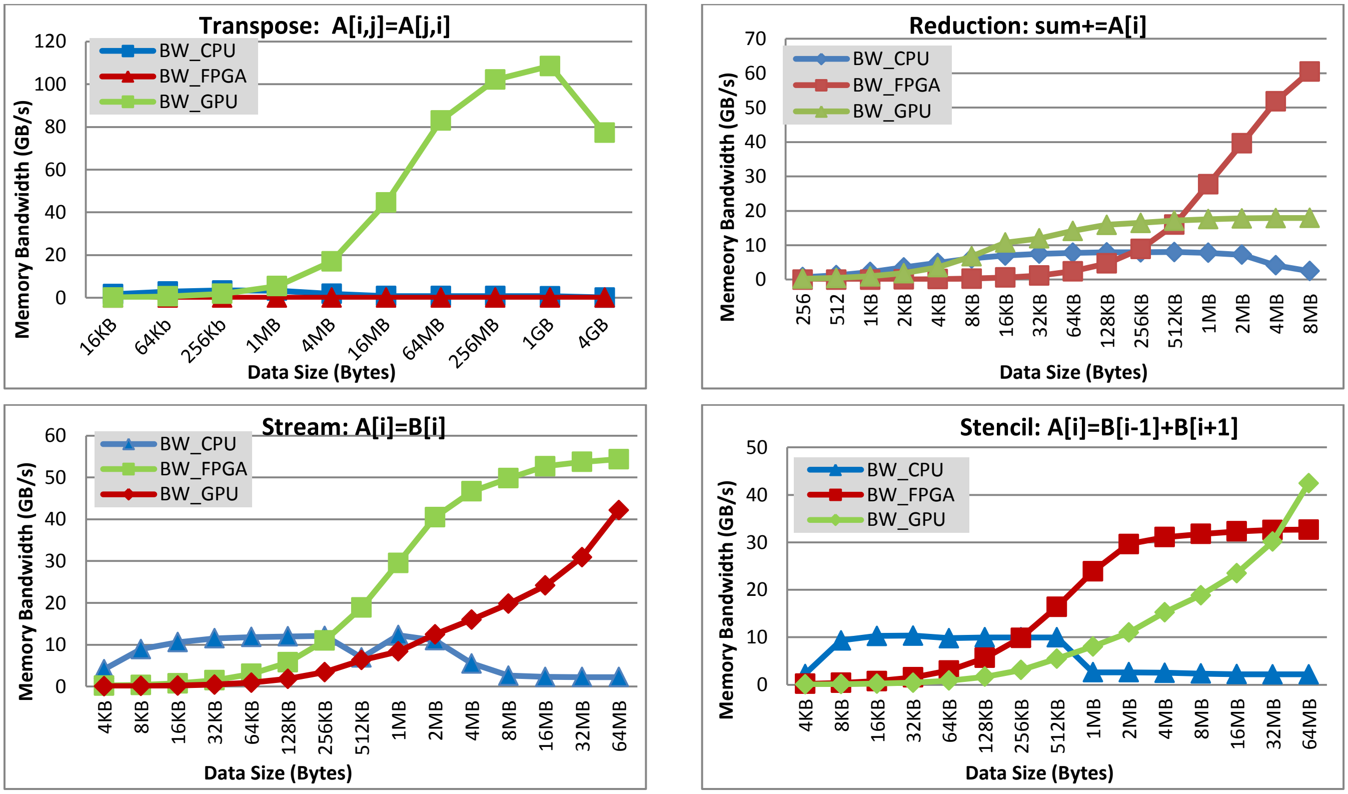

x86 host
Host Memory
SM0
(32 cores, 64kb L1cache/ shared memory)
SM1
(32 cores, 64kb L1cache/ shared memory)
SM15
(32 cores, 64kb L1cache/ shared memory)
L2 cache
Device Memory

Figure 3. NVIDIA FERMI

IV. MACHINE CHARACTERIZATIONS

For this work we developed a benchmark suite that enabled us to profile the rate at which the FPGA, GPU, and CPU can perform idiom operations at different data sizes (range covered by the induction variable expressed in bytes). Each idiom is separately executed by the benchmark at specific data sizes. Prior to timing and execution of the idiom, data is migrated to the memory of the respective device; this enables separate performance models for compute work on the device as well as data migration. Thus, the bandwidths plotted in figure 4, described next, do not include cost of data migration.



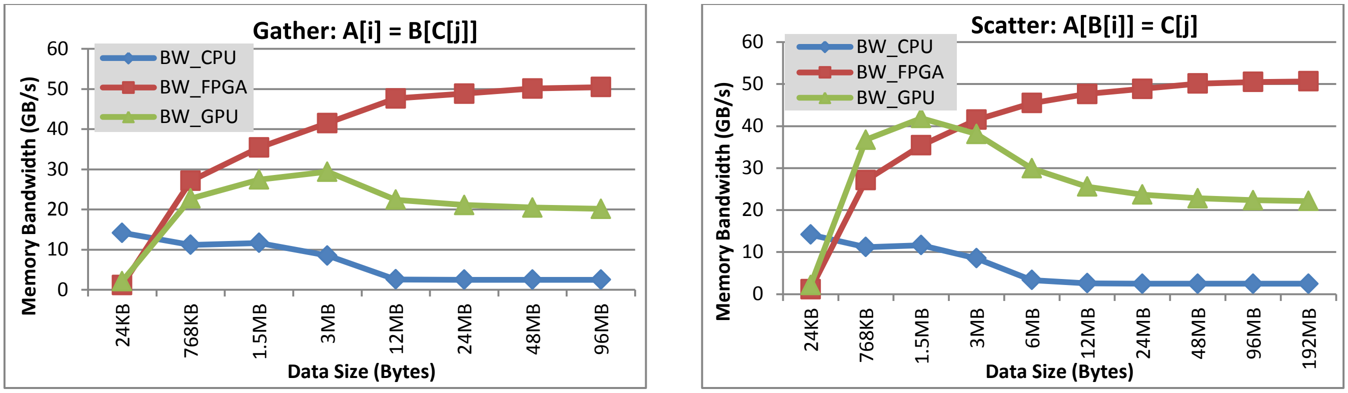


Figure 4. This figure shows memory bandwidths achieved by GPU, FPGA, and CPU for different idiom operations ignoring data transfer costs. Each plot shows bandwidth expressed in GB/s on Y-axis and the data size in bytes on X-axis

Figure 4 shows the bandwidth as a function of data set size for transpose, reduction, stream, stencil, gather, and scatter idioms for a GPU, FPGA, and Xeon CPU. Each plot in this figure shows the achieved bandwidth in GB/s on the Y-axis versus data size in bytes on the X-axis. In all cases, except for the transpose idiom, when data size is in order of Megabytes the hardware accelerator, either the GPU or the FPGA, always performs better than the CPU. For transpose idiom, GPU outperforms both CPU and FPGA, whereas FPGA and CPU performances are comparable. Although we have not confirmed, it is our hypothesis that the compiler used for the FPGA may not be optimized for transpose operation and better performance may be obtained by developing custom transpose FPGA code. For smaller data sizes in many cases the performance of CPU is better or comparable to the accelerators partly due to the cost of setting up the accelerators to perform the operations. Hence, the best choice of the device depends upon the data footprint and idiom.

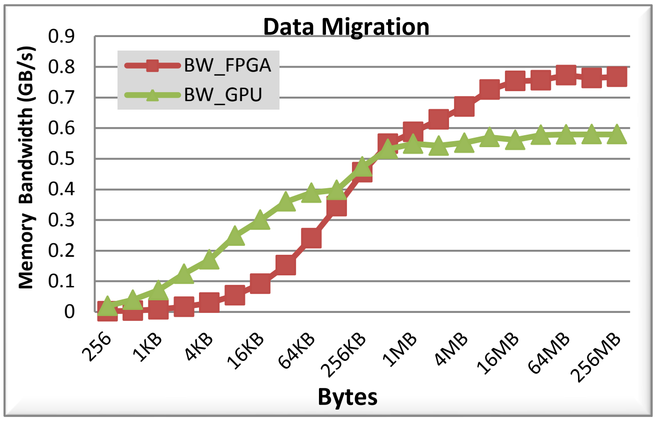


Figure 5. Bandwidth (GB/s) versus Data Transfer Size (Bytes) between CPU and FPGA/GPU

The cost of data migration, i.e., transferring data between CPU and hardware accelerators, can be significant. Shown in figure 5 is the bandwidth achieved for one-way data transfer i.e., to/from CPU from/to hardware accelerators. The cost of data transfers dominates run time for smaller data sizes. For example, we found that, for the stream idiom on the FPGA, if data is transferred to/from the FPGA at begin/end of stream loop, then the FPGA is better than the CPU only for datasets that exceed 2 GB. Thus, to extract maximum benefit from the hardware accelerator, data transfer costs need to be minimized. Combining the results from figure 4 and figure 5 illustrates the complexity of determining the best achievable performance from the GPU/FPGA for a given data size and it is interesting to note this space is complex – there is no clear winner among the CPU, FPGA, GPU it depends on the idiom and the dataset size.

V. Application Characterizations

In order to model and predict how a particular application may benefit by running idioms on accelerator hardware we need to first identify instances of those idioms within the application source code. Furthermore we need to capture the parameters that determine their performance. As shown in Figure 4, data footprint is an indicative parameter that can be used to understand performance. Generally speaking larger datasets benefit to a greater extent by running on accelerator hardware by amortizing the cost of data migration. In this paper we focus on two of the idioms, stream and gather/scatter, to model their performance on GPU and FPGA hardware. The remaining idioms are left as future work.

A. Identifying Idioms

Identification of idioms can be labor-intensive for large HPC codes. To aid in this process, we used an idiom-recognizing tool called PIR (PMaC Idiom Recognizer)[2, 14] to search the source code for idioms. The PIR tool automates the search for idioms in a powerful way by using data-flow analysis during program compilation to augment the identification process.

Table I presents just a sample of the report from PIR. The sample shows how PIR is able to classify the idiom, capture the source file, source line, function name and even the line number of source code used for the identification (additional information about loop depth, start, and end are captured but not shown). For large-scale applications with code base greater than 100,000 lines of code, PIR is a particularly useful tool since it is completely automated.

TABLE I. SAMPLE PIR REPORT

File	Line#	Function	Idiom	Code
foo.c	623	Func1	gather	a[i]= b[d[j]]
tmp.c	992	Func2	stream	x[j]= c[i]

B. *Profiling Identified Idioms*

In order to model the performance of identified idioms on accelerator hardware, we also need to capture the data footprint for each identified idiom. Data footprint depends on data input at runtime; hence we use PEBIL [9], a binary instrumentation tool to capture those parameters. The instrumented binary is executed and data footprint is stored per basic block.

In order to capture data footprint of idioms of interest, the information given by PIR has to be mapped to basic blocks in application binary. As shown in Table I, PIR gives file name and line number for each identified idiom. A special feature of PEBIL was used that prints line number and file name of each basic block. The two outputs are then post processed and the matching results produce a list of basic blocks corresponding to the idioms identified by PIR. This list of basic blocks is then instrumented by PEBIL to capture the data footprint.

VI. PMAC PERFORMANCE MODELING FRAMEWORK

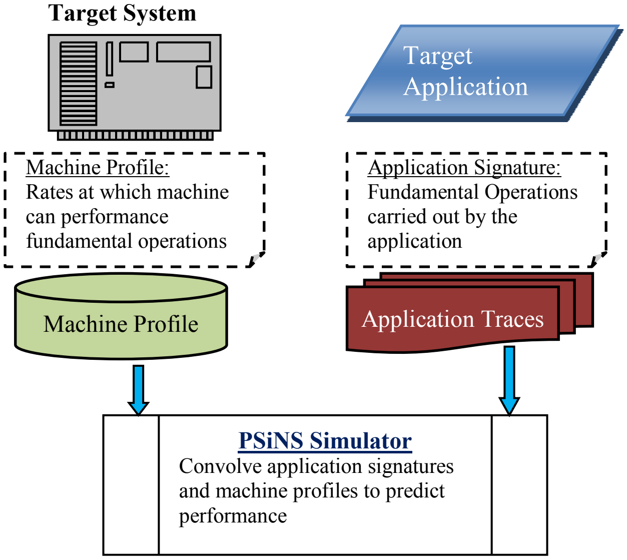

Target System
Target Application
Machine Profile:
Rates at which machine can performance fundamental operations
Application Signature:
Fundamental Operations carried out by the application
Machine Profile
Application Traces
PSiNS Simulator
Convolve application signatures and machine profiles to predict performance

Figure 6. PMaC Prediction Framework

For the purpose of modeling and prediction of accelerators we extended the PMaC performance-modeling framework [11]. This framework is used to provide fast and accurate predictions of large scale HPC application performance. The modeling framework is based on three components: (1) Benchmarks that characterizes how fast a machine can perform certain operations, called Machine profiles, (2) Tracing and simulation tools that gather information about application characteristics and requirements, called Application Signatures, and (3) the convolver, which are methods that predict performance using application signatures and machine profiles. The framework is depicted in Figure 6.

The framework has been designed to accurately predict performance of large-scale parallel applications. For predictions, the framework is composed of a single processor model and a communication model. Each model comprises of an application signature, machine profile, and a convolver that is used for prediction. The two models are incorporated into a simulator, PSiNS [10], which can replay entire application traces and predict runtime for different configurations of network and machine. The models have been used to provide accuracy of within 15% absolute error for prediction of production codes on real-world HPC applications [10]. For a more complete description of the other pieces of the framework, please see Carrington et al.[12] and Tikir et al.[13].

During compute phase the model focuses on the time it takes to move the data through memory hierarchy. Although floating-point operations do take time, often memory access times dominate. A detailed description of memory time is given in Tikir et al. [13]. Equation 1 shows the memory-time of an application as the sum of memory time of all its basic blocks. PEBIL is used to trace memory references and simulate the address stream thorough a cache simulator for the target architecture to obtain the size of memory reference as well as the type (e.g. L1 cache reference, L2 cache reference, …) for all the basic blocks of an application.

$$\boldsymbol{Memory\ Time} = \sum_{i}^{all\ BB} \frac{\#\boldsymbol{Mem\ Ref_{i,j}} \times \boldsymbol{Ref\ Size}}{\boldsymbol{Mem\ BW_j}} \qquad (1)$$

Where,

Mem BW_j = bandwidth of jth type of reference on target system

Ref Size = size of the reference in bytes

$\text{Mem Ref}_{i,j}$ = number of references of basic block i type j

Since we are running parallel applications we also need to capture and model application communication. We use PSiNSTracer [10] to capture traces of MPI calls made by an application. A machine’s ability to perform various MPI functions is captured by simple benchmarks such as ping pong. These traces and machine profiles are then replayed by PSiNS simulator and used for predictions.

Idiom operations are performed during an applications compute phase, and hence, the computational model in the framework was extended for off-loading compute work to the accelerators. In a previous study [14] the framework was extended to predict performance of gather/scatter operations on the FPGA hardware. In this work we extend the framework in two ways: (1) we develop models for the GPU in addition to the FPGA, and (2) we develop models of stream and gather/scatter idioms for the GPU and a model of stream idiom for the FPGA. The model equations are similar in form to equation 1 that calculates memory time when the work is off-loaded to the accelerator hardware.

VII. Experiments and Results

A. Models

1) Prediction Equations

We developed a prediction model for each idiom-device combination. For example, shown in equation 2 is the prediction model used for predicting stream performance on the FPGA. This equation is a piece wise linear fit of the data shown in Figure 4. The modeled bandwidth and measured bandwidth of the FPGA and the GPU for idioms is shown in Figure 7. As shown in this figure the models accurately predict the actual measured bandwidth of both devices.

$$\textit{Memory Time} = \sum_{i}^{all\ BB} \frac{\#\textit{Mem Ref}_{i,j} \times \textit{Ref Size}}{\textit{Mem BW}_{stream}} \qquad (2)$$

Where,

$\text{Mem BW}_{\text{stream}}$ = 13.59*ln(RS) - 159.2: RS <= 4194304,
53.0: RS > 4194304
Ref Size = size of the reference in bytes (RS)
$\text{Mem Ref}_{i,j}$ = # of references of basic block i type j

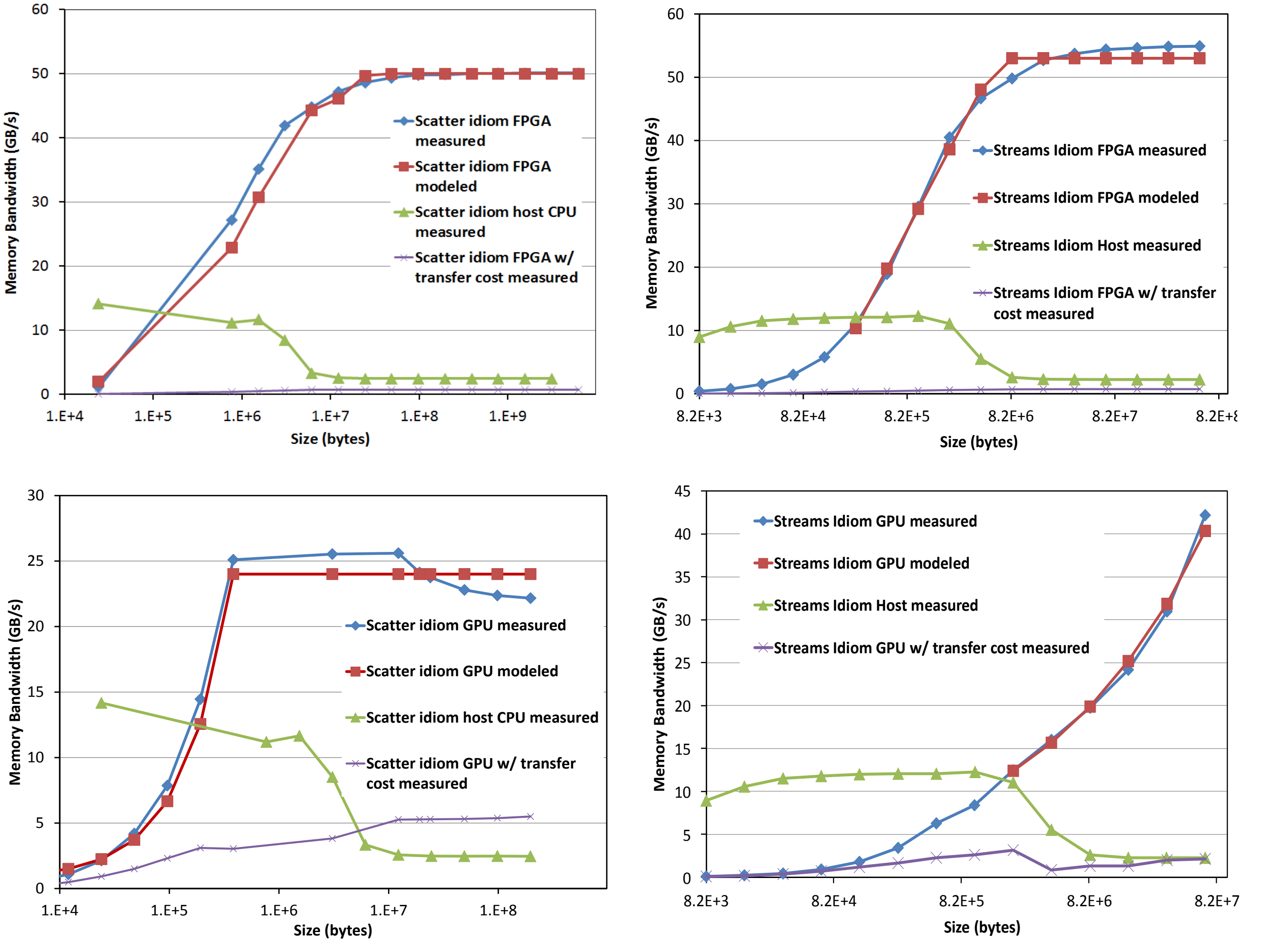


Figure 7. This figure shows, for stream and scatter idioms, measured and modeled memory bandwidths on GPU and FPGA, measured memory bandwidth on host CPU, and measured memory bandwidth on FGPA, GPU with data transfer cost. Each plot shows bandwidth expressed in GB/s on Y-axis and the data size in bytes on X-axis

2) Validation

To validate our accelerator modeling framework we conducted two experiments. In the first experiment we validated the models on a fine-grained level/loop level by porting single loops (i.e. gather/scatter or stream) to the FPGA and GPU and comparing the measured runtime to the predicted. In the second experiment we validate the models accuracy to predict the overall speedup of an entire application when idioms that benefit from accelerators are ported to the accelerator device.

For our first experiment we used the real world application HMMER[5], which is a protein sequence analysis suite and its main algorithm hmmsearch searches an input HMM against a database. In this paper we search the globins.hmm profile in the uniprot_sprot.fasta dataset distributed with the package. This application was run with 8 MPI tasks on both the GPU and the FPGA systems.

HMMER was first analyzed by PIR to search for idioms. PIR reported 666 gather/scatter idioms and 307 stream idioms. We chose the applications largest stream and gather idioms and traced it with PEBIL to identify data footprint of each idiom loop. Next, we measured the run time of the idiom loops on the host CPU. Using our models we obtained predictions for the idiom loops running on the FPGA and the GPU.

To validate our predictions, we ported those loops to the FPGA and the GPU and measured their actual run times. The results of our predictions are shown in Table II.

TABLE II. PREDICTED AND MEASURED IDIOM TIMES ON FPGA, GPU

Idiom	Measured	Predicted	% Error[1]
Stream (FPGA)	384.7	337.0	12.3%
Stream (GPU)	18.4	18.5	0.3%
Gather/Scatter (GPU)	0.074	0.087	17.3%

[1] %Error=abs((Measured – Predicted)/Measured))*100

As seen in this table our models have error rates as low as 0.3% and as high as 17.3%. Note, that we did not port gather loop to the FPGA, which was done in the study by Carrington el al.[14]. In that work error rate was shown to be less than 10% for a gather/scatter loop in FLASH application.

In our second experiment we used the application graph500 [8], which is a popular benchmark created to test HPC systems performance for data-intensive computing. In this experiment we test our models accuracy to predict the speedup of sequential version of graph500 ported to run on the FPGA. We leave porting to GPU as part of future work.

Graph500 consists of two kernels: kernel 1 generates the initial graph, and kernel 2 is used to perform breadth-first search of the graph. The size of the problem is specified as a scale factor, which is used to calculate the number of vertices in the graph as 2scale. Briefly the algorithm proceeds in two steps: it first generates the graph based on the scale factor. In the second step it samples 64 random key and for each key it executes *make_bfs* which a is routine that computes the parent array by performing a breadth-first search starting with the key, and then executes validate routine which validates that the parent array is a correct BFS search tree. Hence, *make_bfs* and validate are executed 64 times. The time for make_bfs routine is used to calculate a performance metric for graph500 called TEPS (travelled edges per second); higher TEPS is better. More information of graph500 can be obtained here [8].

We ran a scale size 24, the largest we could run on our system. At this scale factor, graph500 takes 5980 seconds to finish execution. Of this time it spends 1313.92 seconds for 64 executions of *make_bfs* at a mean TEPS of approximately 13 Million TEPS per execution of *make_bfs*. The *make_bfs* routine spends most of its time in a main inner for loop which has a scatter and stream idiom. Hence, this routine can benefit from the high memory bandwidth that is available in the FPGA. We use a Convey-supplied personality to execute the *make_bfs* computation on the FPGA, while the remaining is executed on the host CPU. With this porting, *make_bfs* speeds up by 98X and, the total run time of graph500 speeds up by 21.64% taking 4686 seconds to finish execution.

Next to predict the speedup of *make_bfs* routine, we use PEBIL to identify and trace the basic blocks of *make_bfs*. PEBIL identifies that the routine's main inner for loop with the scatter and stream idiom are contained in one basic block. Then we use PEBIL to trace the corresponding basic block and determine its data footprint. For purpose of predictions we assume that each idiom operates on half of the basic block's data footprint. Accordingly, the predicted time for the basic block is calculated as the sum of the times using stream and scatter models, where each model predicts time for the half of the basic block's data footprint. To validate our models we use our prediction framework and idiom models to calculate speedup of the *make_bfs* routine and calculate overall speedup of graph500. These results are discussed next.

Our framework predicts that graph500 takes approximately 5847 seconds to execute entirely on the CPU without porting. Our FPGA scatter and stream models predict that the *make_bfs* routine speeds up by 96X on the FPGA, giving an error of about 2% compared to measured speedup of 98X. Next, our framework predicts that the total run time of graph500 ported to the FPGA speeds up by 18.65% taking 4757 seconds to finish execution. Thus, our predicted speed of graph500 is about 3% lower than measured speedup and our predicted run time of 4757 seconds has an error of 1.5% as compared to the measured runtime of 4686 seconds.

B. *Projection Study*

Next we studied a set of full-scale applications running on a production HPC system, Jaguar, a Cray XT5. Jaguar consists of 18,688 compute nodes. Each node contains dual hex-core AMD Opteron processors with 16GB of memory. The resulting system has a total of 224,256 cores and 300TB of main memory. We use this system to capture traces and profiles of large-scale applications. Then, we use our models to calculate speedup for a hypothetical system that is like Jaguar but with FPGAs and GPUs attached to the compute nodes. For this study we assume that when an operation is running on the accelerator, the CPU is blocked waiting for the device to complete. Although additional speedup may be obtained by overlapping CPU and accelerator computes and we leave this as part of future work.

The applications we used for this study are:

- HYCOM[6] is a popular ocean modeling code. We ran the standard dataset on 8 cpus and large dataset on 256 cpus.

- Milc [7] is a code for simulations of SU3 lattice gauge theory on MIMD parallel machines. In this paper we ran the ks_imp_dyn algorithm. We ran a small problem on 8 cpus and a larger problem on 256 cpus.

First we show our models accuracy for predicting runtime for jaguar without accelerators (i.e. the host CPUs); these are shown in Table III. As shows in this table the models have average errors of 6.3% and are no worse than 18.1%. The higher prediction error for HYCOM can be attributed to the inaccuracy of the framework to model relative larger number of branches in the applications inner loops.

TABLE III. MEASURED AND PREDICTED RUN TIMES ON JAGUAR

Application	Measured	Predicted	% Error[1]
Milc (8cpu)	278	277	0.4%
Milc (256cpu)	1,345	1,350	0.4%
HYCOM (8cpu)	262	246	6.1%
HYCOM (256cpu)	809	663	18.1%

[1] %Error=abs((Measured – Predicted)/Measured))*100

TABLE IV. IDIOM INSTANCES IN CODE

Idiom	HYCOM	Milc
Gather/scatter	1,797	156
stream	1,300	105

In order to find out how accelerators might affect the performance of these applications we used performance models of a hypothetical machine like Jaguar but with hardware accelerators (i.e. GPUs and FPGAs). We identified the number of instances of gather/scatter and stream idioms in these workloads and this is shown in Table IV.

In this study the amount of speedup that may be achieved by running idioms on accelerators is limited by the idioms dynamic coverage, i.e., fraction of application time spent in idiom loops. Table V shows the dynamic coverage of idioms for HYCOM, and Milc for both small and large data inputs. Table V also shows that the coverage of runtime by a particular idiom changes with data input. For example, gather/scatter idiom accounts for 14.2% of runtime of HYCOM 8 cpu case, and 4.6% of HYCOM 256 cpu case.

TABLE V. IDIOM RUNTIME CONTRIBUTION

	HYCOM (8cpu)	HYCOM (256cpu)	Milc (8cpu)	Milc (256cpu)
Gather/scatter	14.2%	4.6%	1.2%	0.7%
stream	21.1%	16.9%	5.6%	3.0%

Note that the remainder of the section discusses runtime of only the idiom code and at the end of this section presents results for the entire applications compute time. Using the performance models for Milc 256 cpus and HYCOM 256cpus we explored the per basic-block idiom behavior of these two applications. First, we calculate speedups by individual idioms when entirely run on one of piece of hardware (i.e. GPU, CPU, and FPGA). This data is presented in Table VI and VII for HYCOM and Milc, respectively. The tables summarize the cumulative time across all MPI tasks for gather/scatter and stream idioms. Comparing total idiom runtimes for executing both idioms on CPU with that of executing it entirely on GPU or FPGA has some interesting results. Blindly porting all gather/scatter idioms to the FPGA can improve performance of scatter/gather idioms over the CPU in the range of 7X-15X for the FPGA and 6X-12X for the GPU, while porting stream idioms to the FPGA can improve performance of stream idioms in the range of 12X-14X. Porting all streams idioms to the GPU improves performance of stream idioms by ~10X for Milc. However, blindly porting all instances of the stream idiom in HYCOM to GPU results in a 50% performance loss for the stream idioms. This can be attributed to the data footprint of the streams idioms in HYCOM being too small to saturate all the threads available in the GPU which is required to achieve high memory bandwidth. As we showed in figure 4, significantly larger data sizes are required for the stream idioms to speed up on the GPU.

TABLE VI. RUN TIMES FOR IDIOMS IN HYCOM

	CPU	FPGA	GPU
Gather/Scatter	7,768	495	638
Stream	28,459	2,302	44,166
Total	**36,556**	**2,798**	**44,803**

TABLE VII. RUN TIMES FOR IDIOMS IN MILC

	CPU	FPGA	GPU
Gather/Scatter	2,376	334	399
Stream	10,452	771	1,087
Total	**12,827**	**1,104**	**1,487**

Next sifting through the data reveals that the optimal choice of accelerator differs by basic block of the idioms being ported. Some blocks run faster on the GPU while some run faster on the FPGA, and there are still a small minority that run faster on the CPU. We perform a series of predictions comparing the performance impact of running the set of basic blocks on their optimal choice of hardware: GPU, CPU, or FPGA. Tables VIII and IX show the results of these investigations. The three columns in the table represent the performance effect of picking the optimal hardware for each basic-block when provided the choice of host CPU vs. FPGA, host CPU vs. GPU, and host CPU, GPU, or FPGA. One can compare the results of this more selective porting vs. the type of blind porting results shown in tables VI and VII. There are some interesting results when one compares the run time of entirely running idioms on one device, shown in table VI and VII, versus running idioms on the optimal choice of device by basic block shown in tables VIII and IX. Shown in table VIII is the result of this prediction for HYCOM. In the first column is the best among CPU versus FPGA. In this case optimal matching reduces run time of idiom code by only 5 seconds as compared to running the idioms entirely on the FPGA, shown in table VI. Next, in the second column is the optimal matching among GPU and

CPU. In this case optimal matching makes significant improvements for runtime of idiom code of ~5.5X over the just the CPU alone and ~6.5X over the just the GPU alone. Finally the results of choosing the optimal hardware among all three: GPU, CPU, and FPGA, is shown in the last column. Again there are noticeable improvements for runtime of idiom code of ~14X over CPU, ~17X over GPU and ~7.7% improvement over the FPGA.

TABLE VIII. OPTIMAL MAPPING OF DEVICE FOR HYCOM

	CPU vs. FPGA	CPU vs. GPU	Optimal of CPU, GPU, FPGA
Gather/Scatter	495	638	448
Stream	2,297	6,096	2,149
Total	2,792	6,734	2,596

We perform a similar analysis for Milc and the results are shown in Table IX. Comparing optimal choice of CPU and FPGA we find that the FPGA is the best choice. Similarly, comparing optimal choice of the GPU and the CPU, the GPU is the best choice. Finally the last column in table IX shows that choosing between the three GPU, CPU, FPGA the performance improvement over just choosing the FPGA is <0.1%. Hence for this application gather/scatter and stream idioms can all run on the FPGA for nearly optimal performance.

TABLE IX. OPTIMAL MAPPING OF DEVICE FOR MILC

	CPU vs. FPGA	CPU vs. GPU	Optimal of CPU, GPU, FPGA
Gather/Scatter	334	399	334
Stream	770	1,087	765
Total	1,104	1,486	1,099

Next using our models for stream and gather/scatter we predicted the overall run times of Milc and HYCOM if run on the optimal hardware (i.e. CPU, FPGA, or GPU). Our predictions show that the overall improvement to these applications compute time by porting these idioms to the accelerators is 3.4% for Milc and 20% for HYCOM. While the improvements for Milc are not as significant, the improvements for each individual idiom are quite significant as shown in table IX.

VIII. BACKGROUND

Several techniques exist to model general-purpose architectures [20-39] spanning flavors of analytical and trace-based methods. Analytical models require a detail understanding of the complex interactions of software and hardware and are generally difficult to generalize and automate.

Hardware accelerators have been the focus of several studies. Alam et al [16] investigate using their Modeling Assertions to model the multi-streaming, vector processing capabilities of the X1E on the NAS SP kernel [17] Hong and Kim [18] developed an analytical model for GPU performance and applied it to micro-kernel and benchmarks, but not full scale HPC applications. Govindaraju et al [19] developed a memory model for GPUs for a set of algorithms used in scientific applications. They tested the model on benchmark kernels but not full-scale HPC applications.

In the study by Carrington el al.[14] the performance of gather/scatter idioms was modeled on the FPGAs of the Convey HC-1 system. In this work we characterized both FPGA and GPU with additional idioms and developed models for both the FPGA and the GPU for two idioms. We also presented results of predictions for two full-scale HPC applications, Milc and HYCOM.

IX. CONCLUSIONS AND FUTURE WORK

In this paper we presented characterizations of several idioms or compute patterns on FGPA and GPU. Depending on the idiom and data input size a particular device or the CPU may be the optimal choice.

We also presented an extension of our methodology to model gather/scatter and stream idioms on FPGAs and GPUs. We use our models to project the speedup of gather/scatter and stream idioms on a system similar to Jaguar, a Cray XT5, but with hardware accelerators. Our models show that while its beneficial to run these idioms on accelerators the optimal choice is dependent on data size. We also presented a scenario of optimal choice of CPU, FPGA, and GPU for each idiom-data size pair.

Our models for the projection study ignored the cost of data transfers. However, in this paper we presented some preliminary analysis of data migrations that outlined the importance of data migration cost. Data migration cost was also shown to be an important factor to achievable performance by GPU [40]. In our future work we will extend the models to include the cost of data migration. In this paper, the models assume that the CPU is blocked while the operation is performed on the accelerator. We would like to model the potentials of overlapping CPU compute with accelerator compute. It would also be interesting to investigate power savings opportunities when either CPU or accelerators are blocked. Finally, we will also extend our models for other idioms and devices.

ACKNOWLEDGMENT

This work was supported by the DoD and used elements at the Extreme Scale Systems Center, located at ORNL and funded by the DoD. This research used resources of the National Center for Computational Sciences at Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. We would like to thank Ross Walker for the use of FERMI GPU for experiments. We would also like to thank Convey computers for providing the graph500 personality used in our validation experiments.

REFERENCES

[1] B. Mohr and F. Wolf, "KOJAK - A Tool Set for Automatic Performance Analysis of Parallel Applications," presented at the European Converence on Parallel Computing (EuroPar), 2003.

[2] C. Olschanowsky, M. Meswani, et al, "PIR: A Static Idiom Recognizer," in First International Workshop on Parallel

Software Tools and Tool Infrastructures (PSTI 2010), San Diego, CA, 2010.

[3] Convey Computer - http://www.conveycomputer.com/products.html

[4] http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf

[5] J.P Walters, B. Qudah, and V. Chaudhary, "Accelerating the HMMER Sequence Analysis Suite Using Conventional Processors", in Proceedings of the 20th Int'l Conf. on Advanced Information Networking and Applications (AINA'06), 2006.

[6] http://www.HYCOM.org/HYCOM/overview

[7] http://physics.utah.edu/~detar/milc.html

[8] http://www.graph500.org/index.html

[9] Michael Laurenzano, et al., "PEBIL Efficient Static Binary Instrumentation for Linux," in Proceedings of the International Symposium on Performance Analysis of Systems and Software, White Plains, NY, March 2010.

[10] Mustafa M. Tikir, Michael Laurenzano, Laura Carrington, and Allan Snavely, "PSINS: An Open Source Event Tracer and Execution Simulator for MPI Applications," in Proceedings of Euro-Par, 2009.

[11] A. Snavely, et al., "A Framework for Application Performance Modeling and Prediction," ACM/IEEE Conference on High Performance Networking and Computing, 2002.

[12] L. Carrington, et al., "How well can simple metrics represent the performance of HPC applications?," Proceedings of the ACM/IEEE SC2005 Conference on High Performance Networking and Computing, 2005.

[13] M. Tikir, et al., "Genetic Algorithm Approach to Modeling the Performance of Memory-bound Codes," The Proceeding of the ACM/IEEE Conference on High Performance Networking and Computing, 2007.

[14] L. Carrington, M. Tikir, et al., "An Idiom-finding Tool for Increasing Productivity of Accelerators," in Proceedings of Int'l Conf of Supercomputing (ICS), 2011.

[15] www.Top500.org

[16] Alam, S., Bhatia, N. and Vetter, J. An Exploration of Performance Attributes for Symbolic Modeling of Emerging Processing Devices HPCC, 2007, 683-694

[17] NAS Parallel Benchmarks (NPB) see, http://www.nas.nasa.gov/Resources/Software/npb.html.

[18] Hong, S. and Kim, H. An Analytical Model for a GPU Architecture with Memory-level and Thread-level Parallelism Awareness ISCA'09, Austin, Texas, USA, 2009.

[19] Govindaraju, N., Larson, J., Gray, J. and Manocha, D. A Memory Model for Scientific Algorithms on Graphics Processors Supercomputing, Tampa, Florida USA, 2006.

[20] V. Adve and R. Sakellariou, "Application representations for multiparadigm performance modeling of large-scale parallel scientific codes," The International Journal of High Performance Computing Applications, vol. 14, 2000.

[21] S. Alam and J. Vetter, "A Framework to Develop Symbolic Performance Models of Parallel Applications," presented at the 5th International Workshop on Performance Modeling, Evaluation, and Optimization of Parallel and Distributed Systems, 2006.

[22] G. Almasi, et al., "Demonstrating the scalability of a molecular dynamics application on a Petaflop computer," presented at the Proceedings of the 15th international conference on Supercomputing, Sorrento, Italy, 2001.

[23] B. Armstrong and R. Eigenmann, "Performance forecasting: Towards a methodology for characterizing large computationals applications," in Internationals Conference on Parallel Processing, 1998.

[24] D. Bailey and A. Snavely, "Performance Modeling: Understanding the Present and Predicting the Future," EuroPar, 2005.

[25] J. Bourgeois and F. Spies, "Performance prediction of an NAS benchmark program with chronosmix enviroment," presented at the 6th International Euro-Par Conference, 2000.

[26] M. Clement and M. Quinn, "Automated performance prediction for scalable parallel computing," Parallel Computing, vol. 23, 1997.

[27] M. J. Clement and M. J. Quinn, "Analytical performance prediction on multicomputers," Supercomputing, pp. 886-894, 1993.

[28] D. Culler, et al., "LogP: Towards a realistic modle of parallel computation," in 4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 1993.

[29] M. Faerman, et al., "Adaptive performance prediction for distributed data-intensive applications," presented at the Supercomputing, 1999.

[30] T. Fahringer and M. Zima, "A static parameter based performance prediction tool for parallel programs," presented at the The Int'l Conf. on Supercomputing, 1993.

[31] D. J. Kerbyson, et al., "Predictive Performance and Scalability Modeling of Large-Scale Application," Supercomputing, 2001.

[32] C. Lim, et al., "Implementation lessons of performance prediction tool for parallel conservative simulation," presented at the 6th International Euro-Par Conference, 2000.

[33] G. Marin and J. Mellor-Crummey, "Cross Architecture Performance Predictions for Scientific Applications Using Parameterized Models," In Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems, June 2004.

[34] B. Mohr and F. Wolf, "KOJAK - A Tool Set for Automatic Performance Analysis of Parallel Applications," presented at the European Converence on Parallel Computing (EuroPar), 2003.

[35] J. Simon and J.-M. Wierum, "Accurate Performance Prediction for Massively Parallel Systems and its Applications," Euro-Par'96 Parallel Processing, vol. 1124, pp. 675-688, 1996.

[36] A. van Gemund, "Symbolic performance modeling of parallel systems," IEEE Transactions on Parallel and Distributed Systems, vol. 14, 2003.

[37] A. Wagner, et al., "Performance models for the processor farm paradigm," IEEE Transactions on Parallel and Distributed Systems, vol. 8, 1997.

[38] L. Yang, et al., "Cross-Platform Performance Prediction of Parallel Applications Using Partial Execution," presented at the Proceedings of the 2005 ACM/IEEE conference on Supercomputing, 2005.

[39] X. Zhang and Z. Xu, "Multiprocessor Scalability Predictions Through Detailed Program Execution Analysis," International Conference on Supercomputing, pp. 97-106, 1995.

[40] Chris Gregg and Kim Hazelwood. "Where is the Data? Why You Cannot Debate GPU vs. CPU Performance Without the Answer," International Symposium on Performance Analysis of Systems and Software (ISPASS). Austin, TX. April 2011.

[41] J He, , et al.,, "Automatic Recognition of Performance Idioms in Scientific Applications," in the Proceedings of the 25th IEEE International Parallel and Distributed Processing Symposium (IPDPS'11), Anchorage, Alaska, May 16-20, 2011.