Exploiting Hierarchy in Heterogeneous Environments

Tiffani Williams and Rebecca Parsons
School of Electrical Engineering and Computer Science
University of Central Florida
Orlando, FL 32816-2362
{williams, rebecca}@cs.ucf.edu

Abstract

*Heterogeneous cluster environments are becoming an increasingly popular platform for executing parallel applications. Efficient heterogeneous parallel applications must account for the differences inherent in such an environment. Specifically, faster machines should possess more data items than their slower counterparts and communication should be minimized over slow network links. We propose the k-Heterogeneous Bulk Synchronous Parallel (*HBSP^k*) model, which is based on the BSP model of computation, as a framework for developing applications for heterogeneous systems. The BSP model is appropriate for* 1*-level (one communication network) heterogeneous systems.* HBSP^k *extends BSP hierarchically to address k-level heterogeneous machines. The utility of the model is demonstrated through the design and analysis of the gather and one-to-all broadcast operations. Our results indicate that the* HBSP^k *model guides the efficient design of heterogeneous parallel applications in an architecture-independent manner.*

1. Introduction

Parallel computers of yesteryear changed the way in which we tackled complex problems. However, such machines were expensive and only available to a few fortunate scientists. Trends in parallel computing indicate that distributed, heterogeneous environments will be the platform of choice for computationally intensive applications. Such environments will allow more reserchers to solve their problems by tapping into computational resources from all over the world. Furthermore, heterogeneous environments provide users with the opportunity to reuse existing computer hardware, exploit the performance of certain architectures, and share their computational resources. Despite these advantages, application developers must contend with myriad differences (such as varying computer speeds, different network protocols, and incompatible data formats) in a heterogeneous environment. Current parallel programs are not written to handle such non-uniformity. Thus, a new approach is necessary to promote the development of efficient applications for heterogeneous systems.

We propose the k-Heterogeneous Bulk Synchronous Parallel (HBSP^k) model, which is an extension of the BSP model of parallel computation [19], as a framework for the development of heterogeneous applications. The BSP model provides guidance on designing applications for good performance on homogeneous parallel machines. Furthermore, experimental results have demonstrated the utility of the model—in terms of portability, efficiency, and predictability—on diverse parallel platforms for a wide variety of non-trivial applications [8]. Since the BSP model assumes that all processors have equal computation and communication abilities, it is not appropriate for heterogeneous systems. Instead, it is only suitable for 1-level homogeneous architectures, which consist of a number of identical processors connected by a single communication network.

The HBSP^k model extends BSP hierarchically to address k-level heterogeneous cluster systems. Here, k represents the number of network levels present in the heterogeneous environment. The model describes heterogeneous clusters that are hierarchically connected by internal buses or local-, campus-, or wide-area networks. HBSP^k incorporates parameters that reflect the relative computational and communication speeds at each of the k levels. In multi-level heterogeneous environments, communication costs at different levels of the hierarchy can differ by an order of magnitude or more [12]. As a result, heterogeneous parallel algorithms must be adapted to minimize communication on slower network links. Similarly, computation should be minimized on slower machines. Under HBSP^k, improved performance results from effectively exploiting the relative speeds of the heterogeneous computing components.

Collective communication algorithms are used frequently as building blocks in a variety of parallel algo-

rithms. Proper implementation of these operations is vital to the efficient execution of the parallel algorithms that use them. Collective communication algorithms designed for traditional parallel machines are not adequate for heterogeneous environments. As a result, we present two collective communication algorithms—gather and one-to-all broadcast—for hierarchically-structured, heterogeneous cluster environments. We refer the reader to [20] for a description of additional HBSP^k collective communication algorithms and an experimental analysis of their performance. Our design strategy for these algorithms is two-fold. Faster machines should be more involved in the computation than their slower counterparts. Secondly, faster machines should receive more data items than slower machines. Based on these observations, our algorithms demonstrate the advantages of using the HBSP^k model as a basis for the development of heterogeneous applications.

The rest of this paper is organized as follows. Section 2 gives a brief overview of related work. The HBSP^k model is described in Section 3. Section 4 describes the design and analysis of collective operations for the HBSP^k model. Experimental results are shown in Section 5. Conclusions and directions for future research are given in Section 6.

2. Related work

The Bulk Synchronous Parallel (BSP) [19] model provides the foundation for the HBSP^k model. The BSP model provides guidance on designing applications for good performance on homogeneous parallel machines. Support for BSP can be found in [3, 2, 7, 8], which present theoretical results, empirical results, and experimental parameterization of BSP programs. However, BSP only addresses homogeneous platforms with a single communication network.

Two models that address 1-level heterogeneous platforms are the Heterogeneous Coarse-Grained Multicomputer (HCGM) model [17], a generalization of the CGM [5] model of parallel computation, and the Heterogeneous Bulk Synchronous Parallel (HBSP) [21], which is synonymous with HBSP^1. Both of these models take into account varying processor speeds to develop parallel algorithms for heterogeneous systems. The main difference between the two models is that HCGM is not intended to be an accurate predictor of execution times whereas HBSP attempts to provide the developer with predictable algorithmic performance.

Several research efforts focus on designing collective operations for heterogeneous systems [1, 10, 15]. Additional work demonstrates the importance of collective communication operations for hierarchical networks. Husbands and Hoe [10] develop MPI-StarT, a system that efficiently implements collective routines for a cluster of SMPs. Kielmann *et al.* [14] present MagPie, a system that also handles a two-level communication hierarchy. Karonis *et al.* [12] design a topology-aware version of the broadcast operation for good performance. The P-$logP$ model [13], an extension of LogP [4], is used to optimize performance of wide-area collective operations by determining the optimal tree shape for the communication. Moreover, large messages are split into smaller units resulting in better link utilization.

3. The HBSP^k model

The k-Heterogeneous Bulk Synchronous Parallel (HBSP^k) model is a generalization of the BSP model [19] of parallel computation. HBSP^k provides parameters that allow the user to tailor the model to the required system. As a result, HBSP^k can guide the development of applications for traditional parallel systems, heterogeneous clusters, the Internet, and computational grids [6]. In HBSP^k, each of these systems can be grouped into subtrees (or clusters) based on their ability to communicate with each other relative to a particular level. Although the model accommodates a wide-range of architecture types, the algorithm designer does not have to worry with manipulating an overwhelming number of parameters. More importantly, HBSP^k allows the algorithm designer to think about the collection of heterogeneous computers as a single system.

3.1. Machine representation

The HBSP^k model refers to a class of machines with at most k different levels of communication. Thus, HBSP^0 or single processor systems is the simplest class of machines. The next class of machines is HBSP^1 computers, which consist of at most one communication network. Examples of HBSP^1 computers include single processor systems (i.e. HBSP^0), traditional parallel machines, and heterogeneous workstation clusters. HBSP^2 machines extend the HBSP^1 class to handle heterogeneous collections of multiprocessor machines or clusters. Figure 1 shows an HBSP^2 cluster consisting of three HBSP^1 machines. In general, HBSP^k systems include HBSP^{k-1} computers as well as machines composed of HBSP^{k-1} computers. Thus, the relationship of the machine classes is HBSP^0 $\subset$ HBSP^1 $\cdots$ $\subset$ HBSP^k.

An HBSP^k machine can be represented by a tree $T = (V, E)$. Each node of T represents a heterogeneous machine. The height of the tree is k. The root r of T is an HBSP^k machine. Let d be the length of the path from the root r to a node x. The level of node x is $k - d$. Nodes at level i of T are HBSP^i machines. Furthermore, a machine can play different roles at different levels. Figure 2 shows a tree representation of the HBSP^2 machine shown in Figure 1. The root node corresponds to an HBSP^2 machine. The components of this machine (a symmetric multiprocessor, an SGI workstation, and a LAN) are shown at level 1.

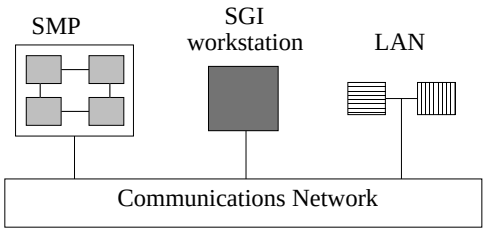

Figure 1. An HBSP^2 cluster consisting of an SMP machine, an SGI workstation, and a LAN.

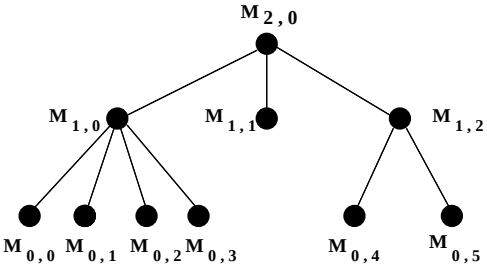

Figure 2. A tree representation of the cluster shown in Figure 1.

Level 0 depicts the individual processors of the symmetric multiprocessor and the LAN.

The indexing scheme of an HBSP^k machine is as follows. Machines at level i, $0 \leq i \leq k$, are labeled $M_{i,0}$, $M_{i,1}$, ..., M_{i,m_i-1}, where m_i represents the number of HBSP^i machines. Consider machine $M_{i,j}$ of an HBSP^k computer, where $0 \leq j < m_{i,j}$. One possible interpretation of $M_{i,j}$ is that it is a *cluster* with identity j on level i. The nodes of the cluster are the children of $M_{i,j}$. In Figure 1, $M_{1,0}$ is an HBSP^1 cluster composed of the nodes $M_{0,0}, M_{0,1}, M_{0,2}$, and $M_{0,3}$. $M_{2,0}$ provides an example of a cluster of clusters. The HBSP^k model places no restriction on the amount of nesting within clusters. Additionally, we may consider the machines at level i of T the *coordinator nodes* of the machines at level $i-1$. As shown in Figure 1, $M_{1,0}$, $M_{1,1}, M_{1,2}$, and $M_{2,0}$ are examples of coordinator nodes. Coordinator nodes fulfill many roles. They can act as a representative for its cluster during intercluster communication. Additionally, to increase algorithmic performance, they may represent the fastest machine in their subtree (or cluster). Under this assumption, the root node is the fastest node of the entire HBSP^k machine.

3.2. HBSP^k computation

An HBSP^k machine consists of HBSP^i machines, where $0 \leq i \leq k$. As a result, an HBSP^k computation consists of some combination of super^i-steps. During a super^i-step, each level i coordinator node performs asynchronously some combination of local computation, message transmissions to other level i coordinator nodes, and message arrivals from its peers. A message sent in one super^i-step is guaranteed to be available to the destination machine at the beginning of the next super^i-step. Each super^i-step is followed by a global synchronization of all the level i nodes.

Consider the class of HBSP^0 machines. For these single processor systems, computation proceeds through a series of super^0-steps (or steps). Communication and synchronization with other processors is not applicable. Unlike the previous class, HBSP^1 machines perform communication. HBSP^1 computers proceed through a series of super^1-steps (or supersteps). During a superstep, each HBSP^0 machine performs asynchronously some combination of local computation, message arrivals, and message transmissions. Thus, an HBSP^1 computation resembles a BSP computation. The main difference between the two is that an HBSP^1 algorithm delegates more work to the faster processors.

For HBSP^2 machines, computation consists of super^1- and super^2-steps. Super^1-steps proceed as described previously. During a super^2-step, the coordinator nodes for each HBSP^1 cluster performs local computation and/or communicates data with other level 1 coordinator nodes. A barrier synchronization of these coordinators separate each super^2-step.

It is interesting to note that a super^i-step could be defined recursively. For expositional purposes, we chose the the approach of defining it iteratively.

3.3. Model parameters

An HBSP^k computer is characterized by the following parameters, which are summarized in Table 1:

- m_i, the number of HBSP^i machines labeled $M_{i,0}, M_{i,1}, \cdots M_{i,m_i-1}$ on level i, where $0 \leq i \leq k$;
- $m_{i,j}$, the number of children of $M_{i,j}$;
- g, a bandwidth indicator that reflects the speed with which the *fastest* machine can inject packets into the network;
- $r_{i,j}$, the speed relative to the *fastest* machine for $M_{i,j}$ to inject a packet into the network;
- $L_{i,j}$, overhead to perform a barrier synchronization of the machines in the subtree of $M_{i,j}$;

Symbol	Meaning
$M_{i,j}$	a machine's identity, where $0 \leq i \leq k$, $0 \leq j < m_i$
m_i	number of HBSP^i machines on level i
$m_{i,j}$	number of children of $M_{i,j}$
g	speed the *fastest* machine can inject packets into the network
$r_{i,j}$	speed relative to the *fastest* machine for $M_{i,j}$ to inject packets into the network
$L_{i,j}$	overhead to perform a barrier synchronization of the machines in the jth cluster of level i
$c_{i,j}$	fraction of the problem size that $M_{i,j}$ receives
h	size of a heterogeneous h-relation
$h_{i,j}$	largest number of packets sent or received by $M_{i,j}$ in a super^i-step
$T_i(\lambda)$	execution time of super^i-step λ

Table 1. Definitions of Notations

- $c_{i,j}$, the fraction of the problem size that $M_{i,j}$ receives.

We assume that the $r_{i,j}$ value of the fastest machine is normalized to 1. If $r_{i,j} = t$, then $M_{i,j}$ communicates t times slower than the fastest node. The $c_{i,j}$ parameter adds a load-balancing feature into the model. Specifically, it attempts to provide $M_{i,j}$ with a problem size that is proportional to its computational and communication abilities. The HBSP^k model says nothing about how the parameter values should be calculated. Instead, it assumes that such costs have been determined appropriately.

3.4. Cost model

The parameters described above allow for cost analysis of HBSP^k programs. First, consider the cost of a super^i-step. w_i represents the largest amount of local computation performed by a level i node. Let $h_{i,j}$ be the largest number of messages sent or received by $M_{i,j}$, where $0 \leq j < m_i$. The size of the *heterogeneous h-relation* is $h = \max\{r_{i,j} \cdot h_{i,j}\}$ with a routing cost of gh. Thus the execution time of super^i-step λ is

$$T_i(\lambda) = w_i + gh + L_{i,j}. \quad (1)$$

The overall cost is the sum of the super^i-step times.

Similarly to BSP, the above cost model demonstrates what factors are important when designing HBSP^k applications. To minimize execution time, the programmer must attempt to (i) balance the local computation in each super^i-step, (ii) balance communication between the machines, and (iii) minimize the number of super^i-steps. Balancing these objectives can be quite difficult. Part of the difficulty arises from the fact that there are multiple layers of heterogeneity. However, HBSP^k provides guidance on how to design efficient heterogeneous programs.

Besides analyzing execution time, the HBSP^k model can be used to determine the penalty associated with using a particular heterogeneous environment. In particular, there are additional overheads incurred by algorithms executing on HBSP^k platforms because of the synchronization and communication costs incurred at each level. HBSP^k gives the programmer information regarding these costs. Not all problems will be able to exploit the capabilities offered by these systems, since the application must tolerate the latencies inherent in using hierarchical platforms. The HBSP^k model provides the user with ways to manipulate these costs and gives them a cost related to executing the application in this environment.

4. Collective communication algorithms

Collective communication plays an important role in the development of parallel programs. It simplifies the programming task, facilitates the implementation of efficient communication schemes, and promotes portability. In this section, we use the HBSP^k model as a platform to develop and analyze two collective communication algorithms. The *gather* operation uses a single node to collect a unique message from each of the other nodes. In the *one-to-all broadcast*, only the source process has the data that needs to be broadcast. At the termination of the procedure, each node has a copy of the data.

4.1. Algorithm design

The HBSP^k model provides parameters that allow algorithm designers to exploit the heterogeneity of the underlying system. The model promotes our two-fold design strategy for HBSP^k collective operations. First, faster machines should be involved in the computation more often than their slower counterparts. Collective operations use specific nodes to collect or distribute data to the other nodes in the system. For faster algorithmic performance, these nodes should be the fastest machines in the system. Secondly, faster machines should receive more data items than slower machines. This principle encourages the use of *balanced* workloads, where machines receive problem sizes relative to their communication and computational abilities. Partitioning the workload so that nodes receive an equal number of elements works quite well for homogeneous environments. However, this strategy encourages *unbalanced* workloads in heterogeneous environments since faster machines typically sit idle waiting for slower nodes to finish a computation.

Our HBSP^1 algorithms are based on BSP communication operations [11]. In an HBSP^1 environment, the number of workstations is $m_{1,0}$ (or m_0). The single coordinator node, $M_{1,0}$, represents the fastest workstation among the

HBSP^0 machines. Hence, $r_{1,0} = 1$. $L_{1,0}$ is the cost of synchronizing the processors. The HBSP^2 collective routines use the HBSP^1 algorithms as a basis. Unlike HBSP^1 platforms, HBSP^2 machines contain a two-level communication network. Level 0 consists of m_0 individual workstations. Level 1 represents the m_1 (or $m_{2,0}$) coordinator nodes for the machines at level 0. Each coordinator, $M_{1,j}$, requires a cost of $L_{1,j}$ to synchronize the nodes in its cluster, where $0 \leq j < m_1$. The root of the entire cluster is $M_{2,0}$, which is the fastest machine. Therefore, $r_{2,0} = 1$. Communication between HBSP^1 and HBSP^2 machines can be quite expensive. As a result, our HBSP^2 algorithms minimally communicate on these links. We do not specify algorithms for higher-level machines (i.e., $k \geq 3$). However, one can generalize the approach given here for these systems.

In the following subsections, let $x_{i,j}$ represent the number of items in $M_{i,j}$'s possession, where $0 \leq i \leq 2, 0 \leq j < m_i$. Balanced workloads assume $x_{i,j} = c_{i,j}n$. The total number of items of interest is n. For notational convenience, the indexes f and s identify the fastest and slowest nodes, respectively. Lastly, to simplify notation, we write n when the correct quantity is some value less than n.

4.2. HBSP^1 gather

In the HBSP^1 algorithm, each level-0 node, $M_{0,j}$, sends its data elements to its coordinator, $M_{1,0}$, where $0 \leq j < m_0$. Here, $M_{0,j}$ sends $x_{0,j}$ (or $c_{0,j} \cdot n$) items to $M_{1,0}$. At the end of the algorithm, $M_{1,0}$ possesses all n elements.

Analysis. The algorithm above consists of only one super^1-step. Therefore, the size of the single, heterogeneous h-relation is $\max\{r_{0,j} \cdot c_{0,j} \cdot n, r_{1,0} \cdot n\}$. Each HBSP^0 processor's $r_{0,j}$ value is relative to the fastest processor. Hence, $r_{1,0} = 1$ and $r_{0,j} \geq r_{1,0}$. Recall that $c_{0,j}$ is inversely proportional to the speed of $M_{0,j}$. Consequently, $r_{0,j}c_{0,j} < 1$. Thus, the HBSP^1 gather cost is $gn + L_{1,0}$.

The above cost of the gather operation is efficient since the fastest processor is performing most of the work. If $r_{0,j}c_{0,j} > 1$, $M_{0,j}$ has a problem size that is too large. Its communication time will dominate the cost of the gather operation. Whenever possible, the fastest processor should handle the most data items. Our analysis demonstrates the importance of balanced workloads, the number of elements per machine is compatible with its computational and communication abilities. The increase in performance is a result of $M_{1,0}$ receiving the items faster. The HBSP^k model, as does BSP, rewards programs with balanced design.

4.3. HBSP^2 gather

The HBSP^2 gather algorithm proceeds as follows. First, each HBSP^1 machine performs an HBSP^1 gather. Afterwards, each of the level 1 nodes send their data items to the root, $M_{2,0}$.

Analysis. The HBSP^2 gather algorithm proceeds as follows. First, each HBSP^1 machine performs an HBSP^1 gather. Afterwards, each of the level 1 nodes send their data items to the root, $M_{2,0}$. Since the problem size is n, $x_{2,0} = n$. The cost of an HBSP^2 gather operation is the sum of the super^1- and super^2-step times. Since each HBSP^1 machine performs a gather operation, the super^1-step cost is the largest time needed for an HBSP^1 cluster to finish the operation. Once the level 1 coordinators have the n data items, they send the data to the root. This super^2-step requires $g \cdot \max\{r_{1,j} \cdot x_{1,j}, r_{2,0} \cdot n\} + L_{2,0}$. Assuming balanced workloads, the super^2-step reduces to $gn + L_{2,0}$. Efficient algorithm execution in this environment implies that the size of the problem size must outweigh the cost of performing the extra level of communication and synchronization.

4.4. HBSP^1 broadcast

In an HBSP^1 broadcast, the computation starts at the root node, where each of its children execute similarly to the two-phase BSP algorithm. $M_{1,0}$ distributes equally the n elements to each of its children. The second phase consists of each processor receiving n elements.

Analysis. As a result, $M_{0,j}$ receives $\frac{n}{m_{1,0}}$ items from $M_{1,0}$. This phase requires a heterogeneous h-relation of size $\max\{r_{1,0} \cdot n, r_{0,j} \cdot \frac{n}{m_{1,0}}\}$. In a typical environment, it is reasonable to assume that $m_{1,0}$ ranges from the tens to the hundreds. It is quite unlikely that a machine would communicate $m_{1,0}$ times slower than the fastest machine. If this is the case, it may be more appropriate not to include that machine in the computation. As a result, the communication time of the first phase reduces to gn. Since each processor must receive the same number of items, the slowest processor will cause a bottleneck. Let $r_{i,s}$ represent the communication time of the slowest node at level i. This results in a communication time of $gr_{0,s}n$. Thus, the complexity of a two-phase broadcast on an HBSP^1 machine is $gn(1 + r_{0,s}) + 2L_{1,0}$.

As a point of comparison, the one-phase broadcast ($M_{1,0}$ sends n items to each processor, $M_{0,j}$) costs $gnm_{2,0} + L_{1,0}$. For reasonable values of $r_{0,s}$, the two-phase approach is the better overall performer. An interesting conclusion concerning the broadcast operation is that it effectively cannot exploit heterogeneity. Since the slowest processor must receive n items, its cost will dictate the complexity of the algorithm. Partitioning the problem size based on the $c_{i,j}$ parameter is ineffective. Although wall clock performance may improve, theoretically, the resulting speedup is negligible.

4.5. HBSP^2 broadcast

The two-phase approach is the algorithm of choice for HBSP^1 machines. Next, we consider broadcasting in an HBSP^2 computer. Given that communication is likely to be more expensive (i.e., higher latency links and increased synchronization costs) in such an environment, we investigate whether the two-phase approach is also applicable for HBSP^2 machines. The algorithm begins with the root node distributing n items to the level 1 coordinator nodes. $M_{2,0}$ may broadcast the data to its children using either a single-phase or two-phase approach. Afterwards, each level 1 coordinator sends the n items to its children using the HBSP^1 broadcast algorithm.

Analysis. The total cost of the algorithm is the sum of the super^1- and super^2-steps. Since both approaches utilize the HBSP^1 broadcast, we focus our discussion on the behavior of the super^2-steps.

In the one-phase approach, the root node sends n elements to the level 1 machines. The cost of the super^2-step is $g \cdot \max\{r_{1,s}n, r_{2,0}nm_{2,0}\} + L_{2,0}$. Suppose that $r_{1,s} > m_{2,0}$. The super^2-step cost is $gr_{1,s}n + L_{2,0}$. Otherwise, its $gr_{2,0}nm_{2,0} + L_{2,0}$. Unlike the above algorithm, the two-phase approach requires two super^2-steps. Initially, the root node sends $\frac{n}{m_{2,0}}$ elements to the level 1 coordinators. Each coordinator, then, broadcasts its $\frac{n}{m_{2,0}}$ elements to the other coordinators. The first super^2-step requires a heterogeneous h-relation of size $\max\{r_{1,s}\frac{n}{m_{2,0}}, r_{2,0}n\}$. The other super^2-step costs $gr_{1,s}n + L_{2,0}$. Suppose that $r_{1,s} > m_{2,0}$. The cost of the super^2-steps is $gr_{1,s}n(\frac{1}{m_{2,0}} + 1) + 2L_{2,0}$. Otherwise, the cost is $gn(r_{1,s} + r_{2,0}) + 2L_{2,0}$.

Based on the above costs, the HBSP^2 broadcast algorithm cannot effectively exploit heterogeneity either. Again, the problem arises from the slowest machine having to receive each of the n items. Furthermore, the analysis shows that broadcasting on an HBSP^2 machine can be expensive because of the extra levels of synchronization and communication. However, if the problem size is large enough, these additional costs can be overcome.

5. Experimental results

In this section, we focus on experimentally validating the performance of the HBSP^1 collective communication algorithms presented in the previous section. The complete details of our experimental approach can be found in [20]. We find it useful to simplify the notation of the HBSP^k model for this environment. The number of workstations is m_0 or p. The single coordinator node, $M_{1,0}$ or P_f, represents the fastest processor among the HBSP^0 processors. P_s refers to the slowest node. To identify the individual processors on level 0, we use the notation P_j to refer to processor $M_{0,j}$. Lastly, the problem size of interest is n.

5.1. Experimental setup

The HBSP^1 collective communication algorithms are implemented using the HBSP Programming Library (HBSPlib), which incorates many of the functions (i.e., message passing, synchronization, enquiry) contained in BSPlib [9]. HBSPlib is written on top of PVM [18], a software package that allows a heterogeneous network of parallel an serial computers to appear as a single, concurrent, computational resource. HBSPlib also incorporates primitives that allow the programmer to take advantage of the heterogeneity of the underlying system. Such functions return the rank of a processor as well as guide the programmer toward balanced workloads.

Our experimental testbed consisted of a non-dedicated heterogeneous cluster of ten SUN and SGI workstations at the University of Central Florida. Each node is connected by a 100Mbit/s Ethernet connection. Our experiments evaluate the impact of processor speed and workload distribution on the overall performance of an algorithm. The ranking of processors is determined by the BYTEmark benchmark [16], which consists of tests such as sorting, floating-point manipulation, and numerical analysis to evaluate the performance of a machine.

The input data for each experiment consists of 100 KBytes to 1000 KBytes of uniformly distributed integers. The problem size refers to the largest number of integers possessed by the root. Experimental results are given in terms of an improvement factor. Let T_A and T_B represent the execution time of algorithm A and algorithm B, respectively. The improvement factor of using algorithm B over algorithm A is $\frac{T_A}{T_B}$.

The HBSP^k model encourages the use of fast processors and balanced workloads. According to the model, applications that embody both of these principles will result in good performance. We designed two types of experiments to validate the predictions of the model. The first experiment tests whether processor speed has an impact on algorithmic performance. Let T_s represent the execution time of a collective routine assuming the root node is the slowest processor, P_s. T_f denotes the algorithmic cost of using P_f as the root. For these experiments, each processor has an equal number of data items since our objective is to monitor the performance of slow versus fast root nodes. Hence, $c_j = \frac{1}{p}$. Our second experiment studies the benefit of balanced workloads. For these experiments, the root node is always the fastest processor. Let T_u be the execution time when the workload is unbalanced. Note that $T_u = T_f$. T_b denotes the execution time when the workload is balanced. Here, c_j is computed using the BYTEmark results.

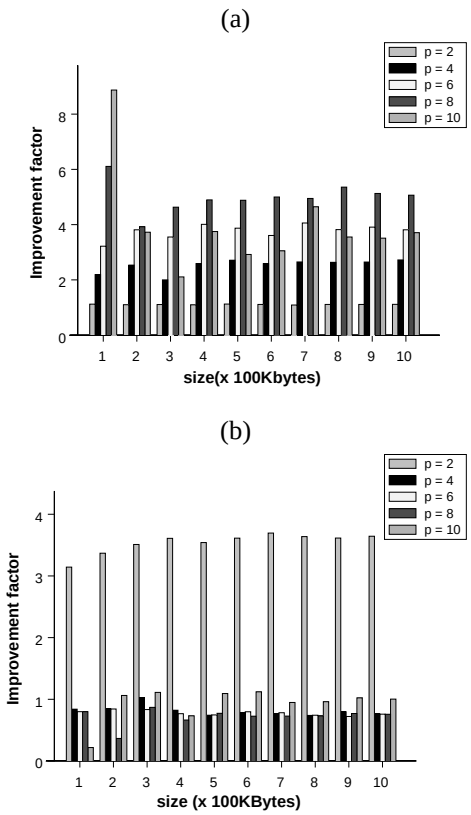

Figure 3. Gather performance. The improvement factor is determined by (a) $\frac{T_s}{T_f}$ and (b) $\frac{T_u}{T_b}$.

5.2. Gather

Figure 3 (a) shows the improvement that results if the root node is P_f. As the number of processors increase, so does performance. The improvement factor is steady across all problem sizes. Unfortunately, there is virtually no benefit to distributing the workload based on a processor's computational abilities, except at $p = 2$. Figure 3 (b) displays the results. The problem lies with the estimation of c_j for the second fastest processor, who sends too many elements to the root node, P_f. Since the second fastest processor's workload does not match its abilities, everyone must wait for it to finishing sending its items to the root node.

For both experiments, the results at $p = 2$ are interesting. First, Figure 3 (a) shows that it is better for the root node to be the slowest workstation. This seems counterintuitive. In our implementation of gather (as well as the other collective operations), a processor does not send data to itself. When P_s is the root, P_f sends $\frac{n}{p}$ items to it. Similarly, if the fastest processor is the root, P_f sends $\frac{n}{p}$ elements to P_s. $T_s < T_f$ implies that it is more beneficial to have P_f waiting on data from P_s. It is clear that the root node should be P_f as the number of processors increase. Unlike the situation at $p = 2$, P_f does not sit idle waiting on data items from P_s. Instead, it handles the messages of the other processors while waiting on the slowest processor's data.

5.3. One-to-all broadcast

Figure 4 (a) compares the execution time of the algorithm assuming the root node is either P_s or P_f. The plot demonstrates that their is neglible improvement in performance. The HBSP^k model predicted this behavior. In fact, the improvement in performance is a result of P_f distributing $\frac{n}{p}$ integers to each processor during the first phase of the algorithm. Our analysis also holds if P_j receives $c_j n$ elements during the first phase of the algorithm. Figure 4 (b) clearly demonstrates that there is no benefit to balanced workloads since each processor must receive all of the items.

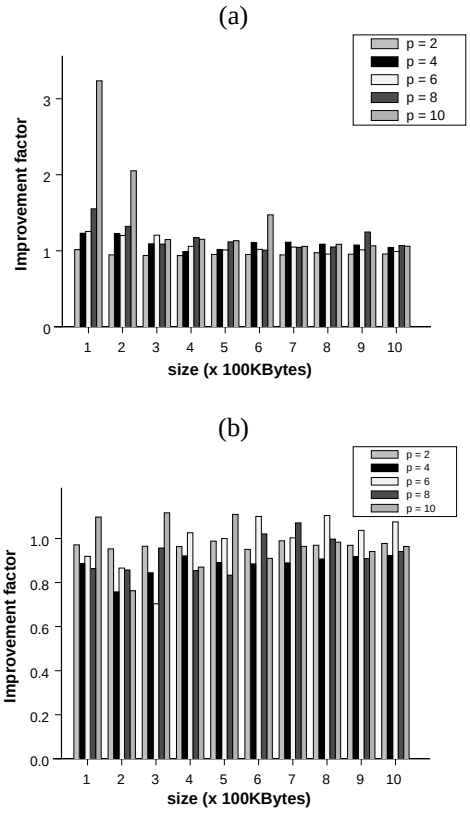

Figure 4. One-to-all broadcast performance. The improvement factor is determined by (a) $\frac{T_s}{T_f}$ and (b) $\frac{T_u}{T_b}$.

6. Conclusions and Future Work

The HBSP^k model provides a framework for the development of parallel applications for k-level heterogeneous platforms. HBSP^k extends the BSP model by incorporating parameters that reflect the relative speeds of the heterogeneous computing components. Although the HBSP^k model is somewhat more complex than BSP, it captures the most important aspects of heterogeneous systems. The utility of the model is demonstrated through the design and analysis of the gather and one-to-all broadcast algorithms. Our results indicate that HBSP^k encourages balanced workloads among machines, if applicable. For example, a close examination of the broadcast operation demonstrates that it is impossible to avoid unbalanced workloads since the slowest machine must receive n items. Overall, the performance of our collective operations is quite impressive (complete results are shown in [20]). Fundamental changes to the algorithms are not necessary to attain an increase in performance. Instead, modifications consist of selection the root node and distributing the workload.

In conclusion, HBSP^k offers a single-system image of a heterogeneous platform to the application developer. By hiding the non-uniformity of the underlying system from the application developer, the HBSP^k model offers an environment that encourages the design of heterogeneous parallel software in an architecture-independent manner. Extensions to this work include designing HBSP^k applications that can take advantage of our efficient heterogeneous communication algorithms. Moreover, we plan to investigate extending the $r_{i,j}$ parameter to accommodate communication costs incurred by $M_{i,j}$ as a result of sending data to various destinations.

References

[1] M. Banikazemi, V. Moorthy, and D. Panda. Efficient collective communication on heterogeneous networks workstations. In *International Conference on Parallel Processing*, pages 460–467, 1998.

[2] R. H. Bisseling. Sparse matrix computations on bulk synchronous parallel computers. In *Proceedings of the International Conference on Industrial and Applied Mathematics*, Hamburg, July 1995.

[3] R. H. Bisseling and W. F. McColl. Scientific computing on bulk synchronous parallel architectures. In B. Pehrson and I. Simon, editors, *Proceedings of the 13th IFIP World Computer Congress*, volume 1, pages 509–514. Elsevier, 1994.

[4] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a realistic model of parallel computation. In *Fourth ACM Symposium on Principles and Practice of Parallel Programming*, pages 1–12, May 1993.

[5] F. Dehne, A. Fabri, and A. Rau-Chaplin. Scalable parallel computational geometry for coarse multicomputers. In *Proc. ACM Symposium on Computational Geometry*, pages 298–307, 1993.

[6] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1998.

[7] A. V. Gerbessiotis and L. G. Valiant. Direct bulk-synchronous parallel algorithms. *Journal of Parallel and Distributed Computing*, 22(2):251–267, August 1994.

[8] M. W. Goudreau, K. Lang, S. B. Rao, T. Suel, and T. Tsantilas. Portable and efficient parallel computing using the BSP model. *IEEE Transactions on Computers*, 48(7):670–689, 1999.

[9] J. M. D. Hill, B. McColl, D. C. Stefanescu, M. W. Goudreau, K. Lang, S. B. Rao, T. Suel, T. Tsantilas, and R. Bisseling. BSPlib: The BSP programming library. *Parallel Computing*, 24(14):1947–1980, 1998.

[10] P. Husbands and J. C. Hoe. MPI-StarT: Delivering network performance to numerical applications. In *Supercomputing '98*, 1998.

[11] B. Juurlink and H. Wijshoff. Communication primitives for bsp computers. *Information Processing Letters*, 58(6):303–310, 1996.

[12] N. T. Karonis, B. R. D. Supinski, I. Foster, and W. Gropp. Exploiting hierarchy in parallel computer networks to optimize collective operation performance. In *14th International Parallel and Distributed Processing Symposium*, pages 377–384, 2000.

[13] T. Kielmann, H. Bal, and S. Gorlatch. Bandwidth-efficient collective communicaiton for cluster wide area systems. In *14th International Parallel and Distributed Processing Symposium*, pages 492–499, 2000.

[14] T. Kielmann, R. F. H. Hofman, H. E. B. A. Platt, and R. A. F. Bhoedjang. MPI's reduction operations in clustered wide area systems. In *Message Passing Interface Developer's and User's Conference*, pages 43–52, Atlanta, GA, March 1999.

[15] B. B. Lowekamp and A. Beguelin. Eco: Efficient collective operations for communication on heterogeneous networks. In *International Parallel Processing Symposium*, pages 399–405, Honolulu, HI, 1996.

[16] B. Magazine. The BYTEmark benchmark. URL http://www.byte.com/bmark/bmark.htm, 1995.

[17] P. Morin. Coarse-grained parallel computing on heterogeneous systems. In *Proceedings of the 1998 ACM Symposium on Applied Computing*, pages 629–634, 1998.

[18] V. S. Sunderam. PVM: a framework for parallel distributed computing. *Concurency: Practice and Experience*, 2(4):315–349, 1990.

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

[20] T. L. Williams. *A General-Purpose Model for Heterogeneous Computation*. Ph.D. dissertation, University of Central Florida, Orlando, December 2000.

[21] T. L. Williams and R. J. Parsons. The heterogeneous bulk synchronous parallel model. In *Parallel and Distributed Processing*, volume 1800 of *Lecture Notes in Computer Science*, pages 102–108. Springer-Verlag, Cancun, Mexico, May 2000.