(submitted to Workshop on Algorithms and Computational Molecular Biology)

A New Programming Paradigm for Achieving High Performance Computation in Network Computing Platform

Cho-Chin Lin
Department of Electronic Engineering
National I-Lan Institute of Technology
I-Lan 260, Taiwan, ROC
Email:cclin@ilantech.edu.tw
Tel:(03)9357400 ext. 644   Fax:(03)9369507

Keywords: Parallel and Distributed Computing, Network Computing, Algorithm

Abstract

Current parallel or distributed systems employs a large volumn of complete nodes to meet the timing requirement in solving applications of large scale. In general, the nodes are interconnected by a network of various topologies. Examples are clusters of workstations/PC 's and the computional grids. In such a platform, an appropriate strategy is needed to squeeze the computing power from the systems.

In general, high performance computing can be achieved by effective resource utilization. In this paper, a novel programming paradigm is proposed for achieving effective resource utilization in network computing platform. We use two parameters to model the platform. Based on the model, a function is proposed to capture the programming feature on the platform. The function is used to group data items into messages and to schedule the messages to their destination site. By choosing a grouping function appropriately, we can maximize the utilization of the computational resources in the system. Finally, matrix multiplication and LU decomposition are used as examples to illustrate the usefulness of our programming paradigm. In this paper, we has shown that computational resources can be effectively utilized by employing our programming paradigm.

1 Introduction

Since computers were developed,they have been employed to assist human being in handling daily events. Many of the applications concerning the issues of human welfare and the science leading to a better living environment need a large volume of computations. For example, the goal of improving atmospheric modeling resolved to a 5-km scale and providing timing result is believed to require 20 TFLOPS of performance [10]. We know that most powerful sequential computers of today can not meet the computational requirement needed to implement the approach. Thus, it is obvious that a serious attack on the application requires high performance computing platform.

The advance in the microprocessor and memory technologies impacts the speed of a computer. The performance of a microprocessor is advancing at a rate of 50 to 100% per year [8]. Today, the state of the art microprocessors can have computation speed up to hundreds of MFLOPS [4]. In addition to that, memory capacity is increasing at a rate comparable to the increase in capacity of DRAM chips: quadrupling in size every three years [7]. Current personal computers or workstations use hundreds of Mbytes. It seems that substantial progress has been achieved in sequential computer technology. However, the performance of the computer still can not suit the applications of increasing complexity. Thus, scalable architectures which employ parallel or distributed processing technology have been proposed to meet the computational requirements.

Scalable architectures have the opportunity to challenge the applications of large scale. One of the architectural solutions for achieving scalability is a network computing system operating in MIMD or SPMD modes. Examples are clusters of workstations/PC's or computational grid [5]. They are formed by combining essentially complete processing nodes (processors, memory modules and I/O capability provided by such a network computing platform potentially provides the basic requirement of the large-scaled applications. However, the limition on the nework capacity usually degenerate the performance of the systems. Thus, it is important to give a novel approach to solve this problem.

In this paper, a novel programming paradigm is proposed for performing high performance computation in network computing environment. Our paradigm achieves the goal by maximizing the utilization of the computational resources. In Section 2, we give the background of the potential problems which can incur in network computing environment. A model and notations used in this paper will be defined in Section 3. In Section 4, our programming paradigm for acceleratimg the execution of a task is proposed. In Section 5, matrix multiplication and

LU-decomposition are used as examples to illustrate the usefulness of the approach. Finally, conclusion is made in Section 6.

2 Background

In a network computing system, a network is necessary for a node to receive data from another node. To accelerate the execution of a task, an adequate strategy is needed for partitioning a task to several subtasks. Those subtasks are assigned and executed curently on the nodes of the system. In the system, the operation mode of the nodes can be classified into two categories: a system of unified nodes and a system of diverse nodes. In a system of unified nodes, all the nodes play the same role to finish a task. It is usually referred to as a parallel computer system. In a system of diverse nodes, each of the nodes emphasizes on various functions of a task. For example, some nodes are responsible for database management or act as data providers, and others perform application-oriented computations. It is usually referred as a distributed system. No matter what categories it belongs to, efficient data exchange among nodes is fundamental for a system to achieve high performance computing.

In general, some communication patterns may have serious impact on the performance of a network computing system. It is due to the limited capacity of the network inherited from its hardware and software. A communication pattern can be the results caused by, for example, insufficient buffer size, contention at network links, or contention at network adaptor. Several researches [1, 2, 9, 11] have been conducted to solve the problem. Many of those try to reorganize a communication pattern to suit the nature of an existing network in order to deliver messages efficiently. The approach is suitable for a dedicated systems, in which all the nodes are execution the same task and stall to reorganize the communication pattern together. However, to reorganize a communication pattern may involve the nodes which do not intend to send any message. Thus, this approach may lead to a system-wide overhead. The overhead further enhances the degree of degeneration. Especially, this adverse effect should be avoided for a sharable system running several applications concurrently. In this paper, an programming paradidm is proposed to maximize the utilization of the computing nodes without incurring system-wide overhead. That is, any message will be sent directly to its destination node without employing any intermediate nodes.

3 Model and Notations

In a network computing system, a node may either serves as a data provider or a computing engine. The computing engine cooperates with the data provider to provide services for users. Any user can access services through public channels and share the computational resources with other users. In the following sections, we refers a group of data providers as a data site and a group of computing engines as a computing site. In this section, an abstract model for the network computing system is proposed. In the model, two parameters are used to capture the computational characteristics in the environment. Based on the model, a novel programming paradigm for developing high performance computing is given. By employing the paradigm, an effective algorithm can be designed for network computing platform. Before we proceed to propose the model, several general notations will be defined in this section. Nevertheless, other notations related to specified applications will be given wherever it is appropriate in the following sections.

The following notations are used to denote simple mathematical operations. The purpose of the notations is to simplify mathematical expression in the following sections for clarity.

Definition 1 *q_n^i is defined to be equal to $\lfloor i/n \rfloor$ and r_n^i is defined to be equal to $(i \bmod n)$.*

The second notation is a *grouping* function denoted as ρ . By choosing an appropriate grouping function, we can associate each of the data items to a set. Thus, it also implies that the data items are partitioned into several groups according to the relation specified by the function.

Definition 2 *Let $\mathcal{E}=\{e_0, e_1, \ldots, e_i, \ldots, e_{|\mathcal{E}|-1}\}$ is an set of data items and $\mathcal{M} = \{m_0, m_1, m_2, \ldots, m_j, \ldots, m_{|\mathcal{M}|-1}\}$ is an ordered set of groups. Grouping function ρ is a binary relation from $\mathcal{E}$ to $\mathcal{M}$ such that $\mathcal{E} \times \mathcal{M}$ is $\{(e_i, m_{\rho(i)}) | \text{where } i \text{ is an integer and } 0 \leq i < |\mathcal{E}|\}$.*

In Section 5, the grouping function is used to specify the sequence of data items sent by a data site. That is, the data site sends data items $e_0, e_1, e_2, \ldots, e_{|\mathcal{E}|-1}$ using messages $m_0, m_1, \ldots, m_{|\mathcal{M}|-1}$ in order. We also assume the computing site receives the messages in the same order.

In a network computing environment, when a service request is called from a remote user, a task is created to handle the corresponding request. In general, most of the resources in a network computing environment are sharable. Examples are CPU cycles, memory space, I/O channels, interconnection networks, and service access channels. Those resources should be managed fairly in order to meet the quality of specified services. The resource manager of

a computing system can be parts of the operating system kernel or the middleware built on the top of the kernel such as DQS[6]. Many factors or events can change the current state of resource allocation to a task. For examples, the current load of the system, the importance of the recently entering task, and the po;icy of employing network bandwidth are crucial to resource allocation strategy. Those factors or events are complexity. However, from the view point of algorithm designers, it is impractical to use too many parameters to capture the characteristics of a system. In this paper, we abstract the phenomenon concerned by algorithm designers into two parameters.

The first parameter of our model is used to capture the quantity of computational power allocable for a task at a computing site. Note that a computing site may consists of one or several computing engines which cooperate to solve an application problem. The parameter is defined as follows:

- g: computational gain (measured in number of CPU cycles). It is defined as the quantity of CPU cycles allocable to a task for perform operations at a computing site. It is measured at the interval between two consecutive messages arriving at the computing site. In general, the computational gain can be varied between two messages. Let the messages be $m_0, m_1, \ldots, m_{|\mathcal{M}|-1}$. Thus, $g(i)$ denotes the computational gain at the interval between message i and message $i+1$, where $0 \leq i < |\mathcal{M}| - 1$. If $i = |\mathcal{M}| - 1$, then $g(i)$ is defined to be the the total CPU cycles needed to complete the remaining computations.

Although the parameter $g(i)$ is measured in number of cycles, it intends to capture the effects of interaction among the available resources. Those resources include CPU cucles, memeory space, I/O utility, etc. Note that $g(i)$ is the upper bound on the CPU cycles at a computing site for a task to employ. A task may employ no more than $g(i)$ CPU cycles for performing computations at the interval between message m_i and message m_{i+1}. The computation gain $g(i)$ intends to normalize the available computational power at a computing site using the interval of two messages. That is, the quantity of $g(i)$ depends not only on the usable local resouces but also on the communication channels. For examples, the communication software overhead, network latency, and network bandwidth can also affect the quantity of $g(i)$. Thus, it is easy to see that two tasks acquiring the same amount of CPU cycles at a fixed time period may not have the same quantity of $g(i)$. Based on the parameter $g(i)$, we define *accumulative* computational gain as follows:

Definition 3 *Let $G(i)$ is accumulative computation gain. Then, $G(i)$ is defined to be equal to $\sum_{k=0}^{i} g(k)$.*

The notation $G(i)$ is the amount of total computational gain allocable to a task at a computing site before the $i+1$_th message arrives. Note that A task may employ no more than $G(i)$ CPU cycles to complete a task.

In a network computing environment, the data items sent by a message are ready for performing computations only after the message is completely received by the computing site. Ready for performing computations implies that additional operations can be performed at the computing site. We denote those computations as triggered computations. In the case of sending a long message, the data provider will the computational activity at the computing site for long time. It may lead to that a task underutilizes the computational gain allocated to it. To maximize the utilization of the computational resources at a computing site, overlapping communication with computation is a possible strategy. The strategy can be achieved by partitioning data items into several groups. Then, each of the groups is sent by a sequence of messages. As soon as the computing site receives a message, computations may be triggered. Although the amount of total computations is fixed for a task, however, the newly triggered computations may not be the same for each of incoming messages. The reason is that data dependency in computations may be within a message or across messages. It will be explained in details in Section 4. Thus, shipping data items to a computing site need to be considered carefully. To capture the phenomenon described above, the second parameter will be proposed to express the amount of additional computations triggered at a computing site for each incoming message.

- f: computational fillet (measured in number of CPU cycles). It is defined as the amount of additional computations triggered at a computing site when a message is received by the site. In general, the computation fillets may be varied for a sequence of messages. Thus,$f(i)$, the i_th fillet, denotes the additional computations triggered at a computing site after message m_i has received at the site.

Since the amount of computations triggered for an incoming message depends on the sequence of the messages, thus, two messages of the same size may not have the same value of f. In addition, the amount of computations triggered by an incoming message should be no less than zero, thus, we have, $f(i) \geq 0, \text{for } i \geq 0$. Based on the parameter computation gain, we define *accumulative* computation fillet (ACF) as follows:

Definition 4 *Let $F(i)$ is accumulative computation fillet. Then, $F(i)$ is defined to be equal to* $\sum_{k=0}^{i} f(k)$

The notation $F(i)$ is the amount of total computational fillets of a task accumulated at a com-

puting site before message m_{i+1} arrives. Note that a task may not have enough computational resources to complete $F(i)$ computations at a computing site even though the computing site has received the message m_{i+1}.

4 Programming paradigm

In a computing system, a task is created to process the request from users. For a network computing system to execute a task, a computing site may need to access data items across a network. In general, the data items are sent by a sequence of messages in order to overlap communucation with computation. The tasks use the computational gain allocated by the computing site to perform operations assigned by the computational fillets. The quantity of a computational fillet assigned by a message depends on twhich of he data items encapsulated in the message and the order of the message in the sending sequence. We will illustrate this using a simple example. Let $\mathcal{E} = \{e_0, e_1, e_2, e_3, e_4, e_5\}$. The data items specified by $\mathcal{E}$ is stored at a data site p_0. The data items are sent by messages m_0 and m_1 for performing computations at a computing site p_1. The computations performed at p_1 are $e_0 + e_1$, $e_0 + e_2$, $e_3 + e_4$, $e_3 + e_5$, $e_4 + e_5$. Considering that the grouping fuctions ρ_0 and ρ_1, each of which partitions data items into two sets. The first grouping function ρ_0 partitions the data items into $s_{00} = \{e_1, e_2, e_3\}$ and $s_{01} = \{e_0, e_4, e_5\}$. The second grouping function ρ_1 partitions the data items into $s_{10} = \{e_0, e_1, e_2\}$ and $s_{11} = \{e_3, e_4, e_5\}$. Two scenarios can happen as follows. The first scenario is that the data site sends messages m_0 and m_1 using s_{00} and s_{01} respectively to p_1, and site p_1 receives the messages in order. By simple analysis, we have $f(0) = 0$ and $f(1) = 5$. The second scenario is that the data site sends messages m_0 and m_2 using s_{10} and s_{11} respectively to p_1 and site p_1 receives the messages in order. By simple analysis, we have $f(0) = 2$ and $f(1) = 3$. Then, it is easy to verify that if the $g(0) = 3$, then $g(1) = 5$ for first scenario. However, for the same vakue of $g(0)$, we have $g(1) = 3$ for the second scenario. It imples that the task can finishes earlier if the second strategy is employed. However, it is not the optimal strategy. We can have better solution if the data site sends messages m_0 and m_1 using s_{11} and s_{10} . In this case, if we have the same value of $g(0)$, then $g(1) = 2$. It implies that the task can finishes even earlier than the previous two strategies do.

In the network computing environment, a data site sends a sequence of messages to a computing site. The computing site provides computational gain to perform the triggered computations. From the above example, we know that If we carefully schedule data items to the computing site, then computations can be triggered earlier. Otherwise, a large volumn of computations will be accumulated to the end of communication. Thus, the execution time of a

task will be prolonged. In our programming paradigm, data items are partitioned into several sets by choosing an appropriate grouping function ρ. The grouping function is chosen based on two strategies:

- Group the data items into messages of size n such that there is no or little data dependency among messages.
- Maximize the utilization of the computional gains by sending the messages according to the volumn of computational fillets. That is, the message with larger computational fillet volumn is sent earlier than those with smaller one.

The first strategy is to capture the data access locality for performing a specified computation step. The second strategy makes effective utilization of computation resources for the task performed at the computing site. Our goal is to minimize the $g(|\mathcal{M}| - 1)$. Based on the above statement, we know that function ρ not only partitions data items but also assigns the delivery order for each of the sending messages.

5 Applications

Matrix multiplication (MM) and LU decomposition (LUD) are basic but important in scientific computations. In this section, MM and LUD are used as examples to illustrate the usefulness of our paradigm. For each of the computations, algorithms are proposed to generate various patterns of computational fillets. The various patterns lead to different utilization rates of the ACG's.

5.1 Matrix multiplication

In this section, matrix multiplication is used to illustrate our paradigm for developing algorithm in a network computing environment. The scenario of the computation is described as follows. A network computing system consists of a data site p_0 and a computing site p_1. Initially, matrices A and B of size $n \times n$ are stored at data site p_0. After the system has receiveed a request of matrix multiplication $A \times B$ from a remote user, the computing site p_1 starts to receive data items from p_0 and performs matrix multiplication. The multiplications are performed at site p_1 as soon as the computations are triggered at the site. The site p_1 uses matrix C for storing temporary and finial result.

```
begin
  for 0 ≤ i < n
      sends A(i,*) to p_1 using the i_th message;
  for 0 ≤ j < n
      sends B(*,j) to p_1 using the (n+j)_th message;
end;
```

(a)

```
begin
  for 0 ≤ i < n
      receives A(i,*) from p_0 in the i_th message;
  for 0 ≤ j < n
      receives B(*,j) from p_0 in the (n+j)_th message;
      //which triggers the computations for deriving C(*,j)
end;
```

(b)

Figure 1: (a) Algorithm MM_α for p_0; (b) Algorithm MM_α for p_1

In this section, three strategies MM_α, MM_β, and MM_γ for designing MM algorithm are proposed and analyzed to illustrate our programming paradigm. Let $\mathcal{E} = \{e_i\}$ contains the data items of matrices $A = [a_{ij}]$ and $B = [b_{ij}]$, where e_{in+j} is a_{ij} and e_{n^2+in+j} is b_{ij} for $0 \leq i, j < n^2$. Each of the algorithms employs different grouping functions for partitioning data items into different sets. The goruping functions group the data to $2n$ sets $m_0, m_1, \ldots, m_{2n-1}$. Each of the sets is of size n. The site p_0 sends set m_i using the i_th message, for $0 \leq i \leq 2n-1$. In the followings, several notations used in matrix multiplication will be defined first.

Definition 5 *$A(i,*)$ denotes all the elements in the i_th row of matrix A and $A(*,j)$ denotes all the elements in the j_th column of matrix A. $C^k(i,j)$ denotes the value of $\sum_{l=0}^{k} C(i,l) \times C(l,j)$.*

In algorithm MM_α, site p_0 sends the elements of matrix A row by row, then sends the elements of matrix B column by column. Thus, the grouping function for MM_α is as follows:

$$\rho(i) = \begin{cases} q_n^i & \text{if } 0 \leq i < n^2 \\ n + r_n^i & \text{if } n^2 \leq i < 2n^2 \end{cases}$$

The operations performed in p_0 and p_1 are shown in Figure 1 In algorithm MM_α, p_1 is able to perform matrix multiplication only after p_0 begins to send the elements of matrix B. Based on the algorithm, we can calculate the computational fillet $f(i)$ amount at the computing site p_1 as soon as the i_th message received by the site p_1. Note that we count the operation of one addition plus one multiplication as one computation step which consumes one CPU cycle. The function $f(i)$ is shown as follow:

$$f(i) = \begin{cases} 0, & \text{for } 0 \leq i < n \\ n^2, & \text{for } n \leq i < 2n \end{cases}$$

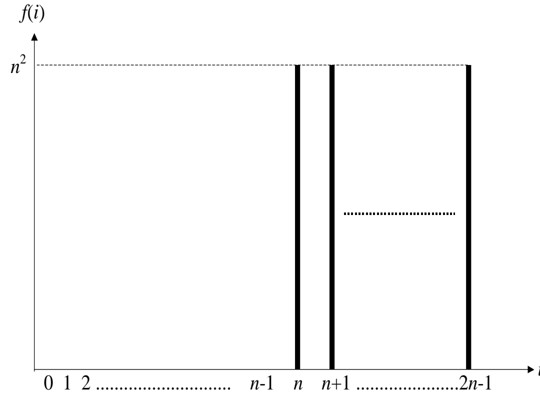


Figure 2: additional operations triggered at p_1 as the i_th message arrives for MM_α

Figure 2 also illustrates the function $f(i)$. From the figure, we observe that no computation can start for the first n messages arrive at site p_1. However, n^2 computations can be performed when each of the following n message arrives at the site p_1.

In algorithm MM_β, p_0 sends the elements of matrix A row by row alternating with the elements of matrix B column by column to computing site p_1. Thus, the grouping function for MM_β is as follows:

$$\rho(i) = \begin{cases} 2q_n^i & \text{if } 0 \leq i < n^2 \\ 2r_n^i + 1 & \text{if } n^2 \leq i < 2n^2 \end{cases}$$

The operations performed in p_0 and p_1 are shown in Figure 3. In algorithm MM_β, p_1 can perform matrix multiplications . p_0 starts to send the elements of matrix B. Thus, by employing MM_β, the computations can be triggered earlier if compared with algorithm MM_α. Based on the grouping function for algorithm MM_β , we can calculate the computational fillet $f(i)$ at site p_1 as soon as the i_th message received by the site p_1. The function $f(i)$ is shown as follow:

$$f(i) = \lfloor (i+1)/2 \rfloor \times n \qquad for\ 0 \leq i < 2n$$

Figure 4 also illustrates the function $f(i)$. From the figure, we can see that operations to be triggered is an increasing function of i. Assume $g(i)$ is an decreasing function of i. Then, we can derive that the value of $G((|\mathcal{M}|-1)$ for algorithm MM_α is no less than that for algorithm MM_β even though they have the same value of $F(|\mathcal{M}|-1)$. Compared with MM_α, it does not delay the operations for the triggered computations to the end of the communication.

In algorithm MM_γ, p_0 sends the elements of matrix A column by column alternating with the elements of matrix B row by row . Thus, the grouping function for MM_γ is as follows:

$$\rho(i) = \begin{cases} 2r_n^i & \text{if } 0 \leq i < n^2 \\ 2q_n^i + 1 & \text{if } n^2 \leq i < 2n^2 \end{cases}$$

```
begin
  for 0 ≤ i < n
     { sends A(i,*) to p_1 using the i_th message;
       sends B(*,i) to p_1 using the (2i+1)_th message;}
end;
```

(a)

```
begin
  receives A(0,*) from p_0 in the 0_th message;
  for 1 ≤ i < n
     { receives B(*,i-1) from p_0 in the (2i-1)_th message;
       //which triggers the computations for deriving C(j,i-1) for 0 ≤ j < i;
       receives A(i,*) from p_0 in the 2i_th message;
       //which triggers the computations for deriving C(i,j) for 0 ≤ j < i;}
  receives B(*,n-1) from p_0 in the (2n-1)_th message;
  //which triggers the computation C(*,n-1);
end;
```

(b)

Figure 3: (a) Algorithm MM_β for p_0; (b) Algorithm MM_β for p_1

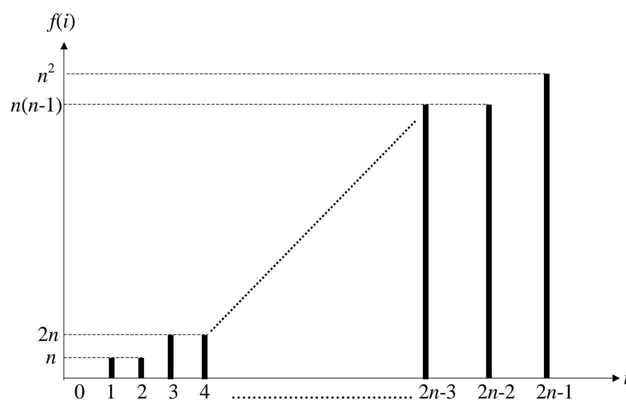
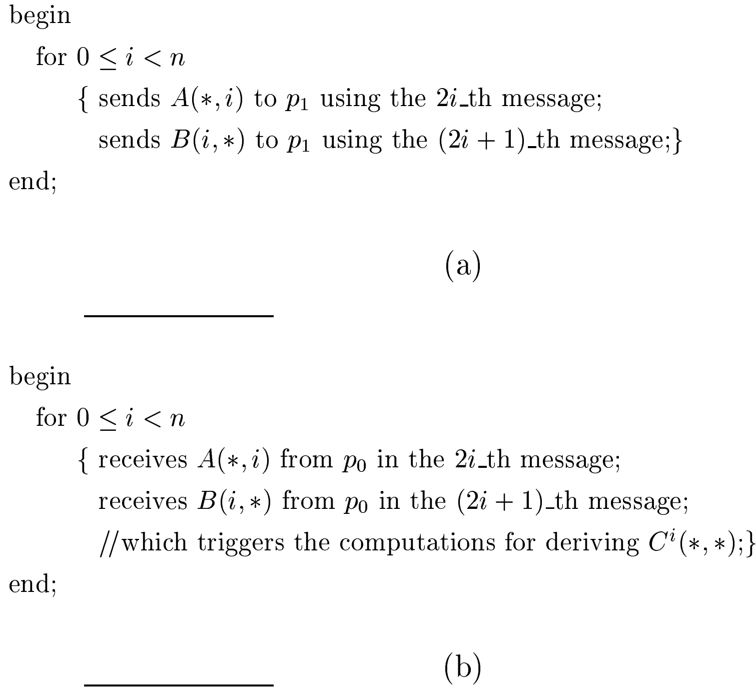


Figure 4: additional operations triggered at p_1 as the i_th message arrives for MM_β



```
begin
  for 0 ≤ i < n
     { sends A(*,i) to p1 using the 2i_th message;
       sends B(i,*) to p1 using the (2i + 1)_th message;}
end;
```

(a)

```
begin
  for 0 ≤ i < n
     { receives A(*,i) from p0 in the 2i_th message;
       receives B(i,*) from p0 in the (2i + 1)_th message;
       //which triggers the computations for deriving C^i(*,*);}
end;
```

(b)

Figure 5: (a) Algorithm MM_γ for p_0; (b) Algorithm MM_γ for p_1

It is shown in Figure 5. In algorithm MM_γ, p_1 can perform matrix multiplications as soon as p_0 begins to send the elements of matrix B. Since the i_th row of matrix B is sent followed by i_th column of matrix A, the additional computations can be triggered by messages with odd index i. Based on the algorithm, we can calculate the quantities of computational fillets. The function $f(i)$ is shown as follow:

$$f(i) = \begin{cases} 0, & \text{if } i \text{ is even and } 0 \le i < 2n \\ n^2, & \text{if } i \text{ is odd and } 0 \le i < 2n \end{cases}$$

Figure 6 also illustrates the function $f(i)$. From the figure, we can see that amount of computations to be triggered is alternating with 0 and n^2. Assume $g(i)$ is an decreasing function of i. Then, we can derive that the value of $G((|\mathcal{M}| - 1)$ for algorithm MM_γ is no less than that of algorithms MM_α and MM_β , even though they have the same value of $F(|\mathcal{M}| - 1)$. Compared with MM_α and MM_β, it does not delay the operations for the triggered computations to the end of the communication.

The comparison among MM_α, MM_β and MM_γ for matrices of size 3×3 is shown in Figure 7. In the figure, we can observe that if we set $g(i) = 4.5$ for $0 \le i < 5$ then $g(5)$s are equal to 18, 12, and 9 for MM_α, MM_β and MM_γ, respectively. Thus, if the system provides 4.5 CPU cycles/second for execution the task, then MM_α, MM_β and MM_γ can finish at 9, 7.66, and 7 seconds, respectively. Let the *ACF*s for MM_α, MM_β and MM_γ be denoted as $F_\alpha(i)$, $F_\beta(i)$ and $F_\gamma(i)$ respectively and the *ACG*s for MM_α, MM_β and MM_γ be denoted as $G_\alpha(i)$, $G_\beta(i)$ and $G_\gamma(i)$ respectively. Then, we also have,

$$F_\alpha(i) \le F_\beta(i) \le F_\gamma(i) \;\text{ for } 0 \le i < |\mathcal{M}| \quad \text{and} \quad G_\alpha(|\mathcal{M}| - 1) \ge G_\beta(|\mathcal{M}| - 1) \ge G_\gamma(|\mathcal{M}| - 1)$$

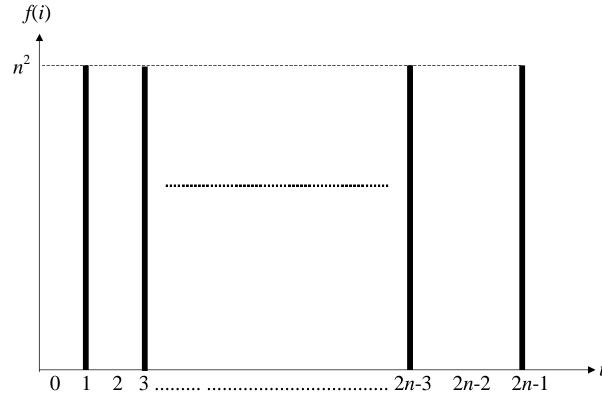
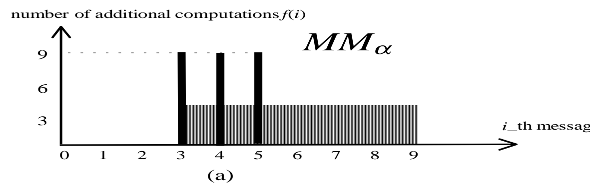
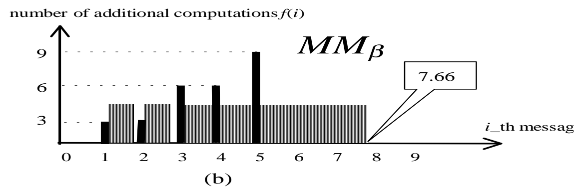


Figure 6: additional operations triggered at p_1 as the i_th message arrives for MM_γ





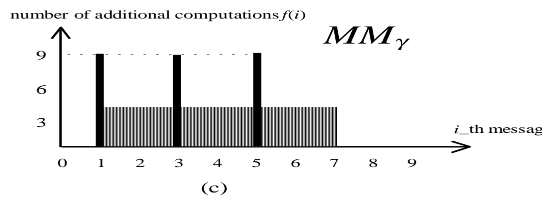


Figure 7: Comparison among MM_α, MM_β and MM_γ

```
begin
  for i = 1 to n
      {u_ii = a_ii;
       for j = i + 1 to n
           {l_ji = a_ji / u_ii;
            u_ij = a_ji;
            for j = i + 1 to n
                {for k = i + 1 to n
                     a_jk = a_jk - l_ji u_ik;}}}
  return L and U
end;
```

Figure 8: Algorithm for LU Decomposition

5.2 LU-decomposition

The second example we will study is LU decomposition (LUD) which is used in solving systems of linear equations. First, we show the sequential algorithm which decomposes matrix A into upper triangular matrix U and lower triangular matrix L. The elements of A, L, U in row i and column j are denoted as a_{ij}, l_{ij} ,u_{ij} respectively. For the purpose of clarity, a sequential LU-decomposition algorithm for a system of single node is shown in Figure 8. Interested readers should refer to [3] for more details on the topic of solving systems of linear equations. Let $\mathcal{E} = \{e_i\}$ contains the data items of matrices $A = [a_{ij}]$, where e_{in+j} is a_{ij} for $0 \leq i, j < n^2$. The scenario of the LUD of our algorithms is as follows: the elements of matrix A of size $n \times n$ stored at p_0 are sent to p_1 for calculating the matrices L and U.

The first algorithm we propose is Algorithm LU_α. It operates as follows: a data site p_0 sends n messages of size n to a computing site p_1 by row major. Thus, the grouping function for ρ is as follows.

$$\rho(i) = q_n^i$$

When the computing site p_1 receives the messages, computations are triggered at site p_1. The elements of L and U are stored at D when the computations proceeds. The operations performed at p_0 and p_1 are shown in Figure 9. In algorithm LU_α, site p_1 starts to perform computations on a row after p_1 has received the second row. The function of computational fillet is as follow:

$$f(i) = (2n - i + 1)i/2$$

In the function, we count multiplication or division operations as one computation step. Figure 10 also illustrates the function $f(i)$. From above equation, we can see that the amount of computations to be triggered increases as the value of i increases.

The second algorithm we propose is Algorithm LU_β. It operates as follows: source site

```
begin
  for 0 ≤ i < n
      sends A(i,*) to p_1 using the i_th message;
```

(a)

```
begin
  for 0 ≤ i < n
      receives A(i,*) from p_0 in the i_th message;
      //which triggers the computations for deriving D(i;*),
end;
```

(b)

Figure 9: (a) Algorithm LU_α for p_0; (b) Algorithm LU_α for p_1

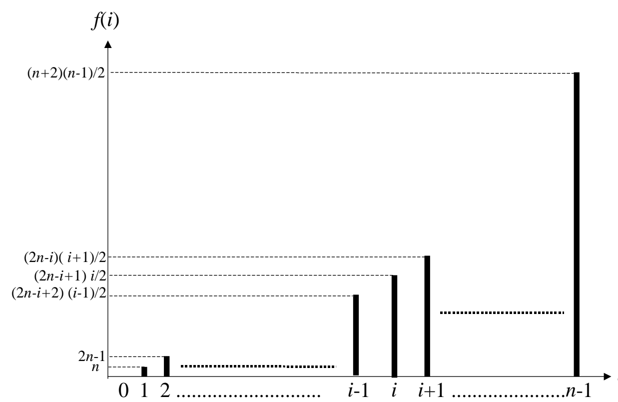


Figure 10: additional operations triggered at p_1 as the i_th message arrives for LU_α

```
begin
  for 0 ≤ j < n
      sends A(*, j) to p_1 using the j_th message;
```

(a)

```
begin
  for 0 ≤ j < n
      receives A(*, j) from p_0 in the i_th message;
      //which triggers the computations for deriving D(*, j),
end;
```

(b)

Figure 11: (a) Algorithm LU_β for p_0; (b) Algorithm LU_β for p_1

p_0 sends n messages of size n to computing site p_1 by column major. When site p_1 receives the messages, computations are triggered in site p_1. Thus, the grouping function for LU_γ is as follows.

$$\rho(i) = r_n^i$$

The elements of L and U are stored at D after the computations proceeds. The operations performed in sites p_0 and p_1 are shown in Figure 11. In algorithm LU_β, site p_1 can start to perform computations on a column after p_1 receives the first column. The function of computation density is as follow:

$$f(i) = (2n - i - 2)(i + 1)/2$$

Figure 12 illustrates the function $f(i)$. From above equation, we can see that the amount of computations to be triggered increases as the value of i increases.

The third algorithm is LU_γ. In algorithm LU_γ, data site p_0 sends the elements of matrix A to computing site p_1, by alternating rows with columns. The first message sends the column of A Thus, the grouping function for LU_γ is given as as follows.

$$\rho(i) = \begin{cases} r_n^{(\sum_{k=1}^{r_n^i}(1+2(n-k)))+1} & \text{if } q_n^i = r_n^i \\ r_n^{(\sum_{k=1}^{r_n^i}(1+2(n-k)))+q_n^i-r_n^i+1} & \text{if } q_n^i > r_n^i \\ r_n^{(\sum_{k=1}^{q_n^i}(1+2(n-k)))+n-2q_n^i+r_n^i} & \text{if } q_n^i < r_n^i \end{cases}$$

The operations performed in sites p_0 and p_1 are shown in Figure 13. In the figure, m_i is defined

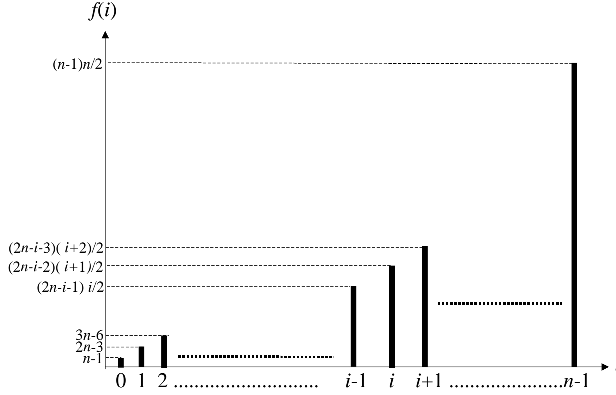


Figure 12: additional operations triggered at p_1 as the i_th message arrives for LU_β

begin
  for $0 \leq i < n$
    sends A to p_1 using the i_th message m_i;
end;

(a)

begin
  for $0 \leq i < n$
    {receives the i_th message m_i from p_0}
    //which triggers the computations for deriving $A(i,k)$,
    //where $j \leq k < n$ for $0 \leq j < i$
end;

(b)

Figure 13: (a) Algorithm LU_γ for p_0; (b) Algorithm LU_γ for p_1

by the grouping function. In algorithm LU_γ, site p_1 can start to perform computations after site p_1 has received rows or columns.

For analysis purpose, we repartition the data items into $(3n-2)$ sets of various sizes. The sets are $S_0, S_l, S_2, \ldots, S_{3n-3}$, where $S_{3i} = \{a_{ii}\}$, $S_{3i+1} = \{a_{i+1,i}, a_{i+2,i}, a_{i+3,i}, \ldots, a_{n-1,i}\}$ and $S_{3i+2} = \{a_{i,i+1}, a_{i,i+2}, a_{i,i+3}, \ldots, a_{i,n-1}\}$. Partitioning the elements of the matrix $A = [a_{ij}]$ of size 9×9 is shown in Figure 14. The number in each of the entries is the index of a set to which the element a_{ij} . For example, $a_{2,1}$ belongs to set S_4.

0	2	2	2	2	2	2	2	2
1	3	5	5	5	5	5	5	5
1	4	6	8	8	8	8	8	8
1	4	7	9	11	11	11	11	11
1	4	7	10	12	14	14	14	14
1	4	7	10	13	15	17	17	17
1	4	7	10	13	16	18	20	20
1	4	7	10	13	16	19	21	23
1	4	7	10	13	16	19	22	24

Figure 14: repartition the data items of matrix A into 25 sets of various sizes

Figure 15: additional operations triggered at p_1 as the i_th message arrives for LU_γ

The function of computational fillets based on the new sets $S_0, S_l, S_2, \ldots, S_{3n-3}$ is defined as follows:

$$f(i) = \begin{cases} 0 & \text{if } r_3^i = 0 \\ (n - q_3^i - 1) & \text{if } r_3^i = 1 \\ (n - q_3^i - 1)^2 & \text{if } r_3^i = 2 \end{cases}$$

Figure 15 also illustrates the function $f(i)$.

we can see that the amount of computations triggered tends to decreases as the value of i increases. Assume $g(i)$ is an decreasing function of i. Then, we can derive that the value of $G((|\mathcal{M}| - 1)$ for algorithm LU_γ is no less than that of algorithms LU_α and LU_β , even though

*i*_th message	0	1	2	3	4	5	6	7	8
# of additional computations	0	9	17	24	30	35	39	42	44
# of accumulated computations	0	9	26	50	80	115	154	196	240

(a)

*i*_th message	0	1	2	3	4	5	6	7	8
# of additional computati ns	8	15	21	26	30	33	35	36	36
# of accumulated computations	8	23	44	70	100	133	168	204	240

(b)

*i*_th message	0	1	2	3	4	5	6	7	8
# of additional computations	8	64	21	38	39	20	22	20	8
# of accumulated computations	8	72	93	131	170	190	212	232	240

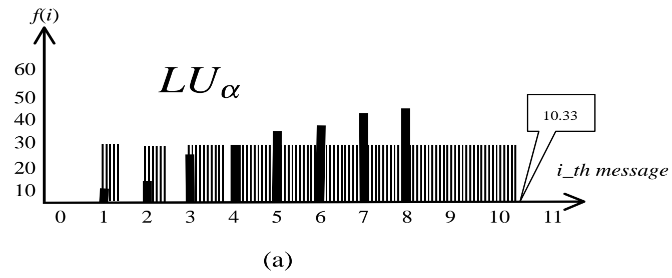
(c)

Figure 16: comparision among LU_α, LU_β and LU_γ for matrix of size 9×9

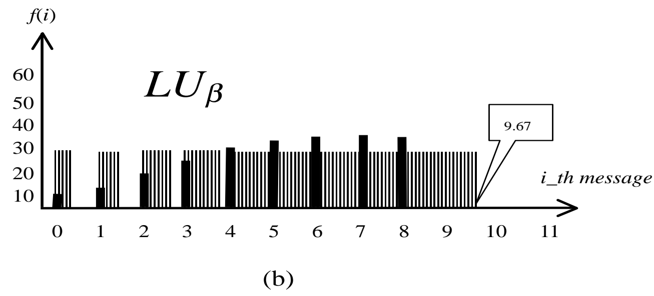
they have the same value of $F(|\mathcal{M}|-1)$. Compared with MM_α and MM_β, it does not delay the operations for the triggered computations to the end of the communication. The comparison among LU_α, LU_β and LU_γ for matrices of size 9×9 is shown in Figure 16 and Figure 17 . In the figure, we can observe that if we set $g(i) = 30$ for $0 \leq i < 8$ then $g(8)$s are equal to 70, 50, and 22 for LU_α, LU_β and LU_γ, respectively. Thus, if the system provides 30 CPU cycles/second for execution the task, then LU_α, LU_β and LU_γ can finish at 10.33, 9.67, and 8.73 seconds, respectively. .

6 Conclusion

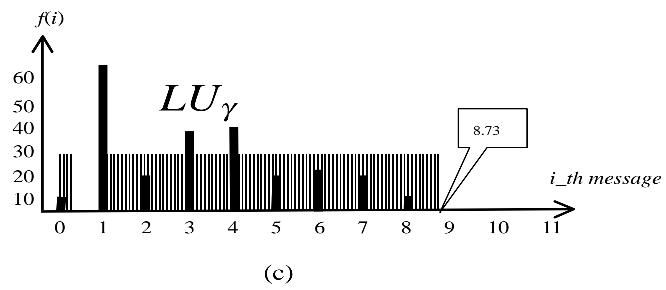
In this paper; an programming paradigm is proposed to maximize the utilization of the computing nodes. The approach employs a grouping function on the data locally at the data site before the data items is sent to the computing site. In a mobile computing environment, the network connection may be disconnected for a while. Our result can also be applied to keep the computing device busy for performing useful computations.



(a)



(b)



(c)

Figure 17: comparision among LU_α, LU_β and LU_γ for matrix of size 9×9

References

[1] P. B. Bhat, V. K. Prasanna, and C. S. Raghavendra,"Adaptive Communication Algorithms for Distributed Heterogeneous Systems," *Proc. of International Symposium on High Performance Distributed Computing,* 1998.

[2] Y. Chung, V. K. Prasanna and C.-L. Wang, "Parallel Algorithms for Linear Application on Distributed Memory Machine," *Proc. of DARPA Image Understanding Workshop,* 1996.

[3] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms,* The MIT Press; 1990.

[4] J. J. Dongarra, "Performance of Various Computers Using Standard Linear Equations Software, (Linpack Benchmark Report)," University of Tennessee, *Computer Science Technical Report, CS-89-58,* 2002.

[5] I. Foster, C. Kesselman, J. and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International J. of Supercomputer Applications,* 15(3), 2001.

[6] T. P. Green. DQS User Interface. *Technical Report,* Supercomputer Computations Research Institute, Florida State University, March 1996.

[7] J. L. Hennessy and D. A. Patterson, *Computer Architecture - A Quantitative Approach,* Morgan Kaufrnann, 1990.

[8] *IEEE Symposium Record - Hot Chips IV,* August 1992.

[9] Tsan-sheng Hsu, Joseph C. Lee, Dian Rae Lopez and William A. Royce, "Task Allocation on a Network of Processors," *IEEE Transactions on Computers,* volume 49, number 12, pages 1339–1353, 2000.

[10] H. J. Siegel et al., "Report of the Purdue Workshop on Grand Challenges in Computer Architecture for the support of High Performance Computing," *J. of Parallel and Distributed Computing,* 1992.

[11] S.-H. Yeh and' J.-J. Wu, "Efficient all-to-all broadcast in heterogeneous network of workstations," *Proc. of International ComputerSymposium,* Dec. 2000.