Performance Modeling for Multilevel Communication in SHMEM+

V. Aggarwal, C. Yoon, A. George, H. Lam, G. Stitt

NSF Center for High-Performance Reconfigurable Computing (CHREC)
ECE Department, University of Florida, Gainesville, FL 32611-6200
{aggarwal, yoon, george, hlam, gstitt}@chrec.org

Abstract

The field of high-performance computing (HPC) is currently undergoing a major transformation brought upon by a variety of new processor device technologies. Accelerator devices (e.g. FPGA, GPU) are becoming increasingly popular as coprocessors in HPC, embedded, and other systems, improving application performance while in some cases also reducing energy consumption. The presence of such devices introduces additional levels of communication and memory hierarchy in the system, which warrants an expansion of conventional parallel-programming practices to address these differences. Programming models and libraries for heterogeneous, parallel, and reconfigurable computing such as SHMEM+ have been developed to support communication and coordination involving a diverse mix of processor devices. However, to evaluate the impact of communication on application performance and obtain optimal performance, a concrete understanding of the underlying communication infrastructure is often imperative. In this paper, we introduce a new multilevel communication model for representing various data transfers encountered in these systems and for predicting performance. Three use cases are presented and evaluated. First, the model enables application developers to perform early design-space exploration of communication patterns in their applications before undertaking the laborious and expensive process of implementation, yielding improved performance and productivity. Second, the model enables system developers to quickly optimize performance of data-transfer routines within tools such as SHMEM+ when being ported to a new platform. Third, the model augments tools such as SHMEM+ to automatically improve performance of data transfers by self-tuning internal parameters to match platform capabilities. Results from experiments with these use cases suggest marked improvement in performance, productivity, and portability.

Categories and Subject Descriptors C.4 [*Performance of Systems*]: Modeling techniques

General Terms Design, Performance

Keywords Performance prediction, modeling, multilevel communication, PGAS, SHMEM, reconfigurable computing.

1. Introduction

Advancements in new technologies such as reconfigurable-logic devices and heterogeneous multi-core and many-core devices are revolutionizing high-performance computing (HPC) [3, 15]. These devices are now commonly employed as accelerators in HPC systems to boost system performance and energy efficiency [18]. Unfortunately, the addition of these devices in the systems also increases the system complexity, making application development more challenging and demanding, with multiple levels of parallelism and communication to be understood and exploited.

To ease design complexity, designers targeting reconfigurable HPC systems which feature a mix of processing devices including microprocessors and FPGAs, can leverage system-level concepts from conventional HPC for developing scalable, parallel applications. However, there are characteristic differences between reconfigurable computing (RC) systems and traditional HPC systems, such as additional levels of memory and communication in the system, which necessitate the expansion of current parallel-programming practices to address these differences. SHMEM+ [1], an extended SHMEM library based on the multilevel PGAS model, was developed to enable communication and synchronization between FPGAs and CPUs in reconfigurable HPC systems. SHMEM+ provides application developers with a productive and portable mechanism for creating scalable, parallel applications that execute over a mix of microprocessors and FPGAs.

One important challenge when using libraries such as SHMEM+ is choosing an appropriate communication strategy. Variances include, for example, who initiates the transfer, what functions are employed, what intermediate steps are involved, etc. Such factors can have a significant impact on the overall performance of an application. Therefore, it is critical for an application designer to understand the underlying communication infrastructure.

Traditional HPC systems and applications assist designers in understanding and optimizing communication infrastructure by employing communication models that use a set of parameters to provide a high-level representation of communication, while hiding the unnecessary details. Although a few general-purpose communication models have been prevalent for traditional HPC systems, these models were not developed to target multilevel systems such as reconfigurable HPC systems. As a result, these models lack sufficient details for representing multilevel systems completely and accurately.

In this paper, we propose a multilevel communication model that accounts for the existence of multiple levels of communication encountered by various data-transfer functions in a library such as SHMEM+. The model provides a system-level representation which integrates the effect of intermediate steps of communication present in a multilevel transfer. In addition, the model allows for the capability of overlapping these intermediate steps. The time for intermediate steps of a transfer itself can be estimated using any of the existing parallel models [2, 5, 7, 17] or determined using microbenchmarking. The multilevel communication model does not impose the use of a specific model for estimating the time of intermediate steps of transfer. The various merits of the proposed performance model for multilevel communication are showcased in this paper through several examples. (a) The model enables a developer to determine communication bottlenecks in applications through early design-space exploration (DSE) which, as demonstrated through a case study described in Section 4, led to a performance improvement of approximately 42%. Besides improvement in performance, the modeling allowed us to eliminate multiple iterations of the expensive development cycle, improving developer productivity. (b) The model also enables system developers to quickly optimize the internal data-transfer functions in the SHMEM+ library for a target RC platform. (c) Furthermore, we show that this model enables self-tuning of the SHMEM+ library that automatically improves the performance of certain data transfers by varying its internal parameters to match the capabilities of the system. This capability further improves the portability of SHMEM+. The effectiveness of self-tuning is demonstrated through experimental results on two different platforms, where an improvement in bandwidth of up to 100% was obtained in certain cases by self-tuned routines in SHMEM+.

The remainder of this paper is organized as follows. Section 2 provides background and related research. Section 3 provides a detailed overview of the multilevel communication model. In Section 4, we showcase the effectiveness of the model in enabling early DSE for an application. Section 5 employs the model to optimize the performance of a SHMEM+ library for a target RC system. In Section 6, we apply the model to augment SHMEM+ with the self-tuning capability. Finally, Section 7 summarizes the work with conclusions and directions for future work.

2. Related Research

Traditionally, developers of parallel programs have performed coordination between tasks using either message-passing libraries such as MPI [12] or shared-memory libraries such as OpenMP [14]. Recently, languages and libraries that present a partitioned global address space (PGAS) to the programmer, such as UPC [6] and SHMEM [16], have become more visible and popular. Most of these languages and libraries were developed for traditional HPC systems based on a homogeneous cluster of microprocessors. A SHMEM+ library [1] based on the multilevel PGAS model was developed to enable communication and synchronization between FPGAs and CPUs in a reconfigurable HPC system. The communication model proposed in this work extends the existing research on the SHMEM+ library to assist application developers in obtaining better performance while reducing their effort. We also leverage the existing research in communication and performance modeling for traditional HPC systems. There are a variety of communication models prevalent in the field of HPC that aim to provide developers with a better understanding of the underlying communication infrastructure. Such models, as described in this section, provide valuable ideas and useful insight towards our proposed model.

2.1 SHMEM+ and Multilevel PGAS

Multilevel PGAS introduced in [1] extends the concept of partitioned, global address space to a multilevel abstraction by integrating a hierarchy of multiple memory components present in reconfigurable HPC systems into a single, virtual memory layer as shown in Figure 1. As a result, application developers are oblivious to the underlying details while transferring data between any two devices in the system. The SHMEM+ library, which is based on

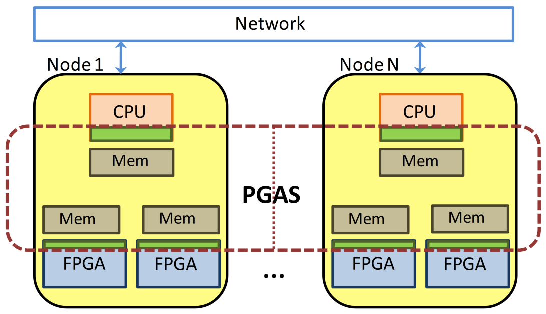


Figure 1. Virtual memory layer in multilevel PGAS integrates a hierarchy of memory components present in system.

the multilevel PGAS model, allows the developers to use the same communication interface irrespective of the source and destination of the transfer. In addition to reducing the programming complexity, SHMEM+ also provides numerous choices for implementing the required communication in the application. Since the choice of functions and the communication strategy used to establish communication can have a significant effect on performance, the communication model presented in this paper becomes an important tool for understanding the behavior of these transfer functions and obtaining high performance for applications.

2.2 Performance Modeling

Communication models have been prevalent in the field of HPC for estimating the performance of communication in parallel applications. Models like Parallel Random Access Machine (PRAM) [7], Bulk Synchronous Parallel (BSP) [17], LogP [5], LogGP [2], PlogP [10], etc. aim to be fairly generic and architecture-independent, and provide high-level estimates of communication performance for parallel programs. However, these models were developed for traditional HPC systems based on a homogeneous set of microprocessors and cannot be directly employed to represent the multilevel transfers in reconfigurable HPC systems.

Other researchers have attempted to build system-level models for heterogeneous systems. Heterogeneous LogGP (HLogGP) [4] considers extensions of LogGP for multiple processor speeds and communication networks within a cluster. In [11], system-level modeling concepts form the basis for a proposed model for heterogeneous clusters. However, the primary emphasis of these models was to target systems based on heterogeneous microprocessors and they were not intended for multilevel systems based on accelerators.

In [9], RC Amenability Test (RAT) defines an analytical model for performance estimation of an RC algorithm on a specified RC platform prior to any implementation. Although it provides a fairly accurate representation of an algorithm targeting a single device, it was not intended to model the effects of scalable, multi-device applications. By contrast, our work focuses on modeling and estimation of the communication performance in a large-scale RC system while leveraging other models (such as RAT) for estimating the performance of computational parts of the application.

3. Multilevel Communication Model

The SHMEM+ library provides a high-productivity environment for establishing communication in RC systems by abstracting the details of the underlying data transfers with a uniform, high-level interface. Figure 2 illustrates the different options for data transfers provided by the SHMEM+ library in a system with a CPU and an FPGA on each node (although the library can support any number

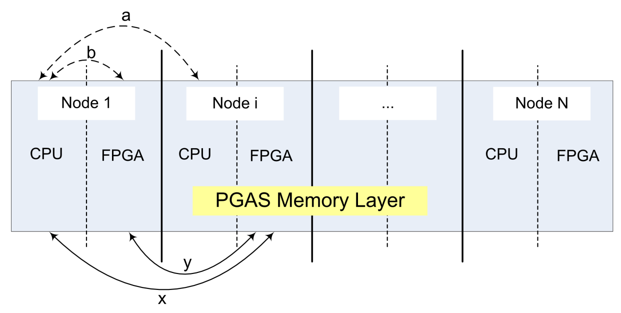


Figure 2. Transfer capabilities supported by various communication routines in SHMEM+.

and variety of devices). Some of the direct-transfer capabilities shown in the figure such as the ones labeled ‘x’ and ‘y’ were enabled by SHMEM+ and did not exist previously.

While having various options to transfer data from one device to another simplifies the development of parallel applications, users should understand the underlying mechanism and tradeoffs associated with such transfers in order to obtain optimal performance. Since only the CPU on each node can physically perform inter-node communication, some of the data transfers are internally achieved through multiple intermediate transfer steps. Consider a function that provides direct transfer between two remote FPGAs (labeled as ‘y’ in Figure 2). This function is internally performed by (a) transferring data to internal buffers on local CPU, (b) followed by a transfer to the CPU on the remote node, and (c) finally writing data to the memory of the destination FPGA. However, such a function may serialize the aforementioned transfer steps which could have been explicitly parallelized by an application developer. For example, in an application where data is collected from several FPGA devices on a remote CPU, an application developer can explicitly parallelize the transfer of data from FPGAs to their local CPUs on all nodes and then transfer the data from the CPUs on each node to the remote CPU where data is collected (more details about this example are presented in Section 4). To enable better understanding of the underlying details of communication present in SHMEM+, we derive a multilevel communication model for the same.

The communication time for a point-to-point data transfer between any two devices of the system shown in Figure 2 can be represented by a generic model described as:

$$T_{comm} = f(T_{cpu\leftrightarrow cpu}) + f(T_{cpu\leftrightarrow fpga}) \quad (1)$$

Where,

T_{comm}	:	Total time for communication,
$T_{cpu\leftrightarrow cpu}$	:	Transfer time for two CPUs on remote nodes,
$T_{cpu\leftrightarrow fpga}$	:	Total time for various transfers between a CPU and its local FPGA,
$f(T)$	=	*serial time T for communication − part of time T hidden by overlap with other communication*
	=	*non-overlapped part of time T*

The goal of the model in Equation 1 is to estimate the performance of a data-transfer routine by incorporating each intermediate step of the transfer separately. Depending upon the source and destination devices, a communication routine may require two transfers between a CPU and an FPGA (such as the case for transfer labeled as ’y’ in Figure 2). $T_{cpu\leftrightarrow fpga}$ in Equation 1 represents the cumulative time of both transfers in such cases. The multilevel model does not estimate the performance of individual transfer steps or impose the use of a specific model for the same. Instead, it allows the use of different communication models for representing each component of the complete transfer. Such modeling is important because it is often difficult to describe the various intermediate steps of transfer using a single communication model. In our experiments we perform microbenchmarking to determine the time for intermediate steps of communication accurately. Many RC systems exhibit asymmetric performance for read (FPGA to CPU) and write (CPU to FPGA) operations. To account for this asymmetry in speed of read and write operations for an FPGA, the model can be rewritten as:

$$T_{comm} = f(T_{cpu\leftrightarrow cpu}) + f(T_{cpu\leftarrow fpga}) + f(T_{cpu\rightarrow fpga}) \quad (2)$$

Where,

$T_{cpu\leftarrow fpga}$	:	Transfer time for read operation from FPGA,
$T_{cpu\rightarrow fpga}$	:	Transfer time for write operation to FPGA

By overlapping various steps of communication, it is possible to reduce the overall time taken by the transfer. It should be noted that the model in Equation 2 (and Equation 1) only considers the effective (non-overlapped) time of each level of communication that influences the overall performance of communication. The provision for “effective” time is an important feature of the model because communication with the FPGA is invariably an expensive operation and is often overlapped with other steps of communication to improve overall performance. In Section 5, we use this feature of the model to optimize transfers between two FPGAs on remote nodes. In addition to estimating communication time, the multilevel communication model can also be combined with existing approaches such as RAT [9] to estimate the execution time of the complete application.

In the following three sections, we discuss several benefits of the multilevel communication model through three use cases. First, the model enables application developers to perform early design-space exploration of communication patterns in their applications before undertaking the laborious and expensive process of implementation, yielding improved performance and productivity. Second, the model enables system developers to quickly optimize performance of data-transfer routines within tools such as SHMEM+ when being ported to a new platform. Third, the model augments tools such as SHMEM+ to automatically improve performance of data transfers by self-tuning its internal parameters to match platform capabilities which improves the portability of SHMEM+.

The results showcased in this work were gathered from experiments on two different systems. Our first system (Mu cluster) consists of four Linux servers connected via QsNetII from Quadrics (offering a raw link bandwidth of 10Gb/s). Each server is comprised of an AMD 2GHz Opteron 246 processor and a tightly coupled set of four FPGA accelerators on a PROCStar-III PCIe board from GiDEL. The FPGA board features four Altera Stratix-III EP3SE260 FPGAs, each with two external DDR2 memory banks of 2GB and one bank of 256MB. The second system is the Novo-G RC supercomputer, which is comprised of 24 computer servers (of which only 16 were fully functional at the time of this research), each equipped with a Nehalem quad-core Xeon processor and a PROCStar-III board. The servers are connected via DDR InfiniBand interconnect technology (offering a raw link bandwidth of 20Gb/s).

4. Early DSE in Applications

One advantage of the multilevel communication model is the enabling of early design-space exploration (DSE) that can improve performance and reduce development time for an application. By providing performance estimates, the model allows developers to

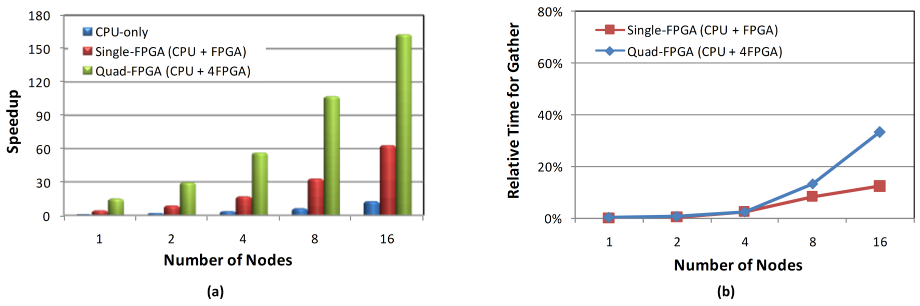


Figure 3. (a) Scaling behavior of CBIR application observed from experiments conducted on Novo-G for a search database consisting of 22,000 images, each of size 128 × 128 pixels of 8 bits, (b) Time to gather results (4 bytes per image, i.e. 88KB of data) on root node as a percentage of application execution time.

make design decisions about the communication infrastructure before undertaking expensive implementation. We illustrate this capability using an example of a content-based image retrieval (CBIR) application.

CBIR is a common application encountered in computer vision and consists of searching a large database of digital images for the ones that are visually similar to a given query image, where the search is based on contents of the image. The content in this context can be one of the several features present in the image, such as colors, shapes, textures, or any other information that can be derived from the image. CBIR has been widely adopted in many domains such as biomedicine, military, commerce, education, and Web image classification and searching. Each image in a CBIR [13] system is represented by a feature vector, which is based on the contents of the image. Similarity between a query image and the set of images in the database is determined by measuring similarity between their feature vectors. The processes of determining the feature vector and analyzing images for similarities are often the most computationally intensive stages in any CBIR system [8].

The parallel algorithm employed in our experiments distributes the set of images to be searched over a set of nodes and allows multiple processing devices to evaluate these images simultaneously (more details about implementation of the algorithm can be found in [1]). The steps involved in our parallel implementation based on SHMEM+ are as follows:

1. All nodes perform initialization using *shmem_init*, which also configures local FPGAs on each node with the desired bitfile.
2. CPUs on all nodes read their subset of input images from a storage device (such as a local hard disk or a network storage device) along with the feature vector of the query image.
3. CPUs transfer the subset of images to their local FPGAs for hardware acceleration using the *shmem_putmem* function.
4. CPUs initiate the execution on their local FPGAs through a "GO" signal. FPGAs on all nodes compute feature vectors and similarity measures for their subset of images in parallel.
5. FPGAs signal the completion of execution to local CPUs through a "DONE" signal. Once computation on CPUs and FPGAs on each node is complete, all nodes synchronize using *shmem_barrier_all*.
6. Finally, similarity values from all of the FPGAs are gathered on the root node using one of the several approaches (determined by our analysis presented in Section 4.1). Results are then sorted in decreasing order of similarity.

By employing multiple FPGAs to accelerate the application, the computation time of the algorithm can be reduced significantly as shown by the performance of a typical implementation in Figure 3(a). The single-FPGA designs refer to the designs which employ multiple nodes with each node using a single FPGA. Similarly, quad-FPGA designs execute over multiple nodes with each node using four local FPGAs. As the application is scaled across more nodes, communication starts becoming a significant proportion of the total time. Figure 3(b) shows that the amount of time required to collect the results on the root node increases substantially with increasing number of processing nodes. The effect is more pronounced for the quad-FPGA design as the results need to be collected from more FPGAs (64 FPGAs for a system size of 16 nodes). As a result, the relative performance gain from employing more nodes (and FPGAs) starts decreasing.

To resolve this bottleneck, we need to understand the mechanism by which the application collects results on the root node. The gather operation in the application can be implemented in multiple ways such as:

***Approach 1*:** By using a *shmem_getmem* function on the root node to receive the output data from all of the FPGAs involved in the application individually.

***Approach 2*:** By using a *shmem_putmem* function on every node to send the results computed by each FPGA to the root node individually.

***Approach 3*:** By first collecting the results from the local FPGAs on every node's CPU and then sending them to the root node collectively using *shmem_putmem* on each node.

Although none of the aforementioned approaches may present an optimal solution, they were chosen for our case study to exemplify the capability of our multilevel communication model in estimating the performance of the gather operation (implemented using different approaches). More efficient but complicated approaches such as tree reductions and broadcasts to be incorporated in SHMEM+ are the subject of future research.

4.1 Performance Estimation

While performing the same gather operation, the aforementioned approaches can offer different performance and require different levels of developer effort. To evaluate the impact of these three approaches on the overall performance of the CBIR application, we estimate the performance of gather operation using each approach.

The difference in the performance becomes apparent by applying our multilevel communication model to the three approaches. The estimated time for performing the gather operation can be ex-

Table 1. Observed time and estimated time to perform gather operation by single-FPGA designs using three different approaches on Novo-G. All times are reported in milliseconds. Total amount of data collected on the root node is 8MB in all cases.

n	$T_{cpu\leftarrow fpga}$	$T_{cpu\leftrightarrow cpu}$	Approach 1		Approach 2		Approach 3	
			Est.	Obs.	Est.	Obs.	Est.	Obs.
1	8.14	5.72	8.14	8.33	8.14	8.27	8.14	8.22
2	4.93	2.88	12.74	12.41	7.81	7.98	7.81	7.98
4	2.93	1.45	16.07	16.79	7.28	6.90	7.28	6.84
8	1.91	0.73	20.39	22.28	7.02	6.45	7.02	6.39
16	1.51	0.37	29.71	35.12	7.06	6.17	7.06	6.16

Table 2. Observed time and estimated time to perform gather operation for quad-FPGA designs on Novo-G using (a) Approaches 1 and 2, (b) Approach 3. All times are reported in milliseconds. Total amount of data collected on the root node is 8MB in all cases.

n	$T_{cpu\leftarrow fpga}$	$T_{cpu\leftrightarrow cpu}$	Approach 1		Approach 2	
			Est.	Obs.	Est.	Obs.
1	2.93	1.45	11.72	12.00	11.72	12.00
2	1.91	0.73	18.20	20.85	10.56	11.99
4	1.51	0.37	28.60	31.72	10.48	8.54
8	1.24	0.20	45.28	52.50	10.56	8.29
16	1.24	0.10	85.36	81.37	10.96	8.21

(a)

n	$T_{cpu\leftarrow fpga}$	$T_{cpu\leftrightarrow cpu}$	Approach 3	
			Est.	Obs.
1	2.93	5.72	11.72	13.93
2	1.91	2.88	10.52	10.99
4	1.51	1.45	10.39	10.64
8	1.24	0.73	10.07	8.57
16	1.24	0.37	10.51	8.95

(b)

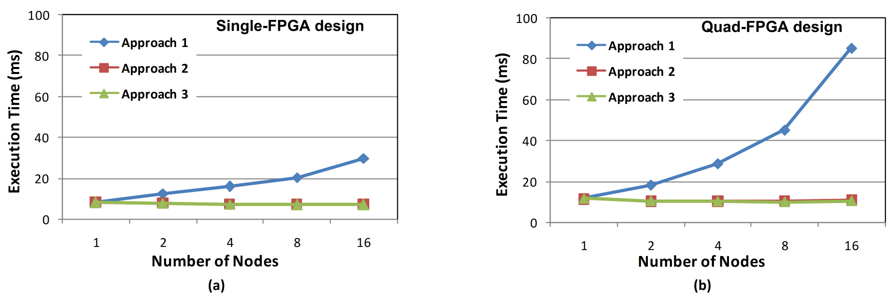


Figure 4. Estimated performance of gather operation on Novo-G using three different approaches for (a) Single-FPGA designs, (b) Quad-FPGA designs. Total amount of data collected on the root node is 8MB in all cases.

pressed using the model as:

$$T_{gather} = \sum_{i=1}^{n} T_{comm_i} \tag{3}$$

$$\begin{aligned} T_{comm_i} &= f(T_{cpu\leftarrow fpga}) + f(T_{cpu\leftrightarrow cpu}) + f(T_{cpu\rightarrow fpga}) \\ &= \textit{non-overlapped part of } T_{cpu\leftarrow fpga} \\ &\quad + \textit{non-overlapped part of } T_{cpu\leftrightarrow cpu} + 0 \end{aligned} \tag{4}$$

Where,

n	:	Number of nodes involved in the gather operation,
T_{gather}	:	Total time for gather operation,
T_{comm_i}	:	Communication time corresponding to i^{th} node

Since there are no write operations to an FPGA, the corresponding term becomes zero in Equation 4. While Approach 1 seems the most intuitive way to perform a gather, the process of invoking a get operation to receive data from all devices individually serializes all of the transfers. As a result, none of the communication time can be overlapped. Therefore, the time to perform the gather operation for Approach 1 can be described as follows:

Approach 1, Single-FPGA Design:

$$T_{gather} = (n-1) \times (T_{cpu\leftarrow fpga} + T_{cpu\leftrightarrow cpu}) + T_{cpu\leftarrow fpga} \tag{5}$$

Approach 1, Quad-FPGA Design:

$$T_{gather} = 4 \times [(n-1) \times (T_{cpu\leftarrow fpga} + T_{cpu\leftrightarrow cpu}) + T_{cpu\leftarrow fpga}] \tag{6}$$

Note that the root node only requires a read operation from its local FPGA, which does not involve a network transaction (last term in Equations 5 and 6).

By contrast, Approach 3 requires more effort from a developer as it first collects the data from the FPGAs on the local CPU before sending the data across to the root node. However, this approach allows each node to overlap the read operation from its local FPGAs (which is usually an expensive operation) with the same operation on the other nodes. Therefore, the time to perform the gather operation for Approach 3 can be represented as follows:

Approach 3, Single-FPGA Design:

$$T_{gather} = T_{cpu\leftarrow fpga} + (n-1) \times T_{cpu\leftrightarrow cpu} \tag{7}$$

Approach 3, Quad-FPGA Design:

$$T_{gather} = 4 \times T_{cpu\leftarrow fpga} + (n-1) \times T_{cpu\leftrightarrow cpu} \tag{8}$$

Approach 2, which appears very similar to Approach 1 on the surface, behaves much like Approach 3. Allowing each node to put the data directly from the local FPGAs to the root node essentially overlaps the part of the transfer which reads the data from the local FPGAs (into a temporary buffer in SHMEM+) with the same operation on the other nodes. As a result, Approach 2 yields performance that is comparable to the third approach while requiring comparatively lower developer effort. The performance of Approach 2 is identical to Approach 3 for single-FPGA designs. For quad-FPGA designs, Approach 2 requires four CPU-to-CPU transfers from each node to the root node (of a smaller data size) as opposed to a single transfer required by Approach 3. The estimated time can be represented using the equations as follows:

Approach 2, Single-FPGA Design:

$$T_{gather} = T_{cpu \leftarrow fpga} + (n-1) \times T_{cpu \leftrightarrow cpu} \tag{9}$$

Approach 2, Quad-FPGA Design:

$$T_{gather} = 4 \times [T_{cpu \leftarrow fpga} + (n-1) \times T_{cpu \leftrightarrow cpu}] \tag{10}$$

In order to compute the estimates for the time required by the gather operation, we performed microbenchmarking to determine $T_{cpu \leftarrow fpga}$ and $T_{cpu \leftrightarrow cpu}$ for different data sizes. Table 1 lists the input parameters (n, $T_{cpu \leftarrow fpga}$, and $T_{cpu \leftrightarrow cpu}$), the estimated gather times, and the corresponding times observed experimentally for performing the gather operation using single-FPGA designs on Novo-G. The results reported in the table correspond to a gather operation collecting 8MB data on the root node (to allow for longer gather times in our analysis, a data size much larger than required by our CBIR implementation was chosen). The estimates listed in the table for Approaches 1, 2 and 3 are computed using Equations 5, 9, and 7 respectively. For example when $n = 16$ (last row of Table 1), the estimate for Approach 3 can be computed from Equation 7 as:

$$\begin{aligned} T_{gather} &= T_{cpu \leftarrow fpga} + (n-1) \times T_{cpu \leftrightarrow cpu} \\ &= 1.51 + (16-1) \times 0.37 = 7.06 \text{ ms} \end{aligned}$$

The estimated and observed time for the quad-FPGA designs on Novo-G are listed in Tables 2(a) and (b) along with the corresponding input parameters required for computing the estimates. In Approach 3, CPUs collect data from their local FPGAs before sending the collected data to the root node. As a result, $T_{cpu \leftrightarrow cpu}$ for Approach 3 is different than for Approaches 1 and 2. The results are tabulated in separate tables (Tables 2(a) and (b)). The estimates listed in Tables 2(a) and (b) for Approach 1, 2 and 3 are computed using Equations 6, 10 and 8 respectively. For example when $n = 8$ (in Table 2 (a)), the estimate for Approach 2 can be computed from Equation 10 as:

$$\begin{aligned} T_{gather} &= 4 \times [T_{cpu \leftarrow fpga} + (n-1) \times T_{cpu \leftrightarrow cpu}] \\ &= 4 \times [1.24 + (8-1) \times 0.20] = 10.56 \text{ ms} \end{aligned}$$

Figure 4 shows the estimated performance of the gather operation using the different approaches for (a) single-FPGA designs and (b) quad-FPGA designs. While the time required for gathering the data increases significantly for the first approach, it stays relatively constant for the second and third approaches. The effect is more pronounced for the quad-FPGA designs because there are more processing devices involved in the gather operation.

4.2 Experimental Results

The estimates computed by use of the proposed model were verified by comparing them with experimental performance observed for the three approaches (reported in Tables 1 and 2). The average error between the estimates and experimental results was 9.4%

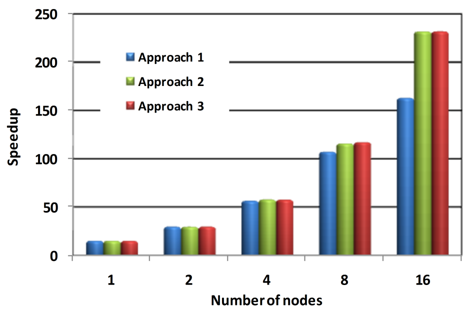


Figure 5. Speedup observed for quad-FPGA designs of CBIR over a sequential baseline executing on a single CPU on Novo-G. Experiments were conducted for a search database consisting of 22,000 images, each of size 128 × 128 pixels of 8 bits.

(which is considered reasonably accurate given the focus on early DSE prior to any implementation). A few cases experienced higher errors (between 20-25%), such as the quad-FPGA design over 16 nodes using Approach 2. With more processing devices, the time for intermediate steps of transfer became considerably small such that large variations were observed in experimental data and in the results of microbenchmarking (which form the inputs for the estimation model). As a result, even a minor deviation in measurements manifested as a larger relative error. Although not ideal, the absolute error had little impact on our design decisions as the trend observed between the experimental data and the estimates based on our model were consistent.

From the behavior of the three approaches, it appears that a CBIR application employing Approach 1 would lead to significant performance degradation as the system size increases, whereas with Approaches 2 and 3, the application would continue to offer satisfactory performance. The choice between using either of the two approaches (2 or 3) would be based on the level of programming complexity and the effort required from the developer. Approach 2 appears promising as it offers performance comparable to Approach 3 while requiring lower developer effort. To observe the effect of the three approaches on overall application performance, we developed a full CBIR implementation using the three approaches. Figure 5 shows the performance obtained for quad-FPGA designs of CBIR application on Novo-G using the three approaches. The results agree with the behavior predicted based on the estimates from our model. Approaches 2 and 3 offer similar performance, better than Approach 1. By using the proposed model to quickly perform early DSE of communication patterns, we were able to improve performance of the application by approximately 42%. This exercise showcased the use of our multilevel communication model as a tool for enabling DSE in the early phase of application development. Such a tool can eliminate several iterations of expensive design cycle and help in improving developer productivity.

5. Optimizing SHMEM+ Functions

The multilevel communication model allows system developers to quickly optimize the performance of a communication library such as SHMEM+ when porting it to a new system. Consider the case of data transfers between two remote FPGAs. As shown in Figure 6, such a transfer can be performed by (1) reading the data from the source FPGA to its local CPU; (2) transferring the data from the local CPU to the CPU on the remote node; (3) and finally, writing the data from the CPU on the remote node to the destination FPGA.

Instead of performing Steps 1, 2 and 3 sequentially, an efficient implementation for the transfer shown in Figure 6 may overlap

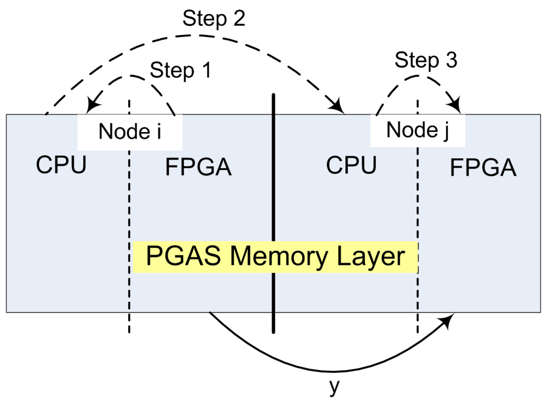


Figure 6. Intermediate steps involved in data transfers between two remote FPGAs using SHMEM+.

the intermediate steps of such a transfer. For example, Step 3 can be overlapped with Steps 1 and 2 collectively. In order to overlap various steps of a transfer, the data needs to be divided into smaller packets. The size of the data packet can have a significant impact on overall performance of the transfer. A small packet size would lead to low performance for intermediate steps of the transfer, while a large packet size would limit the amount of communication that can be efficiently overlapped. Determining an appropriate packet size can be a laborious task and may occasionally require development of a testbench and numerous executions of the testbench with various packet sizes until satisfactory performance is obtained.

5.1 Performance Estimation

The proposed model can assist system developers in determining an appropriate packet size without requiring any test code. To estimate the performance of the transfer for a particular packet size, our model can be employed as follows:
Let,

N : Number of packets
D : Size of data being transferred in bytes
P : Size of data packets in bytes
$T(L)$: Time T for transferring L bytes

then,

$$N = \lceil \frac{D}{P} \rceil \tag{11}$$

$$T_{step1} = \begin{cases} T_{cpu \leftarrow fpga}(P), & \text{when } D > P \\ T_{cpu \leftarrow fpga}(D), & \text{when } D < P \end{cases} \tag{12}$$

$$T_{step2} = \begin{cases} T_{cpu \leftrightarrow cpu}(P), & \text{when } D > P \\ T_{cpu \leftrightarrow cpu}(D), & \text{when } D < P \end{cases} \tag{13}$$

$$T_{step3} = \begin{cases} T_{cpu \rightarrow fpga}(P), & \text{when } D > P \\ T_{cpu \rightarrow fpga}(D), & \text{when } D < P \end{cases} \tag{14}$$

Here, T_{step1}, T_{step2} and T_{step3} represent the time to transfer a single packet for the corresponding steps in Figure 6. When $D < P$, the entire data is sent in a single packet. Whereas for $D > P$, the transfer is broken into packets of size P. By overlapping Step 3 with Steps 1 and 2 collectively, the overall time of transfer can be described as:

$$\begin{aligned} T_{comm} &= f(T_{cpu \leftarrow fpga} + T_{cpu \leftrightarrow cpu}) + f(T_{cpu \rightarrow fpga}) \\ &= \textit{non-overlapped part of } (T_{cpu \leftarrow fpga} + T_{cpu \leftrightarrow cpu}) \\ &\quad + \textit{non-overlapped part of } T_{cpu \rightarrow fpga} \\ &= (T_{step1} + T_{step2}) + (N-1) \times \\ &\quad max((T_{step1} + T_{step2}), T_{step3}) + T_{step3} \end{aligned} \tag{15}$$

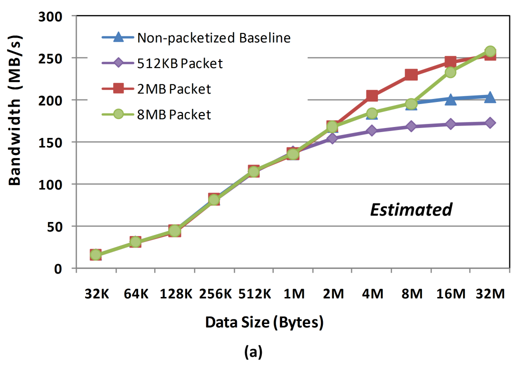
Since Steps 1 and 2 are collectively overlapped with Step 3, only the time for the greater of the two components affects the overall time of transfer. In addition, Steps 1 and 2 for the first packet and Step 3 for the last packet cannot be overlapped and are included separately. By applying Equation 15 for different packet-sizes, overall time of the transfer can be estimated for various packet sizes. Table 3 presents the estimates computed for transfer time and bandwidth when P = 2MB and P= 512KB. The input parameters (D, N, T_{step1}, T_{step2} and T_{step3}) required to compute the estimates are also listed in the table. The values reported in the table for T_{step1}, T_{step2} and T_{step3} were determined empirically using microbenchmarks on Novo-G. For example, estimates for $P =$ 2MB and $D =$ 16MB can be computed using Equation 15 as:

$$\begin{aligned} T_{comm} &= (T_{step1} + T_{step2}) + (N-1) \times \\ &\quad max((T_{step1} + T_{step2}), T_{step3}) + T_{step3} \\ &= (3 + 1.45) + (8 - 1) \times max((3 + 1.45), 8) + 8 \\ &= 68.45 \text{ ms} \\ B/W &= \frac{D}{T_{comm}} \\ &= \frac{16 \times 1024 \times 1024}{68.45 \times 10^{-3}} \times 10^{-6} = 245.1 \text{ MB/s} \end{aligned}$$

Table 3. Estimating performance of packetized transfers between two remote FPGAs on Novo-G for $P =$ 2MB and 512KB.

For P = 2MB						
D (Bytes)	N	T_{step1} (ms)	T_{step2} (ms)	T_{step3} (ms)	T_{comm} (ms)	B/W (MB/s)
512K	1	1.17	0.37	3.01	4.55	115.2
1M	1	2.00	0.73	4.99	7.72	135.8
2M	1	3.00	1.45	8.00	12.45	168.4
4M	2	3.00	1.45	8.00	20.45	205.1
8M	4	3.00	1.45	8.00	36.45	230.1
16M	8	3.00	1.45	8.00	68.45	245.1
32M	16	3.00	1.45	8.00	132.45	253.3
For P = 512KB						
D (Bytes)	N	T_{step1} (ms)	T_{step2} (ms)	T_{step3} (ms)	T_{comm} (ms)	B/W (MB/s)
512K	1	1.17	0.37	3.01	4.55	115.2
1M	2	1.17	0.37	3.01	7.56	138.7
2M	4	1.17	0.37	3.01	13.58	154.4
4M	8	1.17	0.37	3.01	25.62	163.7
8M	16	1.17	0.37	3.01	49.70	168.8
16M	32	1.17	0.37	3.01	97.86	171.4
32M	64	1.17	0.37	3.01	194.18	172.8

Following a similar approach, performance of the packetized transfers can be estimated for other packet sizes. Figure 7(a) shows the estimated bandwidth for a variety of packet sizes on Novo-G. Based on the estimates, the packet size which offers the best performance can be determined. The results indicate that for $P =$ 512KB the overall bandwidth is lower than the non-packetized baseline. The packet size of 2MB was found to offer satisfactory perfor-



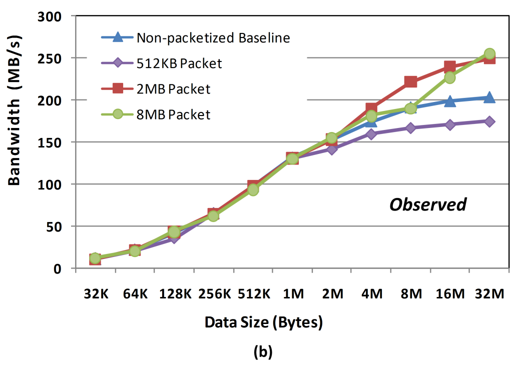


Figure 7. Bandwidth of transfers between remote FPGAs (a) Estimated by using multilevel communication model, (b) Observed experimentally on Novo-G using a testbench.

mance over a large range of data sizes under consideration. Depending on the size of the data transfer involved, the improvement obtained ranged from 11% to 24% (for the range of data sizes under consideration).

5.2 Experimental Results

Figure 7(b) shows the bandwidth of packetized transfers between two remote FPGAs for various packet sizes, recorded experimentally using a testbench on Novo-G. The trend observed from the experimental data concurs with the estimates generated using our model. While it took our team two to three hours to run microbenchmarks and compute the estimates (from the input parameters, as shown in Table 3) to determine the best packet size, the process of developing a testbench and conducting several trials to determine the best packet size experimentally took in the order of two days. The system developers can benefit greatly from reduction in their time and effort using modeling and estimation to perform such optimizations on a target system.

The errors observed between the estimated and observed bandwidth are reported in Table 4. The average of errors reported in the table is under 6%. A few cases (especially transfers involving small data sizes) experienced higher errors. For smaller data sizes, the time for intermediate steps of transfers (which forms the input to our model) became considerably small. As a result, even a minor variation in microbenchmarking results manifested as a larger relative error in the estimates computed using the model. Nevertheless, the behavior of estimated performance concurs with the observed performance, which helped us in determining the best packet size. A similar methodology can also be employed to optimize other data transfers in the SHMEM+ library.

Table 4. Relative error between estimated bandwidth and observed bandwidth of transfers between remote FPGAs.

Data Size (Bytes)	Non-packetized Baseline	512KB Packet	2MB Packet	8MB Packet
512K	15.5%	18.3%	14.7%	18.8%
1M	3.3%	5.6%	3.5%	4.1%
2M	9.3%	8.4%	8.9%	7.9%
4M	5.3%	2.5%	7.1%	1.5%
8M	2.2%	1.1%	3.6%	2.5%
16M	1.4%	0.4%	2.2%	2.5%
32M	0.5%	-1.0%	1.4%	1.0%

6. Self-tuning SHMEM+ Library

Depending on the capabilities of the interconnect technology and the I/O bus, a communication library such as SHMEM+ may have different ranges for optimal operation on different systems. For example, a certain packet size for transfers discussed in Section 5 may lead to satisfactory performance on some systems while yielding sub-optimal performance on others. In addition, the variation in the system load can also change the operational characteristics of a system. Allowing a communication library such as SHMEM+ to automatically tune itself to the capabilities of a target system can increase its usefulness in achieving portable performance.

The methodology described in Section 5 can be extended to automate the optimization process. The multilevel communication model can be embedded in the SHMEM+ library to allow it to automatically tune its performance on a target system, hence preserving performance while improving portability. Such a capability also enables applications to dynamically tune the SHMEM+ library according to the system load at several points during an application's execution.

To incorporate this feature in SHMEM+, we modified the initialization function (*shmem_init*) in the SHMEM+ library to invoke a "self-optimization" function. This function executes a series of microbenchmarks to determine the input parameters for estimation. The parameters can be the transmission times for different data sizes over the network and the I/O bus, or a set of model parameters (e.g. LogGP, PlogP) which can then be used to estimate the transmission times over these interconnects. The self-optimization function then uses the multilevel communication model to determine the packet sizes required to obtain optimal performance for various SHMEM+ routines (as demonstrated manually in Section 5.1. The information generated by the self-optimization function is also stored to a file, from which it can be later retrieved to eliminate the need for performing these tests for every execution of an application. However, if the system performance varies over a period of time, an application may choose to invoke the optimization function to re-tune the performance of the library.

6.1 Experimental Results

We augmented the SHMEM+ library with self-tuning capability and evaluated its effectiveness on several different system configurations. Figure 8 compares the bandwidth observed for transfers between remote FPGAs using the baseline SHMEM+ (without self-tuning) with the bandwidth observed for those transfers by a self-tuned SHMEM+. The Novo-G and Mu cluster differ in the interconnect technology used by the two systems. We also added diversity in the capabilities of the I/O bus by employing two different

Table 5. Packet size determined by self-tuned SHMEM+ library for transfers between two FPGAs over a range of data sizes on different systems. '-' indicates no packetization was beneficial. All numbers represent bytes of data (e.g. 1M = 1MB).

	Data Size	1K	4K	16K	64K	128K	256K	512K	1M	2M	4M	8M	16M	32M
Packet Size	Novo-G (slower DMA)	-	-	-	-	-	-	-	512K	-	2M	2M	2M	8M
	Novo-G (faster DMA)	-	-	-	-	-	-	-	-	1M	2M	2M	2M	2M
	Mu Cluster	-	-	-	-	-	-	-	512K	512K	512K	512K	512K	512K

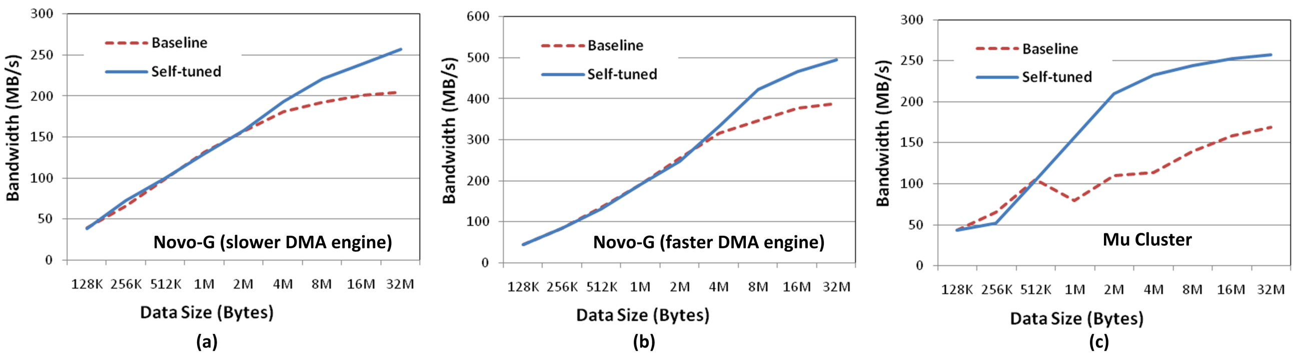


Figure 8. Bandwidth of transfers between remote FPGAs obtained by baseline SHMEM+ library and self-tuned version of SHMEM+ on (a) Novo-G employing slower DMA engine, (b) Novo-G employing faster DMA engine, (c) Mu Cluster.

Table 6. Performance improvement for transfers between remote FPGAs obtained by self-tuned SHMEM+ library vs. baseline.

Data Size (Bytes)	Novo-G (slower DMA engine)	Novo-G (faster DMA engine)	Mu Cluster
4M	6.9%	5.5%	104.3%
8M	14.5%	22.0%	74.6%
16M	18.9%	23.7%	59.3%
32M	25.1%	27.7%	52.6%

versions of the DMA engine (with varying transfer speeds) provided by the FPGA-board vendor on the Novo-G system. Figure 8 shows that the self-tuned SHMEM+ library was able to automatically determine the appropriate packet size for transfers on different systems and offer significantly improved performance. Table 5 lists the packet size that were determined to offer best performance by the self-optimization function for transfers between remote FPGAs. Table 6 highlights the improvement in bandwidth obtained by the routines in the self-tuned SHMEM+ library over the baseline version. More importantly, the self-tuning capability improves the portability of SHMEM+.

7. Conclusions and Future Work

Scalable systems employing a mix of FPGAs and other accelerators with CPUs are becoming increasingly important in the field of HPC. Programming models and libraries for heterogeneous, parallel, and reconfigurable computing such as SHMEM+ provide a mechanism for establishing communication in such systems. A communication model is an important tool for optimizing the performance of an application and developing a concrete understanding of the underlying communication infrastructure. The new multilevel communication model proposed in this paper provides a system-level representation to integrate the effect of multiple levels of communication that are routinely encountered in scalable RC systems. The model has provisions for accurately representing the opportunities for overlapping intermediate steps of communication, which is critical for obtaining high performance.

In this paper, we demonstrated the benefits of our model as a tool for performing early DSE to optimize the communication infrastructure yielding improved performance and productivity. An improvement of 42% was observed in the overall performance of our CBIR application. The communication model enabled us to simplify the process of optimizing the transfer functions in SHMEM+ for a target system. Furthermore, the model allowed us to augment the SHMEM+ library to automatically tune its performance based on the capabilities of a system, hence making SHMEM+ more portable. Improvement in performance of up to 100% was obtained in certain cases by self-tuned routines in SHMEM+.

In future research, we would like to use the communication model for optimizing more functions in the SHMEM+ library. Although this work focused mainly on reconfigurable HPC systems, the application of our model to multilevel systems based on other types of accelerators will also be explored in future research.

Acknowledgments

This work was supported in part by the I/UCRC Program of the National Science Foundation under Grant No. EEC-0642422. This work was also supported by the United States Department of Defense and used resources of the Extreme-Scale Systems Center at Oak Ridge National Laboratory. The authors gratefully acknowledge equipment and tools from Altera and GiDEL.

References

[1] V. Aggarwal, A. George, K. Yalamanchili, C. Yoon, H. Lam, and G. Stitt. Bridging parallel and reconfigurable computing with multilevel PGAS and SHMEM+. In *HPRCTA '09: Proceedings of the Third International Workshop on High-Performance Reconfigurable Computing Technology and Applications*, pages 47–54, Portland, Oregon, 2009.

[2] A. Alexandrov, M. Ionescu, K. Schauser, and C. Scheiman. LogGP: Incorporating long messages into the LogP model for parallel computation. *Journal of Parallel and Distributed Computing*, 44(1):71–79, 1997.

[3] K. Barker, K. Davis, A. Hoisie, D. Kerbyson, M. Lang, S. Pakin, and J. Sancho. Entering the petaflop era: the architecture and performance of roadrunner. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–11, Austin, Texas, 2008.

[4] J. Bosque and L. Perez. HLogGP: a new parallel computational model for heterogeneous clusters. In *Proceedings of the 2004 IEEE International Symposium on Cluster Computing and the Grid*, pages 403–410, Chicago, Illinois, 2004.

[5] D. Culler, R. Karp, D. Patterson, A. Sahay, K. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: towards a realistic model of parallel computation. *SIGPLAN Notices*, 28(7):1–12, 1993.

[6] T. El-Ghazawi, W. Carlson, and J. Draper. UPC language specifications v1.0. `http://upc.gwu.edu/docs/upc_spec_1.1.1.pdf`, February 2001.

[7] S. Fortune and J. Wyllie. Parallelism in random access machines. In *Proceedings of the 10th ACM Symposium on Theory of Computing*, pages 114–118, San Diego, California, 1978.

[8] I. Guyon, S. Gunn, M. Nikravesh, and L. Zadeh. *Feature Extraction, Foundations and Applications*. Springer, 2006.

[9] B. Holland, K. Nagarajan, and A. George. RAT: RC Amenability Test for rapid performance prediction. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 1(4):1–31, 2009.

[10] T. Kielmann, H. Bal, and S. Gorlatch. Bandwidth-efficient collective communication for clustered wide area systems. In *Proc. International Parallel and Distributed Processing Symposium (IPDPS)*, pages 492–499, San Juan, Puerto Rico, 1999.

[11] A. Lastovetsky, I.-H. Mkwawa, and M. O'Flynn. An accurate communication model of a heterogeneous cluster based on a switch-enabled ethernet network. In *Proceedings of the 12th International Conference on Parallel and Distributed Systems*, pages 15–20, Minneapolis, Minnesota, 2006.

[12] MPI. MPI standard. `http://www.mcs.anl.gov/research/projects/mpi/`.

[13] T. Ojala, M. Rautiainen, E. Matinmikko, and M. Aittola. Semantic image retrieval with HSV correlograms. In *Proc. of Twelfth Scandinavian Conference on Image Analysis*, pages 621–627, Bergen, Norway, 2001.

[14] OpenMP. The OpenMP API specification for parallel programming. `http://openmp.org/wp/`.

[15] C. Pascoe, A. Lawande, H. Lam, A. George, Y. Sun, and W. Farmerie. Reconfigurable supercomputing with scalable systolic arrays and in-stream control for wavefront genomics processing. In *Proceedings of Symposium on Application Accelerators in High-Performance Computing (SAAHPC)*, Knoxville, Tennessee, 2010.

[16] SGI. Introduction to the SHMEM programming model. `http://docs.sgi.com/library/tpl/cgi-bin/getdoc.cgi?coll=linux&db=man&fname=/usr/share/catman/man3/intro_shmem.3.html&srch=intro_shmem`.

[17] L. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.

[18] J. Williams, A. George, J. Richardson, K. Gosrani, C. Massie, and H. Lam. Characterization of fixed and reconfigurable multi-core devices for application acceleration. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 3(4), January 2011.