Packet Size Optimization for Supporting Coarse-Grained Pipelined Parallelism

Wei Du
Department of Computer and Information Sciences
Ohio State University, Columbus OH 43210
duw@cis.ohio-state.edu

Gagan Agrawal
Department of Computer and Information Sciences
Ohio State University, Columbus OH 43210
agrawal@cis.ohio-state.edu

ABSTRACT

The emergence of grid and a new class of *data-driven* applications is making a new form of parallelism desirable, which we refer to as *coarse-grained pipelined* parallelism. In this paper, we focus on the problem of choosing *packet size*, i.e., the unit of transfer between the pipeline units, in exploiting this form of parallelism.

We develop an analytical model for this purpose. Because the pipeline includes both communication and computation phases, the frequency and/or volume of communication between different phases can be different. We consider two models, fixed-frequency and fixed-size, and derive mathematical expressions for both.

We have carried out detailed evaluation of our models using three applications, executed with different parameters and datasets. Our experiments show that the choice of packet size makes a significant difference in the execution time, and the packet sizes suggested by the model result in the lowest or very close to the lowest possible execution time.

1. INTRODUCTION

The work presented in this paper is in the context of *coarse-grained pipelined parallelism*. Two recent trends are making this form of parallelism feasible and desirable. The first trend is the emergence of grid computing. A grid environment facilitates better sharing of data and computing resources. Particularly, the availability of data repositories and access to data collection instruments and sensors is creating a new scenario for execution of many applications.

The second trend is the emergence of a new class of data-driven or data-intensive applications. This class includes scientific data analysis, data mining, data visualization, and image analysis. These applications are typically both compute and data intensive, and require fast or even interactive response time. Therefore, these applications are a suitable target for parallelization.

Consider the execution of such data-driven applications in scenarios where the data is available on a repository or a data collection site on the internet, and the final results are required on a user's desktop. It is usually not possible to perform all analysis at the site hosting such a shared data repository or a data collection instrument. Similarly, networking and storage limitations make it impossible to download all data at a single site before processing. Thus, the application needs to be broken into a process or stage that executes on the site hosting or collecting the data, one or more stages that executes on clusters or SMP machines, and a final stage that executes on the user's local machine. We refer to such a model of execution as coarse-grained parallel pipelined execution.

Many research groups have developed runtime support and scheduling techniques for this class of applications [2, 21, 26]. In our recent work, we have developed language and compiler support for this form of parallelism [5].

One important factor that impacts the execution of applications with this model is *packet size*, i.e, the unit of data transfer from one pipeline stage to another. Clearly, a very large packet size may prevent the pipelined parallelism from being exploited. At the same time, a very small packet size could lead to high overheads because of communication latencies. Thus, choosing a suitable packet size is important for performance.

In this paper, we develop and validate analytical models for determining the packet size that will result in the lowest execution time. Our work builds on the work of Wang *et al.* on packet size optimization for multi-link communication pipeline [24]. The key difference in our work is that computation is involved in the pipeline and needs to be modeled. Moreover, because of computation, the frequency and/or the volume of communication at different stages can be different.

We consider two distinct models, *fixed-frequency* and *fixed-size* communication. In the fixed-frequency model, all computation stages except the last one send out one packet after getting one as input. The size of the output packet may be different from the size of the input packet. In the fixed-size model, the computation stages always communicate same size or almost same size packets. To meet this requirment, the stages may need to wait and process more than one input packets, so the frequency could be different. We have developed analytical models for determining the optimal packet size for both of the above models.

We have carried out a detailed evaluation of our work using three applications, executed with different parameters and datasets. The main observations from our experiments are as follows. First, we show that the choice of packet size can make a large difference in the execution time. Second, we have shown that the packet sizes chosen by our model

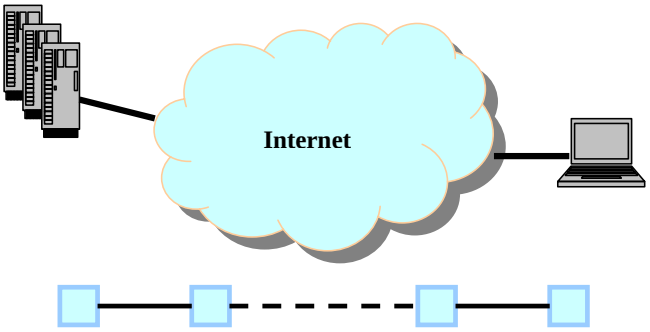

Figure 1: A Simple Pipeline. The first stage is on the site hosting the data repository, the last stage is on the user's desktop, there are one or more stages in between performing transformation and processing.

result in the lowest or very close to the lowest possible execution time. With little extra effort, our models can be applied to applications running on the Grid environment.

The rest of the paper is organized as follows. Background information on our target class of applications and the coarse-grained pipelined execution model is sketched in Section 2. The packet size problem and our basic approach are discussed in Section 3. The details of analytical model for the different cases are presented in Section 4. Section 5 focuses on the experimental validation of our models. We compare our work with related research efforts in Section 6 and conclude in Section 7.

2. TARGET APPLICATIONS AND COARSE-GRAINED PIPELINED PARALLELISM

In this section, we describe our target applications and the scenario for their execution in a distributed environment. We describe the *coarse-grained pipelined parallel* execution model and explain why it is a good match for our target applications.

Our focus is on a variety of data-driven applications, arising in domains like scientific data analysis, data mining, visualization, and image analysis, and spanning both scientific and commercial interests.

Processing and analyzing large volumes of data plays an increasingly important role in many domains of scientific research. In many applications, because of computational and storage requirements, and to ensure fault tolerance and high availability, datasets are stored on storage systems distributed across a wide area network, creating a science data grid [18, 17]. The types of datasets that a data grid can store includes, among others, simulations of time-dependent phenomena that periodically generate snapshots of their state [4, 13, 22, 19, 14], archives of raw and processed remote sensing data [16, 11, 15], and archives of medical images [1, 6]. A variety of analysis and processing can be performed on these scientific datasets and images.

Business decisions today are increasingly driven by analysis of data. Various data mining and On Line Analytical Processing (OLAP) techniques are used in decision support systems. Grid technologies are driving the creation of *virtual organizations*, which comprise collections of institutions or entities sharing and analyzing information [7]. These virtual organizations can be expected to use a variety of business decision support functionalities that have been commonly used in centralized organizations. Thus, we expect that data mining and OLAP will be common operations performed in a grid environment.

Two obvious ways of executing an application that processes datasets available in grid-based data repositories are, downloading all data at the user's local machine, or performing all computations at sites hosting data repositories. However, neither of these two options is feasible in most situations. Sufficient storage space and/or network bandwidth is not likely to be available to download all data. At the same time, a node hosting a data repository is not likely to offer sufficient computing power to execute the entire application.

Therefore, a natural option for executing these applications is to use a pipeline of computing resources, where the site hosting the data repository is the first stage and the user's local machine where the results are required is the final stage. Typically, one or more clusters and/or SMP machines serve as the intermediate stage(s). An example pipeline is shown in Figure 1.

We refer to the above model of execution as the *coarse-grained pipelined parallelism*. The processing associated with an application is carried out in several stages. These stages are executed on a pipeline of computing units, each of which handles the intermediate results obtained from the previous stage. Typically, the first stage in this pipeline is the unit where the input data is available, and the last stage is where the final results are viewed. Many research groups have developed runtime support and scheduling techniques for this class of applications [2, 21, 26]. In our recent work, we have developed language and compiler support for this form of parallelism [5].

Mapping of our target class of applications to the pipelined model is facilitated by an important observation. Our study of a variety of scientific and commercial data intensive applications shows that *generalized reduction operations* are very common in the processing structure. This observation applies across a large number of scientific data intensive applications [4, 13, 19, 14, 16, 11, 15], data mining algorithms [10] including association mining, clustering, and decision tree construction, OLAP applications involving algebraic and distributed aggregations [8], and key visualization algorithms such as the ones for isosurface rendering [12]. Processing for generalized reductions consists of three main steps: (1) Retrieving data items of interest, (2) Applying application-specific transformation operations on the retrieved input items, and, (3) Mapping the input items to output items and aggregating, in some application specific way, all the input items that map to the same output data item. Most importantly, aggregation operations involve *commutative* and *associative* operations, i.e., the correctness of the output data values does not depend on the order input data items are aggregated. For simplicity, we can refer to the first two steps as *local processing*, and the third step as *global combination*.

The common structure identified above has an important implication. Different steps involved in local processing can be performed independently on different sections of data. This enables exploitation of pipelined parallelism, as well as shared memory or distributed memory parallelism available at an intermediate stage of the pipeline.

Our work is being implemented and evaluated using DataCutter, which is an existing runtime system for supporting pipelined parallelism in a grid environment [3, 2]. Specifically, DataCutter supports a *filter-stream* model of execution. Typically, one filter executes one stage of the pipeline, using one or more input *streams*. The results of the processing are packed and sent as one or more output *streams*.

3. PACKET SIZE OPTIMIZATION PROBLEM

This section formulates the packet size optimization problem that we are addressing in this paper. We also illustrate the various cases we are considering. Detailed results for each of these cases are presented in Section 4.

3.1 Overview of the Problem

As we had stated previously, in our target setting, the application is decomposed into several stages and mapped onto a pipeline of computing and communication units. The first stage of the pipeline is always on the site hosting the data repository and is responsible for reading, subsetting, and forwarding the data to the following stages. The last stage is typically on a user's desktop and in charge of collecting and viewing the results.

Suppose the entire operation involves reading and processing a data file of size B. One possibility is to transmit and process this file as one big packet. This option is likely to be impractical, because of the limited memory that may be available at different stages of the pipeline. Moreover, this option does not allow overlapping of the operations at different stages of the pipeline. Now, consider another extreme, where a single data-item is received, processed, and forwarded at each processing stage. Again, this option is likely to give poor performance, because of the high communication latency and the overhead involved in switching between communication and computation at each node. Thus, an important optimization parameter for getting high performance from the pipeline is the granularity at which packets are received, processed, and forwarded at each pipeline unit.

3.2 Solution Approach

Optimization of packet size in a multi-link communication pipeline has been researched by Wang *et al.* [24]. While their formulation can form the basis for our analysis, their results are not directly applicable to the problem we are considering. This is because of the following two reasons:

- Our pipeline includes both communication and computation stages. Both of them need to be modeled for determining the impact of packet size on the overall execution time.
- Because of computation, the volume and/or the frequency of communication between the different stages can vary. This is not directly handled by Wang *et al.*'s work.

Consider a pipeline with m filters or computing stages. There are $m-1$ streams connecting these stages. For our analysis, we consider it as a pipeline with $n = 2m-1$ stages. Similar to Wang *et al.*, we consider the communication time to be an affine function of the size of the packet communicated, i.e., it comprises a fixed overhead per message and a per-byte cost. Based upon our experiences with our target class of applications, we find that the same model can be used for computation stages. There is usually a fixed overhead associated with receiving a packet and starting the processing. The remaining cost is usually linear in the size of the packet.

To present our analysis, we use the following terminology.

- n: the number of pipeline stages, including both computation and communication stages
- G_i: the fixed per-packet *overhead* for stage i
- g_i: the per-byte cost for stage i
- B: the size of the entire data file
- k: the number of packets
- $t_{i,j}$: the time the ith packet spends in the jth stage, $t_{i,j} = size\ of\ packet\ i * g_j + G_j$

We still need to be able to handle the fact that the volume and/or the frequency of communication between different stages may not be the same. This is because an output packet of a computation stage is most likely different from its input packet, in both content and size. We introduce α_i as the ratio of output size to input size for the stage i. Obviously, this value is equal to 1 for all communication stages. We also assume that the value is same for all packets passing through stage i.

In our analysis, we will consider two models of execution. These models are *fixed-frequency* and *fixed-size* communication. In the fixed-frequency model, all computation stages except the last one send out one packet after getting one as input. The size of the output packet may be different from the size of the input packet. In the fixed-size model, the computation stages always communicate same size or almost same size packets. That is, the computation stages might need to wait for more than one input packets in order to prepare an output packet of the required size. Note that it is possible to execute the pipeline in a way that both the frequency and size of packets communicated may be different. We do not consider this possibility in this paper.

To choose the packet size to minimize the execution time, we need expressions for the total execution time. Before elaborating the different cases we are handling, we review the basic results from Wang *et al.* [24]. The expression for execution time is based upon the notion of a *bottleneck* stage. This is the stage which is always busy after the first packet is received, and before the last packet exits it. We assume that the b^{th} stage is the bottleneck stage, and is characterized by the two-tuple (G_b,g_b). Then the total execution time of a program can be stated as:

$$T = T_f + T_b + T_l \quad (1)$$

where T_f, T_b, and T_l are defined as follows:

- T_f: the time the first packet takes to reach the bottleneck stage.
- T_b: the time all packets spend in the bottleneck stage.
- T_l: the time the *last packet* takes to exit the pipeline after leaving the bottleneck stage.

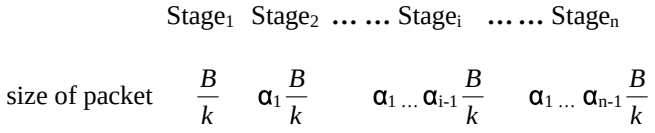

Figure 4: Changing Size of the Packet Flowing Through the Pipeline Shown in Figure 3

Figure 2 shows the execution time-line of a pipeline consisting of three computing stages and two communication stages.

Note that the above formulation assumes that there is only a single bottleneck stage in the pipeline. However, this does not limit our analysis in any way. Suppose, there are two bottleneck stages in a pipeline. Then, the amount of time spent by all packets in these two stages will be identical. Therefore, we will derive the same expression by considering either of these two as the bottleneck.

4. CHOOSING OPTIMAL PACKET SIZES

In this section, we derive the expressions for choosing the optimal packet size. Recall that B is the total size of input file, k is the number of packets used, and $p = \lceil \frac{B}{k} \rceil$ is the size of each packet. We will aim at determining the value of k in our analysis. In the next two subsections, we focus on fixed-frequency and fixed-sized communication patterns, respectively.

4.1 Fixed-Frequency Communication Pattern

The expression for minimizing the execution time depends upon which stage is the bottleneck stage. Here the bottleneck stage is defined as the pipeline stage which has the largest processing time for each incoming packet. Considering the packet size change pattern shown in Figure 4, the formal definition of bottleneck stage b under the fixed-frequency communication model is as follows.

$$b = \left\{ i \left| \left(\prod_{j=1}^{i-1} \alpha_j \right) \frac{B}{k} g_i + G_i = max \left\{ \left(\prod_{l=1}^{j-1} \alpha_l \right) \frac{B}{k} g_j + G_j \right\}, j = 1, \cdots, n \right. \right\}.$$

We separately consider two different cases, corresponding to bottleneck at the first stage, and bottleneck at any of the other stages.

4.1.1 Bottleneck at the First Stage

Figure 3 shows the execution time-line for a 1st-stage bottleneck pipeline, and Figure 4 illustrates how the size of a packet changes in the pipeline. We rewrite T_f, T_b and T_l as follows.

$$T_f = 0 \tag{2}$$

$$T_b = B * g_1 + k * G_1 \tag{3}$$

$$\begin{aligned} T_l &= (\alpha_1 \frac{B}{k} * g_2 + G_2) + (\alpha_1 \alpha_2 \frac{B}{k} * g_3 + G_3) + \cdots \\ &\quad (\alpha_1 \alpha_2 \cdots \alpha_{i-1} \frac{B}{k} * g_i + G_i) + \cdots + \\ &\quad (\alpha_1 \alpha_2 \cdots \alpha_{n-1} \frac{B}{k} * g_n + G_n) \end{aligned} \tag{4}$$

Substituting equations 2, 3 and 4 in the equation 1, we get

$$\begin{aligned} T &= T_f + T_b + T_l \\ &= B * g_1 + k * G_1 + \\ &\quad (\alpha_1 \frac{B}{k} * g_2 + G_2) + (\alpha_1 \alpha_2 \frac{B}{k} * g_3 + G_3) + \cdots \\ &\quad (\alpha_1 \alpha_2 \cdots \alpha_{i-1} \frac{B}{k} * g_i + G_i) + \cdots + \\ &\quad (\alpha_1 \alpha_2 \cdots \alpha_{n-1} \frac{B}{k} * g_n + G_n) \\ &= B * g_1 + k * G_1 + \sum_{i=2}^{n} \left(\left(\prod_{j=1}^{i-1} \alpha_j \right) \frac{B}{k} * g_i + G_i \right) \end{aligned} \tag{5}$$

Since $\frac{d^2T}{d^2k} > 0$, to obtain the value of k such that T is minimized, we differentiate the above equation with respect to k, and set the result to 0.

$$\frac{dT}{dk} = -\frac{B \sum_{i=2}^{n} \left(\prod_{j=1}^{i-1} \alpha_j \right) g_i}{k^2} + G_1 = 0 \tag{6}$$

Solving equation 6, we can get

$$k = \left\lceil \sqrt{\frac{B \sum_{i=2}^{n} \left(\prod_{j=1}^{i-1} \alpha_j \right) g_i}{G_1}} \right\rceil. \tag{7}$$

4.1.2 Bottleneck at Second or Later Stages

The execution time-line of a pipeline with bottleneck at second or later stages is given in Figure 5. The change in the size of the packet is already depicted by Figure 4. Now $Stage_b$ is the bottleneck stage, where $b \neq 1$. We rewrite T_f, T_b and T_l as follows.

$$\begin{aligned} T_f &= (\frac{B}{k} * g_1 + G_1) + (\alpha_1 \frac{B}{k} * g_2 + G_2) + \cdots + \\ &\quad (\alpha_1 \alpha_2 \cdots \alpha_{i-1} \frac{B}{k} * g_i + G_i) + \cdots + \\ &\quad (\alpha_1 \alpha_2 \cdots \alpha_{b-2} \frac{B}{k} * g_{b-1} + G_{b-1}) \end{aligned} \tag{8}$$

$$T_b = \alpha_1 \alpha_2 \cdots \alpha_{b-1} B * g_b + k * G_b \tag{9}$$

$$\begin{aligned} T_l &= (\alpha_1 \alpha_2 \cdots \alpha_b \frac{B}{k} * g_{b+1} + G_{b+1}) + \cdots + \\ &\quad (\alpha_1 \alpha_2 \cdots \alpha_{n-1} \frac{B}{k} * g_n + G_n) \end{aligned} \tag{10}$$

Substituting equations 8, 9 and 10 in the equation 1, differentiating the resulting equation with respect to k, and setting the result to 0, then solving the equation, we get

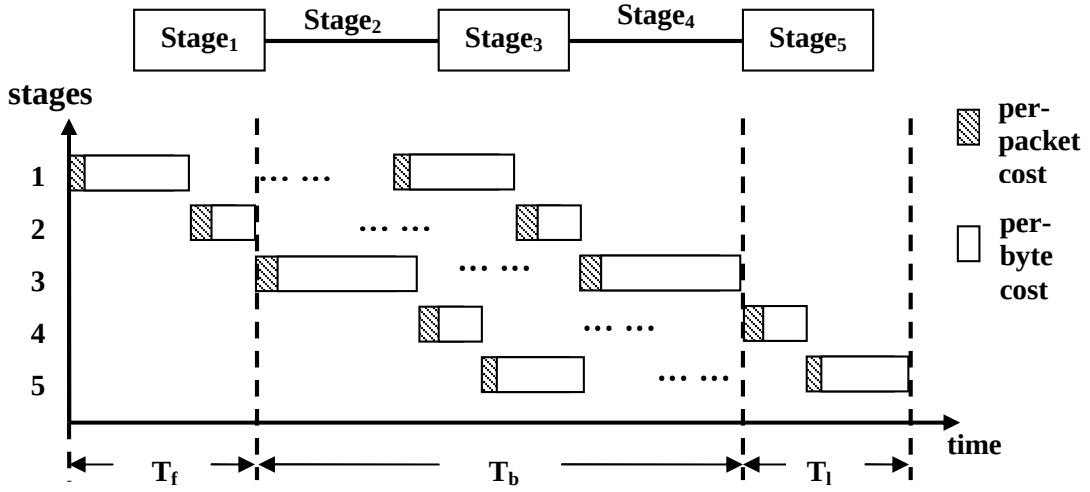

Figure 2: Execution Time-line of a Pipeline with Three Computation Stages and Two Communication Stages. $Stage_3$ is the bottleneck stage.

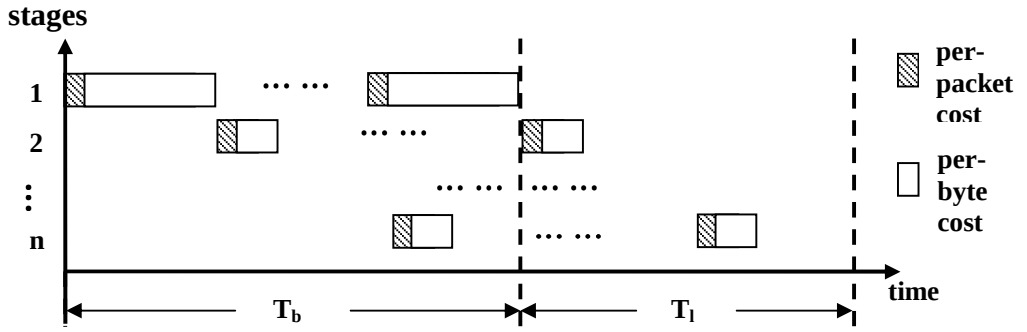

Figure 3: Execution Time-line of A *1st*-Stage Bottleneck Pipeline under Fixed-Frequency Communication Pattern

$$k = \left\lceil \sqrt{\frac{B\left(g_1 + \sum_{i \neq b}\left(\prod_{j=1}^{i-1}\alpha_j\right) g_i\right)}{G_b}} \right\rceil . \qquad (11)$$

4.2 Fixed-Size Communication Pattern

This scheme differs from the previous one in that there are fixed-size packets flowing on the communication links, except the last packet. Hence, communication frequency varies depending on the packet content and the filtering condition. Such pattern may save us some overhead. On the other hand, it can also introduce more idle time for later stages, which in turn increases the total runtime.

$$b = \left\{ i \left| \prod_{j=1}^{i-1}\alpha_j \left(\frac{B}{k}g_i + G_i\right) = max\left\{ \prod_{j=1}^{l-1}\alpha_j \left(\frac{B}{k}g_l + G_l\right)\right\}, l = 1, \cdots, n \right.\right\} \qquad (12)$$

As mentioned earlier, α_i is the ratio of output size with respect to input size for stage i. So to have an output packet whose size is same as that of the input, stage i needs to process $\left\lceil \frac{1}{\alpha_i} \right\rceil$ packets before sending out a packet. Here, we define the bottleneck stage b as in 12.

Intuitively, we compare the time spent at each stage to process the original packet, and the slowest one is the bottleneck stage. We again consider the two cases, corresponding to bottleneck at the first stage, and bottleneck at second or later stages. The latter is discussed first.

4.2.1 Bottleneck at Second or Later Stages

Figure 6 depicts the execution time-line for a pipeline with bottleneck at second or later stages. The size of a packet flowing in the pipeline is $\frac{B}{k}$. Note that longer delays for later stages are possible here, due to the fixed-size communication requirement. Each stage i needs to process $\left\lceil \frac{1}{\alpha_i} \right\rceil$ input packets when preparing the output. Taking into account these factors, T_f, T_b and T_l could be rewritten as follows.

$$\begin{aligned} T_f = & \left\lceil \frac{1}{\alpha_1} \right\rceil \left(\frac{B}{k} * g_1 + G_1\right) + \cdots + \\ & \left\lceil \frac{1}{\alpha_i} \right\rceil \left(\frac{B}{k} * g_i + G_i\right) + \cdots + \\ & \left\lceil \frac{1}{\alpha_{b-1}} \right\rceil \left(\frac{B}{k} * g_{b-1} + G_{b-1}\right) \end{aligned} \qquad (13)$$

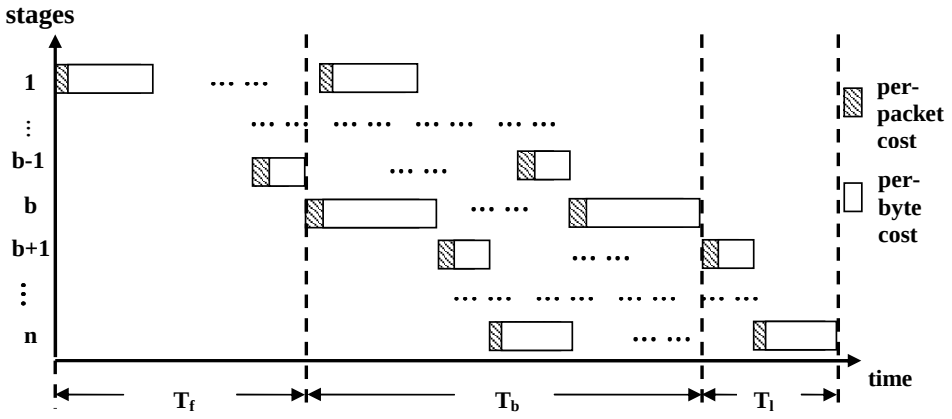

Figure 5: Execution Time-line of a Pipeline With Bottleneck at Second or Later Stages: Fixed-Frequency Communication Pattern

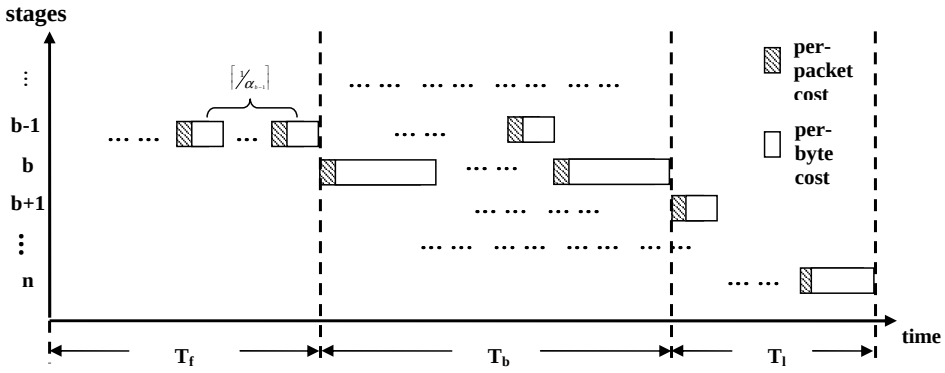

Figure 6: Execution Time-line of a Pipeline With Bottleneck at Second or Later Stages: Fixed-Size Communication Pattern

$$T_b = \alpha_1\alpha_2\cdots\alpha_{b-1}k(\frac{B}{k}g_b + G_b) \tag{14}$$

$$T_l = (\frac{B}{k} * g_{b+1} + G_{b+1}) + \cdots + (\frac{B}{k} * g_n + G_n) \tag{15}$$

Substituting equations 13, 14 and 15 in the equation 1, then differentiating the resulting equation with respect to k, and setting the result to 0, we get

$$\frac{dT}{dk} = -\frac{B\left(\sum_{i<b}\frac{g_i}{\alpha_i} + \sum_{i>b} g_i\right)}{k^2} + \left(\prod_{i=1}^{b-1}\alpha_i\right)G_b = 0 \tag{16}$$

Solving equation 16, we can get

$$k = \left\lceil \sqrt{\frac{B\left(\sum_{i<b}\frac{g_i}{\alpha_i} + \sum_{i>b} g_i\right)}{\left(\prod_{i=1}^{b-1}\alpha_i\right)G_b}} \right\rceil . \tag{17}$$

4.2.2 Bottleneck at First Stage

This is a special case of Section *4.2.1*, as there is no idle time before the bottleneck stage begins to work. The execution time-line of such a pipeline is given in Figure 7. We rewrite T_f, T_b and T_l as follows.

$$T_f = 0 \tag{18}$$

$$T_b = B * g_1 + k * G_1 \tag{19}$$

$$T_l = (\frac{B}{k} * g_2 + G_2) + \cdots + (\frac{B}{k} * g_n + G_n) \tag{20}$$

Substituting equations 18, 19 and 20 in the equation 1, and following the steps shown in section *4.1.1*, we get

$$k = \left\lceil \sqrt{\frac{B\sum_{i=2}^{n} g_i}{G_1}} \right\rceil . \tag{21}$$

We summarize results from all models in table 1. There are several observations from these results.

1. The total execution time T can be represented by the summation of two simple linear functions. $T = ak + \frac{b}{k}$, where $a, b > 0$. Figure 8 plots these two functions. The cross point(A) of these two lines gives the optimal value of k. Also if a is small, the plot will be more like the dotted line, then the variation of T will not be very significant over the range right to A, and not too far from A. Otherwise, if a is large, a sharper curve could be expected.
2. The final optimal solution only depends on the value of G_b, α_i, and the values of g_i for non-bottleneck stages. Factors such as the per-byte cost for bottleneck stage and per-packet cost for non-bottleneck stages do not have any impact of the optimal value of k.

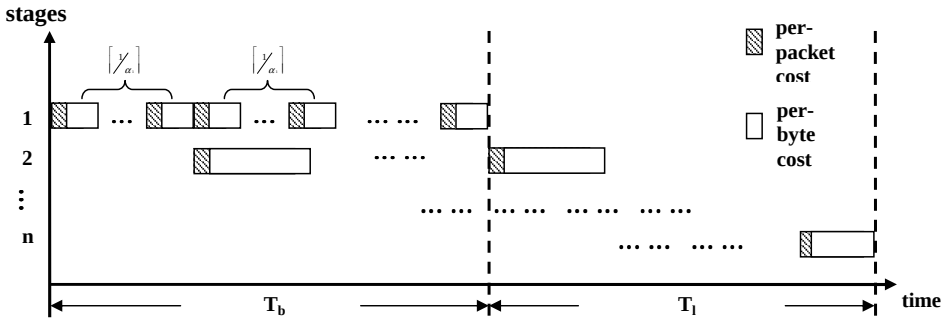

Figure 7: Execution Time-line of a *1st* Stage Bottleneck Pipeline under Fixed-Size Communication Pattern

Communication Pattern	Bottleneck Stage b	k
fixed-frequency	b=1	$k = \sqrt{\frac{B\sum_{i=2}^{n}(\prod_{j=1}^{i-1}\alpha_j)g_i}{G_1}}$
	$b \neq 1$	$k = \sqrt{\frac{B\left(g_1+\sum_{i\neq b}(\prod_{j=1}^{i-1}\alpha_j)g_i\right)}{G_b}}$
fixed-size	b=1	$k = \sqrt{\frac{B\sum_{i=2}^{n} g_i}{G_1}}$
	$b \neq 1$	$k = \sqrt{\frac{B\left(\sum_{i<b}\frac{g_i}{\alpha_i}+\sum_{i>b} g_i\right)}{(\prod_{i=1}^{b-1}\alpha_i)G_b}}$

Table 1: Summary of All Models

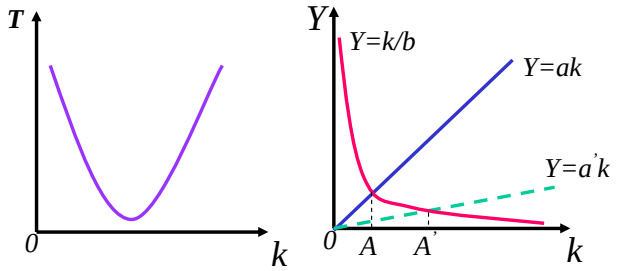

Figure 8: T can be represented as summation of two simple linear functions

4.3 Applicability to the Grid Environment

An obvious question is, how can the analytical models we have derived be used in a grid environment, given the variability of the resources in such an environment. Clearly, our models will need the fixed (G_i) and per-byte (g_i) cost for both computing and communication phases. We believe that the use of coarse-grained pipelined model will require the availability of dedicated computing resources. Thus, the fixed and per-byte cost for computing phases will not change during the execution of an application. These costs can be obtained during an initial execution of the application on the same or similar environment and used as part of our models.

The communication bandwidth and latency are more likely to vary dynamically in a grid environment. To obtain information about these, existing tools such as the Network Weather Service (NWS) [25] can be used. The closed-form expressions we have derived can be used with newly obtained parameters and optimal packet size can be computed without a significant runtime overhead.

5. EXPERIMENTAL RESULTS

This section reports a series of experiments we conducted for validating our work. Our experiments have focused on two aspects: 1) demonstrating that the choice of packet size can make a significant difference on the overall execution time of our target applications, and 2) showing that the packet sizes suggested by the expressions we have derived give best or very close to the best performance.

5.1 Experimental Setting and Applications

In the long run, we expect that pipelined parallelism can be exploited in wide-area networks. However, this is going to require high bandwidth networks and certain level of quality of service support. Recent trends are clearly pointing in this direction, for example, the five sites that are part of the NSF funded Teragrid project expect to be connected with a 40 Gb/second network [20]. However, for our study, we did not have access to a wide-area network that gave high bandwidth and allowed repeatable experiments. Therefore, all our experiments were conducted within a single cluster. The cluster we used had 700 MHz Pentium machines connected through Myrinet LANai 7.0.

In the set of applications we have focused on, the following four computation stages are common:

- Read (R) stage, which is responsible for reading a chunk of points from the data file and prepares packets for sending to subsequent stages.
- A filtering or subsetting stage, denoted by F, which

filters out points based upon a relatively inexpensive test.

- A local processing stage, denoted by L, which performs the processing independently on each packet.
- A global processing stage, denoted by G, which combines local results to compute the final results.

It is obvious that $\alpha_1 = 1$ since the R stage does not perform any processing on the input data before it forwards them to next stage. Similarly, for all communication stages $\alpha_i = 1, i = 2, 4, 6$.

We use three applications to test our model, two of which are algorithms implementing isosurface rendering. They are, z-buffer based isosurface rendering and active pixels based rendering, referred to as `ZBUF` and `ACTP`, respectively. Isosurface rendering is a key visualization problem. The inputs to the problem are a three-dimensional grid, a scalar isosurface value, and a two-dimensional viewing screen with an angle associated with it. The goal is to view a surface, as seen from the given viewing angle, which captures the points in the grid where the scalar value matches the given isosurface value.

Our third application is k-nearest neighbor search, referred to as `KNN`. It is one of the basic data mining problems [9]. Here, the training samples are described by an n-dimensional numeric space. Given a new point and a range, the goal is to find the k training samples that are closest to the new point within the specific range. This application also involves four stages.

We created two versions for both `KNN` and `ZBUF`, based upon certain choices of parameters. `KNN-1st`, `ACTP` and `ZBUF-1st` are versions in which the first phase of the pipeline is the bottleneck stage, while in `KNN-2nd` and `ZBUF-2nd`, the second computation stage is the bottleneck stage. These versions allow us to validate expressions for different cases that we had listed in the previous section.

In our experiments, we have run these applications under both fixed-frequency and fixed-size communication schemes with various inputs sizes.

5.2 Measuring Pipeline Parameters

For using the expressions we have derived for optimal packet sizes, we need to know the fixed overhead and the per-byte cost associated with each stage of an application. We now describe how these values are obtained.

Note that we need to execute an application at least once to obtain these values. However, this is still a lot easier than iteratively finding the optimal packet size. Moreover, the values associated with a communication phase are independent of the application and can be determined just once for the target execution environment. Similarly, if some phases are common between multiple applications, the parameters can be determined once and used later.

To determine the fixed (G_i) and per-byte (g_i) cost for a computation phase, we identify the program components that represent these costs. This is illustrated by the following example, taken from the *Range_Query* filter code of the K-nearest neighbor search algorithm. The functionality shown here is to get an input packet from the input stream, and process it point by point.

```
while (1)
    {
(1)   if ((pbuf = arg.ins[0].read()) == NULL) break;
(2)   pbufptr = pbuf->getPtr();
(3)   memcpy(&nump, pbufptr, sizeof(int));
(4)   pbufptr += sizeof(int);
(5)   for (int i=0; i<nump; i++)
       {
(6)      t = new KPOINT();
         ... ...
       }
      ... ...
     }
```

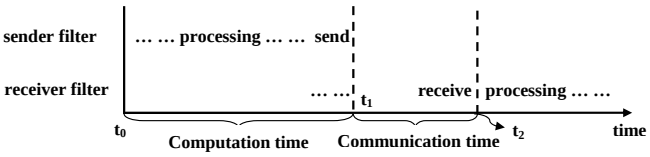

Figure 9: A Time-line Illustrating the Division of Computation and Communication Time

In the above code, the cost to execute statements (2), (3) and (4) does not change with the input packet size, and is considered as part of the fixed overhead. The cost for executing *for* statement is determined by the value of *nump*, which is the number of points contained in an input packet. So, the execution time of all statements inside *for* loop is considered as the per-byte cost. We instrument the code to measure both of the above. The parameters we obtained for the computational phases for all applications are summarized in Table 2.

We used the following procedure to determine the parameters associated with a communication phase. It is important to note that this is not strictly the time required over the communication link. The time spent in the middleware in preparing or receiving a message is also included in the communication time. This is illustrated through the Figure 9. The communication time starts when sender finishes the *send* operation, and ends when receiver finishes the *receive* operation, i.e. it is of duration $t_2 - t_1$ in Figure 9.

For the communication links in the cluster we used, G_i was 32.6 μs and g_i was 1.73 μs, for very communication phase i.

5.3 Evaluation of Fixed-Frequency Model

We now present experimental results validating the expressions we have derived for the fixed-frequency model.

5.3.1 K-Nearest Neighbor Search

In this experiment, we test the `KNN-1st` with $K = 3$, $B = 108M$, and $\alpha_3 = 75.85\%$. In Figure 10, we show how the execution time of this application varies as the packet size is changed. As can be seen from the figure, execution time can vary by up to a factor of 3 as the packet size is changed. However, the lowest execution time can be achieved from choosing any packet size from a significant range of values.

The packet size suggested by our model can be calculated as follows. Using the parameters from Table 2 and the Equa-

Communication Pattern	Applications	Parameters			
Fixed-frequency	KNN-1st	G_1	g_3	g_5	α_3
		2.41	0.054	0.013	75.85%
	ZBUF-2nd	G_3	g_1	g_5	α_3
		2.1606	0.1289	0.7569	3.42%
	ACTP	G_1	g_3	g_5	α_3
		1.5338	0.1494	0.2191	2.95%
Fixed-size	KNN-2nd	G_3	g_1	g_5	α_3
		2.4087	0.0872	0.0127	1
	ZBUF-1st	G_1	g_3	g_5	α_3
		1.3848	0.1207	1.0296	1
	ACTP	G_1	g_3	g_5	α_3
		1.4552	0.1356	0.1937	1

Table 2: Parameters for Computational Phases for All Applications

tion 7, we get

$$k = \left\lceil \sqrt{\frac{108 * 10^6(1.73 + 0.054 + 75.85\% * (1.73 + 0.013))}{2.41}} \right\rceil = 11798$$

then

$$\left\lceil \frac{B}{k} \right\rceil = 9154 \text{ bytes}$$

From Figure 10, we can see that this value of packet size can give the lowest execution time. Thus, our model suggests a packet size that minimizes execution time.

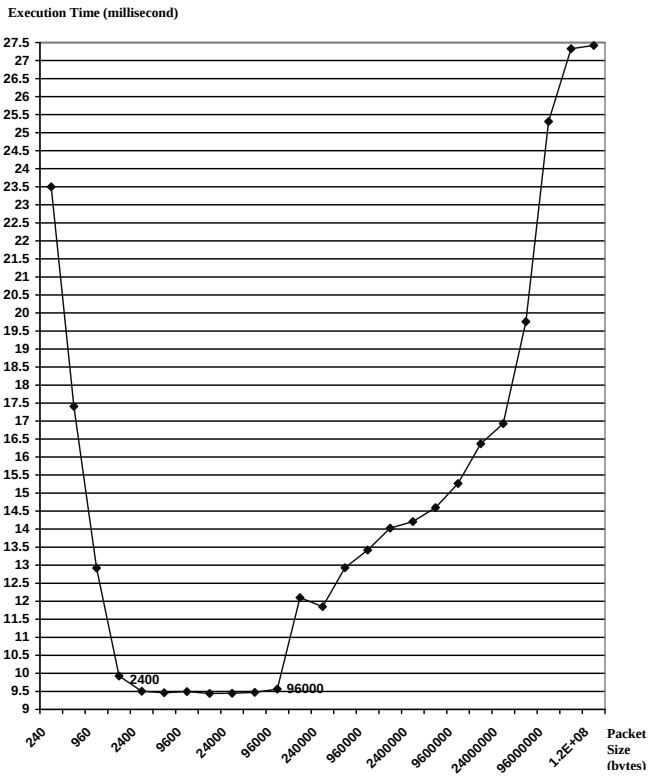

Figure 10: Runtime of KNN-1st under Fixed-Frequency Scheme

5.3.2 *Zbuffer Based ISO-Surface Rendering*

We test the application `ZBUF-2nd` here. The size of data file B is about 145M, the output image size is 512x512, and value of α_3 is 3.42%. Calculating k by taking parameters in Table 2 and using the Equation 11, we get

$$k = \left\lceil \sqrt{\frac{152729808(0.1289 + 1.73 + 3.42\% * (1.73 + 0.7569)}{2.1606}} \right\rceil = 11726$$

then

$$\left\lceil \frac{B}{k} \right\rceil = 13025 \text{ bytes}$$

The impact of packet size on the execution time is shown in Figure 11. Two things can again be observed from this Figure. First, the choice of the packet size can have a large impact on execution time. Second, the packet size suggested by our model minimizes the execution time.

5.3.3 *ActivePixel Based ISO-Surface Rendering*

The size of data file B is about 145M, the output image size is 2048x2048, and value of α_3 is 0.0295. Calculating k by taking parameters in Table 2 and using the Equation 7, we get

$$k = \left\lceil \sqrt{\frac{152729808(1.73 + 0.1494 + 0.0295 * (1.73 + 0.2191)}{1.5338}} \right\rceil = 13888$$

then

$$\left\lceil \frac{B}{k} \right\rceil = 10997 \text{ bytes}$$

The variation in execution time as the packet size is changed is shown in Figure 12. Again, we see that execution time can vary significantly and the packet size suggested by our model achieves the lowest execution time.

5.4 Evaluation of Fixed-Size Model

We now validate the expressions we have derived for the fixed-size communication model.

5.4.1 *K-Nearest Neighbor Search*

In this experiment, we test the `KNN-2nd` with $K = 200$, $B = 108M$, and $\alpha_3 = 1$. Using the parameters in Table 2

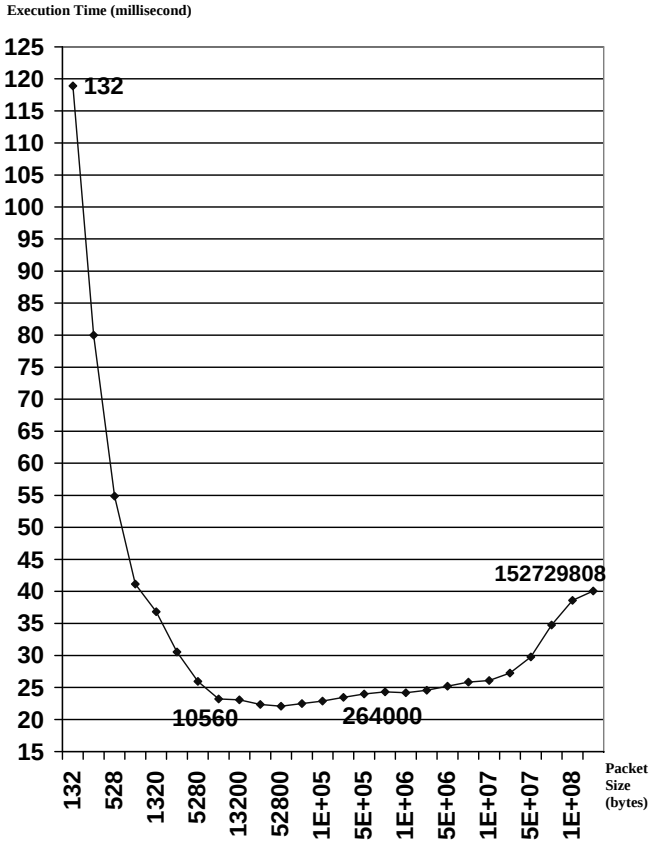

Figure 11: Runtime of Zbuffer Based ISO-Surface Rendering (ZBUF-2nd) under Fixed-Frequency Scheme

and the Equation 17, we get

$$k = \left\lceil \sqrt{\frac{108 * 10^6(0.0872 + 1.73 + 1 * (1.73 + 0.0127))}{2.4087}} \right\rceil = 12634$$

then

$$\left\lceil \frac{B}{k} \right\rceil = 8549 \text{ bytes}$$

which falls in the experimental optimal range shown in Figure 13.

5.4.2 *Zbuffer Based ISO-Surface Rendering*

The size of data file B is about 597M, the output image size is 2048x2048, and value of α_3 is 1. Using the parameters from Table 2 and the Equation 21, we get

$$k = \left\lceil \sqrt{\frac{610919236(1.73 + 0.1207 + 1 * (1.73 + 1.0296))}{1.3848}} \right\rceil = 45099$$

then

$$\left\lceil \frac{B}{k} \right\rceil = 13546 \text{ bytes}$$

Again, the value suggested here gives the lowest execution time, as validated in Figure 14.

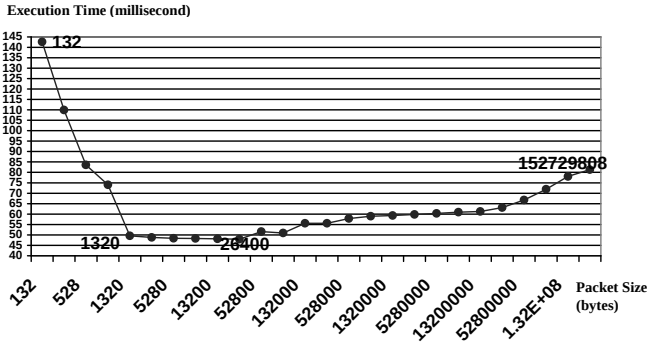

Figure 12: Runtime of ActivePixels Based ISO-Surface Rendering (ACTP) under Fixed-Frequency Scheme

5.4.3 *ActivePixel Based ISO-Surface Rendering*

The size of data file B is about 597M, the output image size is 512x512, and value of α_3 is 1. Using the parameters from Table 2 and the Equation 21, we get

$$k = \left\lceil \sqrt{\frac{610919236(1.73 + 0.1356 + 1 * (1.73 + 0.1937))}{1.4552}} \right\rceil = 39885$$

then

$$\left\lceil \frac{B}{k} \right\rceil = 15317 \text{ bytes}$$

which falls in the experimental optimal range shown in Figure 15.

6. RELATED WORK

Our work is closely related to the work on optimizing packet size in a multi-link communication pipeline by Wang *et al.* [24]. In comparison, our contributions are two fold. First, we have derived expressions for cases where computation is involved and as a result, frequency and/or the volume of communication across different links is different. Second, we have carried out a detailed validation of the model for our target applications and their execution scenario.

Several researchers have developed runtime, scheduling, or language/compiler support for coarse-grained pipelined parallelism. The Stampede project [21] has focused on interactive multimedia applications, which have several common characteristics with the applications we have targeted. The support offered is in the form of cluster-wide threads and shared objects. Yang *et al.* have developed a scheduler for vision applications which are executed in a pipelined fashion within a cluster [26]. They include support for meeting real-time constraints. Our previous work involved developing language support on top of DataCutter [5]. None of these efforts had, however, considered the packet size optimization problem in details.

Pipelined parallelism has also been considered for *communication-exposed* or pipelined FPGA architectures. Ziegler *et al.* have developed compiler analysis for mapping a program to such an architecture [27]. While they consider different granularity of communication between the FPGAs, the evaluation of different choices is done through a design tool, and not analytically. The StreamIt effort at MIT targets similar applications and architecture [23].

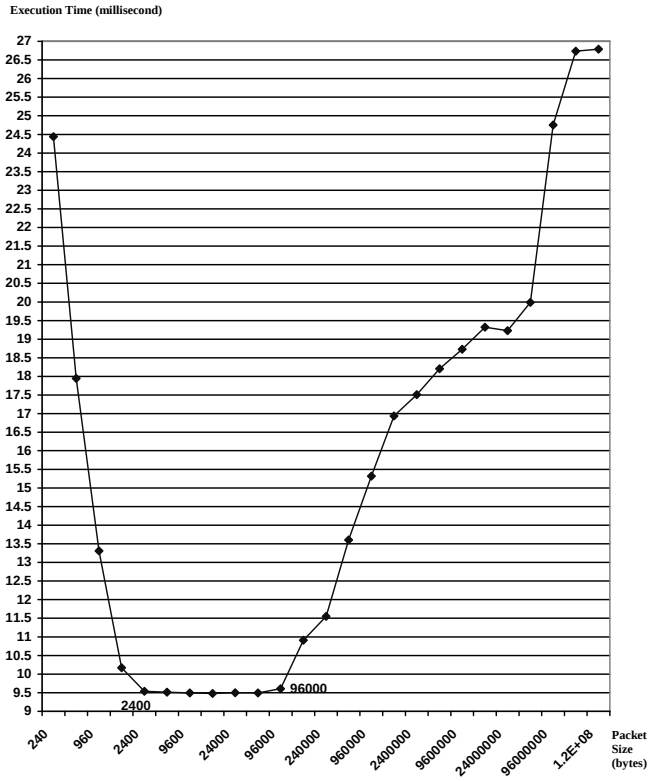

Figure 13: Runtime of KNN-2nd under Fixed-Size Scheme

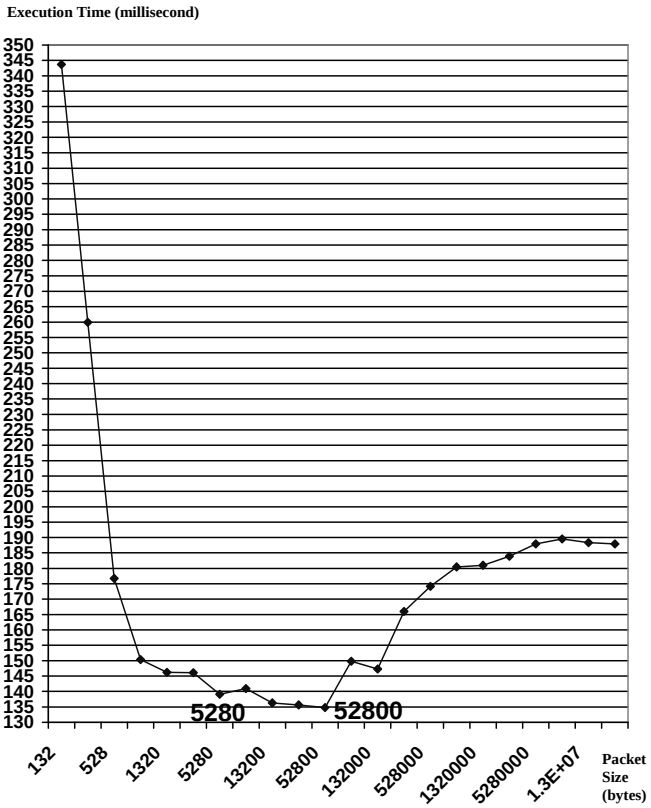

Figure 14: Runtime of Zbuffer Based ISO-Surface Rendering(`ZBUF-1st`) under Fixed-Size Scheme

7. CONCLUSIONS

The work presented here has been in the context of coarse-grained pipelined execution model. This model is very suitable for the execution of data-driven applications in an environment where the data is available on remote data repositories and the results are desirable on a user's desktop. Within this context, we have focused on determining packet size that will achieve the best performance.

We have derived analytical expressions for this purpose. Though our work builds on top of the work of Wang *et al.* on packet size for multi-link communication pipelines, we also addressed two new challenges. First, we had to model computation phases. Second, because of computation, the frequency and/or the volume of communication between the different communication stages can be different. We have considered two models, fixed-frequency and fixed-size communications, and derived expressions for choosing the packet size.

We have carried out detailed evaluation of our models using three applications, executed with different parameters and datasets. Our experiments have shown that the choice of packet size makes a significant difference in the execution time, and the packet sizes suggested by the model result in the lowest or very close to the lowest possible execution time. With a little extra effort, the models can be applied to applications running on the Grid environment.

Acknowledgements: The authors are grateful to the members of DataCutter project, including Tahsin Kurc, Renato Ferreira, Umit Catalyurek, Mike Beynon, Alan Sussman, and Joel Saltz, for providing us with DataCutter implementation, manual codes, and datasets, and for helping us with our experiments.

8. REFERENCES

[1] Asmara Afework, Michael D. Beynon, Fabian Bustamante, Angelo Demarzo, Renato Ferreira, Robert Miller, Mark Silberman, Joel Saltz, Alan Sussman, and Hubert Tsang. Digital dynamic telepathology - the Virtual Microscope. In *Proceedings of the 1998 AMIA Annual Fall Symposium*. American Medical Informatics Association, November 1998.

[2] Michael D. Beynon, Tahsin Kurc, Umit Catalyurek, Chialin Chang, Alan Sussman, and Joel Saltz. Distributed processing of very large datasets with DataCutter. *Parallel Computing*, 27(11):1457–1478, October 2001.

[3] Michael D. Beynon, Tahsin Kurc, Alan Sussman, and Joel Saltz. Optimizing execution of component-based applications using group instances. In *Proceedings of the Conference on Cluster Computing and the Grid (CCGRID)*, pages 56–63. IEEE Computer Society Press, May 2001.

[4] Srinivas Chippada, Clint N. Dawson, Monica L. Martínez, and Mary F. Wheeler. A Godunov-type finite volume method for the system of shallow water equations. *Computer Methods in Applied Mechanics and Engineering (to appear)*, 1997. Also a TICAM Report 96-57, University of Texas, Austin, TX 78712.

[5] Wei Du, Renato Ferreira, and Gagan Agrawal.

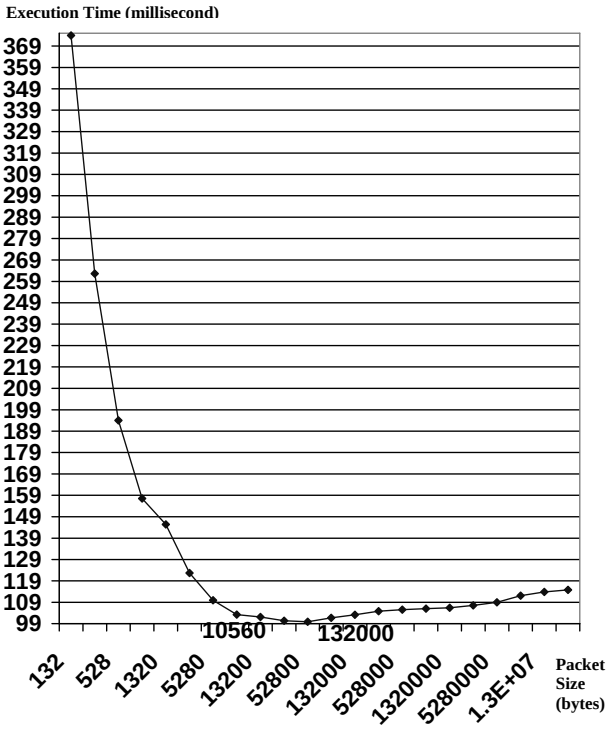

Figure 15: Runtime of ActivePixel Based ISO-Surface Rendering (ACTP) under Fixed-Size Scheme

Compiler Support for Exploiting Coarse-Grained Pipelined Parallelism. In *Proceedings of Supercomputing 2003*, November 2003. to appear.

[6] R. Ferreira, B. Moon, J. Humphries, A. Sussman, J. Saltz, R. Miller, and A. Demarzo. The Virtual Microscope. In *Proceedings of the 1997 AMIA Annual Fall Symposium*, pages 449–453. American Medical Informatics Association, Hanley and Belfus, Inc., October 1997. Also available as University of Maryland Technical Report CS-TR-3777 and UMIACS-TR-97-35.

[7] Ian Foster, Carl Kesselman, and Steven Tuecke. The Anatomy of Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputing Applications*, 2001.

[8] Jim Gray, Adam Bosworth, Andrew Layman, and Hamid Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. In *Proceedings of the 1996 International Conference on Data Engineering*, pages 152–159, February 1996.

[9] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, 2000.

[10] Ruoming Jin and Gagan Agrawal. A middleware for developing parallel data mining implementations. In *Proceedings of the first SIAM conference on Data Mining*, April 2001.

[11] Land Satellite Thematic Mapper (TM). *http://edcwww.cr.usgs.gov/nsdi/html/landsat_tm/landsat_tm.*

[12] W. Lorensen and H. Cline. Marching Cubes: A High Resoltion 3D Surface Reconstruction Algorithm. *Computer Graphics*, 21(4):163–169, 1987.

[13] Richard A. Luettich, Johannes J. Westerink, and Norman W. Scheffner. *ADCIRC*: An advanced three-dimensional circulation model for shelves, coasts, and estuaries. Technical Report 1, Department of the Army, U.S. Army Corps of Engineers, Washington, D.C. 20314-1000, December 1991.

[14] Kwan-Liu Ma and Z.C. Zheng. 3D visualization of unsteady 2D airplane wake vortices. In *Proceedings of Visualization'94*, pages 124–31, Oct 1994.

[15] The Moderate Resolution Imaging Spectrometer. *http://ltpwww.gsfc.nasa.gov/MODIS/MODIS.html.*

[16] NASA Goddard Distributed Active Archive Center (DAAC). Advanced Very High Resolution Radiometer Global Area Coverage (AVHRR GAC) data. *http://daac.gsfc.nasa.gov/CAMPAIGN_DOCS/LAND_BIO/origins.html.*

[17] Grid Physics Network. GriPhyN. http://www.griphyn.org.

[18] Ron Oldfield. Summary of existing and developing data grids. White paper, Remote Data Access Group, Global Grid Forum, available from http://www.sdsc.edu/GridForum/RemoteData/Papers/papers.html.

[19] G. Patnaik, K. Kailasnath, and E.S. Oran. Effect of gravity on flame instabilities in premixed gases. *AIAA Journal*, 29(12):2141–8, Dec 1991.

[20] Teragrid project partners. The TeraGrid: A Primer, September 2002. Available at www.teragrid.org.

[21] U. Ramachandran, R. S. Nikhil, N. Harel, J. M. Rehg, and K. Knobe. Space-Time Memory: A Parallel Programming Abstraction for Interactive Multimedia Applications. In *Proceedings of the Conference on Principles and Practices of Parallel Programming (PPoPP)*, pages 183–192. ACM Press, May 1999.

[22] T. Tanaka. Configurations of the solar wind flow and magnetic field around the planets with no magnetic field: calculation by a new MHD. *Jounal of Geophysical Research*, 98(A10):17251–62, Oct 1993.

[23] William Thies, Michal Karczmarek, and Saman Amarasinghe. StreamIt: A Language for Streaming Applications. In *Proceedings of Conference on Compiler Construction (CC)*, April 2002.

[24] R. Y. Wang, A. Krishnamurthy, R. P. Martin, T. E. Abderson, and D. E. Culler. Modeling Communication Pipeline Latency. In *Proceedings of the ACM SIGMETRICS Conference*. ACM Press, June 1998.

[25] R. Wolski. Forecasting Network Performance to Support Dynamis Scheduling Using the Network Weather Service. In *Proceedings of the Conference on High Performance Distributed Computing (HPDC)*, 1997.

[26] M. T. Yang, R. Kasturi, and A. Sivasubramaniam. An Automatic Scheduler for Real-Time Vision Applications. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, 2001.

[27] Heidi Ziegler, Mary W. Hall, and Pedro Diniz. Compiler Generated Communication for Pipelined FPGA Applications. In *Proceedings of the Design Automation Conference (DAC)*, June 2003.