

# On Message Sequences for Incremental Network Computing\*

CHO-CHIN LIN

*Department of Electronic Engineering  
National Ilan University  
Yilan, 260 Taiwan  
E-mail: cclin@niu.edu.tw*

In a network computing platform, tasks compete with others for shared resources to communicate messages. Incremental computing masks communication latency by overlapping computation with communication. However, a sequence of messages with a large latency variance still makes computations proceed intermittently. In this paper, the impact of the message sequence on computation efficiency is studied and a framework which employs a well organized message sequence to maximize the efficiency of computations is introduced. Firstly, a network computing model for performing incremental computations is proposed. Based on the model, theorems are developed as the groundwork based on which algorithms for finding a well organized message sequence are derived. Finally, algorithms which find a well organized message sequence in  $O\left(\frac{r!}{k!}\right)^{k+1}$  and  $O\left(\frac{r!}{(k!)^k}\right)$  comparison steps are given for sending  $r$  input data items using  $\frac{r}{k}$  messages of a given size  $k$ .

**Keywords:** network computing, high performance computing, incremental computing, parallel and distributed computing, message sequence

## 1. INTRODUCTION

Due to the advances in information technologies, the capability of many personal devices such as PDA's or laptop computers can now meet the computational requirements of daily life. By employing the device, many services can be computed based on the data retrieved from a data storage. In general, the storage capacity of a personal device is limited. Thus, it is not possible to store world-wide data in the device. Recently, a network computing infrastructure [1] has been proposed to conquer the problem, in which network-wide repositories are integrated to store huge amount of data and the computing facilities compute the requested services based on the data retrieved through network. On the network computing platform, the computations proceed as follows: remote data repositories send sequences of messages to computing facilities, then the facilities use the recently arrived messages aggressively to compute the partial result. Since communicating messages requires both hardware modules to route messages and software message layers to deliver messages, communication latency can be experienced by the computing facilities. Researches [2, 3] have shown the software overhead is one of the major bottlenecks in communicating messages. However, the communication latency can still be unstable even though software overhead can be eliminated. The unstable communication latency comes from the nature of resource sharing inherited from net-

---

Received June 30, 2006; revised November 13, 2006 & January 2, 2007; accepted January 23, 2007.

Communicated by Sy-Yen Kuo.

\* This research was partially supported by the National Science Council of Taiwan, R.O.C. under grants No. NSC 95-2221-E-197-013 and NSC 96-2221-E-197-007.

work computing. In the network computing platform, the users need to share with others the data access capabilities of the repositories and the bandwidths of the communication channels. Thus, the messages from a data repository may arrive at a computing device intermittently and the unstable communication latency results in low CPU utilization of the computing device. Consider the computations  $((a + b)/(a - b))^2 + (c - d)$  performed at a computing device based on the data items  $a$ ,  $b$ ,  $c$  and  $d$  sent from a data repository using message  $m_0$  and  $m_1$  in order. The arriving interval of the two messages is  $g$  clocks. Assume the computing device takes 4 clocks and 1 clock to complete the computations  $((a + b)/(a - b))^2$  and  $(c - d)$ , respectively. If  $m_0 = \{a, b\}$  and  $m_1 = \{c, d\}$ , then the computing device completes the computations in  $\max\{4, g\} + 2$  clocks. However, if  $m_0 = \{c, d\}$  and  $m_1 = \{a, b\}$ , then the computing device completes the computations in  $\max\{1, g\} + 5$  clocks. Since  $\max\{4, g\} + 2 \leq \max\{1, g\} + 5$ , it is obvious that the first message sequence is better than the second sequence. It implies that the computation and communication issues are closely inter-wined in the network-based computing and should be studied together.

This research focuses on the area of network computing, where loosely coupled computers collaborate in the execution of one application. The computing devices in the network are provided with data from a data server and start computing incrementally. Incremental computing is a computing process which computes the partial result based on the partial input data. This paper analyzes how the order of the data sent to the computing devices influences the performance of the overall system. A framework which employs a well organized message sequence to maximize the efficiency of computations is also introduced. In this paper, a model, theorems and a new technique are proposed. First, a network computing model is proposed for studying the effect of message sequences on a network computing platform. Based on the model, the impact of a message sequence on computation efficiency is investigated. Then, theorems which profit the find of a well organized message sequence are given. Finally, algorithms which find a well organized message sequence in  $O\left(\left(\frac{r}{k}\right)^{k+1}\right)$  and  $O\left(\frac{r!}{(k!)^{\frac{r}{k}}}\right)$  comparison steps are developed, where  $r$  is the number of total data items and  $k$  is the size of a message. The effort of this study advances the current state of research in network computing. This study shows that hiding latency using a well organized message sequence can further squeeze the CPU cycles from a remote computing device. This paper also suggests that the *data layouts* in the data servers and the *computation protocols* between two collaborating computers should be tailored to match the computation structure of a task.

In the past, many researches have devoted to boosting the communication performance and the proposed techniques have been proved to be very successful. Several researches [4-10] have been concentrated on developing message scheduling strategies which maximize the utilization of communication bandwidth. They can be used to improve the efficiency of complicated communication routines. Communication latency hiding is a well-known technique to increase CPU utilization and several researches [11-14] have studied the effect of latency hiding. In [11], the gain of communication latency hiding by overlapping computation with communication was analyzed. However, overlapping computation with communication was not necessarily without cost. In [12], the effect of overlapping computation with communication was investigated for runtime-dependent network contention. In [13], the interaction of two latency hiding techniques,

pre-fetching and write buffer under weak consistency, with the limited bandwidth in a large multiprocessor was investigated. In [14], the performance of a system employing a dedicated communication processor and that of a system without employing a dedicated communication processor were compared. In [15], it has been shown that the pattern of a message sequence has a serious impact on the performance of network-based computing using LU decomposition as an example. In [16], several fundamental theorems have been given to profit the design of efficient algorithms on network computing platform. In [17], *dominant* data sequences have been found for the three applications: the product of two polynomials, matrix multiplication and Fast Fourier Transform. The authors also demonstrated that no dominant sequences exist for any of the three applications if a special data representation is used for the sparse cases of matrix multiplication, polynomial product and FFT computation. This paper extends the works in [15, 16].

A model, theorems and a new technique have been proposed in this paper. First, a model is given in section 2 to capture the characteristics of a network-based computing platform. The formal definition of incremental computing is given in section 3. Theorems profiting the designs of efficient applications and serving as the principle of the algorithms for finding a dominant message sequence are developed in section 4. In section 5, algorithms for finding a dominant message sequence which maximizes the CPU utilization are proposed. Finally, concluding remarks are made in section 6.

## 2. A MODEL FOR NETWORK COMPUTING PLATFORM

In this section, two parameters are given to capture the characteristics of a network computing platform. In the platform, the server is a repository which acts as a data provider and the computing device is a computation provider. Before a computing device starts to compute a service, it sends a request to the data server. After the data server has noticed the event of the request, it responds by sending data items to the device for computing the service. The operation can be summarized as follows: a server sends data items  $d_0 d_1 \dots d_{r-1}$  using a sequence of messages  $m_0 m_1 \dots m_{N-1}$  to a device and the device uses the data items as its input to compute the service concurrently. In this paper, it is assumed the order of the messages arriving at the computing device is the same as the order of the messages sent from the server. That is, if the server sends messages  $m_0 m_1 m_2 \dots m_{N-1}$  in order, then the computing device will receive  $m_i$  before  $m_{i+1}$  for  $0 \leq i < N - 1$ . An onto function  $\rho$  which maps  $\mathcal{A} = \{0, 1, 2, \dots, r - 1\}$  to  $\mathcal{B} = \{0, 1, 2, \dots, N - 1\}$  assigns a tag to each of the input data items. A data item  $d_i$  with a tag  $\rho(i)$  is sent by message  $m_{\rho(i)}$ . Note that a message can deliver more than one data item. For example, a data item  $d_0$  with a tag  $\rho(0) = 8$  is sent by message  $m_8$ . Another data item  $d_3$  with the same tag number  $\rho(3) = 8$  is also sent by the same message  $m_8$ . By employing the function, the order of sending input data items can be specified. The data items packed in a message are ready to be used only after the message has been completely received by the computing device. A computation which is executable based on the arrived data is defined as a *triggered* computation.

In the following paragraph, an abstract model of network computing is given. Let  $M$  be the sequence of messages  $m_0 m_1 \dots m_{N-1}$ . Define  $\mathcal{H}(M, k)$  as the function which calculates the number of triggered computations based on the first  $k + 1$  messages in  $M$ :  $m_0 m_1$

... $m_k$ . The first parameter is defined for each incoming message to express the amount of additional computations triggered at the computing device.

- $f_k$ : the  $k$ th computational fillet of a task (measured in number of CPU cycles). It is defined as the amount of additional computations triggered at a computing device when message  $m_k$  has been completely received. That is,  $f_k = \kappa(\mathcal{H}(M, k) - \mathcal{H}(M, k - 1))$ , where  $\kappa$  is the number of CPU cycles needed to complete a computation. Note that  $m_0$  is the first message, thus, define  $f_k = 0$  for  $k < 0$ .

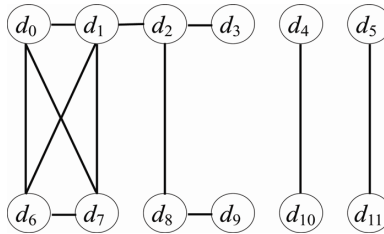


Fig. 1. Illustration of the computations performed by a task.

The definition of a computational fillet is illustrated by a computation task as shown in Fig. 1. In the figure, the computation structure of a task is represented by an undirected graph  $G(V, E)$ , where  $V = \{d_0, d_1, d_2, \dots, d_{11}\}$  and  $E$  is a set of edges. The existence of an edge  $(d_i, d_j)$  implies that computation of *one* unit can be performed if both  $d_i$  and  $d_j$  are available at the computing device. Assume the server sends the data items using messages  $m_0 m_1 m_2$ , where  $m_0 = \{d_0 d_1 d_2 d_3\}$ ,  $m_1 = \{d_4 d_5 d_6 d_7\}$  and  $m_2 = \{d_8 d_9 d_{10} d_{11}\}$ . According to the definition of the computational fillet, we have  $f_0 = 3$ ,  $f_1 = 5$  and  $f_2 = 4$ . It is obvious other data-sending sequences exist and the computational fillets are different. For example, if the sequence of the task in Fig. 1 is  $m_0 = \{d_4 d_5 d_6 d_7\}$ ,  $m_1 = \{d_0 d_1 d_2 d_3\}$  and  $m_2 = \{d_8 d_9 d_{10} d_{11}\}$ , then we have  $f_0 = 1$ ,  $f_1 = 7$  and  $f_2 = 4$ . In the following sections, denote  $\mathcal{F}$  and  $F_k = \sum_{i=0}^k f_i$  as sequence  $(f_0, f_1, f_2, \dots, f_{N-1})$  and the  $k$ th *accumulative* computational fillet, respectively. Let  $|S|$  denote the number of elements in set  $S$  or sequence  $S$ . In this paper, we have  $|\mathcal{F}| = N$ .

The second parameter of the model is given to capture the pattern of available CPU cycles, which are segmented by a sequence of messages. The parameter is defined as follows:

- $g_k$ : the  $k$ th computational gain for a task (measured in number of CPU cycles). It is the number of available CPU cycles for the computing device to perform computations at the interval of arrived messages  $m_k$  and  $m_{k+1}$ . Denote  $t_k$  and  $t_{k+1}$  as the times that messages  $m_k$  and  $m_{k+1}$  have been completely received by the computing device. Then,  $g_k = \tau(t_{k+1} - t_k)$ , where  $\tau$  is the number of CPU cycles per unit time provided by the computing device. If  $m_k$  is the last message, then  $g_k$  is defined to be the total CPU cycles needed to complete the computations specified by  $f_{N-1}$  plus the previous triggered computations which have not been executed. Note that  $m_0$  is the first message, thus, define  $g_k = 0$  for  $k < 0$ .

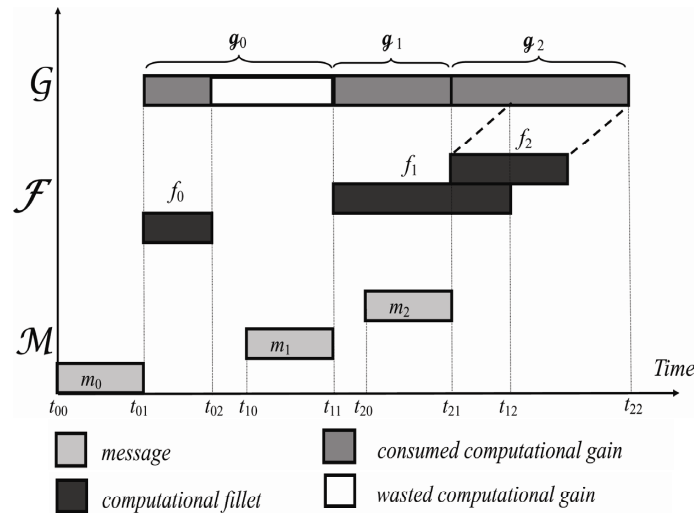


Fig. 2. Illustration of the incoming messages and the computations at a computing device.

The definition of a computational gain can be illustrated in Fig. 2. In the figure,  $t_{i0}$  is the time at which the computing device begins to receive message  $m_i$ ,  $t_{i1}$  is the time at which message  $m_i$  has been completely received and computational fillet  $f_i$  is triggered. Note that  $f_i$  is triggered immediately after message  $m_i$  has been completely received. In Fig. 2, the lengths of the black boxes representing the computational fillets are only for illustrating the various amounts of triggered computations for the received messages. In the figure,  $g_i$  is the number of available CPU cycles for the computing device to perform computations at the interval between  $t_{i1}$  and  $t_{i+11}$ . For a computing device with a fixed processing speed, the values of the computational gains may be variant due to the uncertainty of communication latency. For example, the value of  $g_0$  is larger than that of  $g_1$  in Fig. 2. In general, the values of computational gains depend not only on the processing speed of a computing device but also on the available bandwidth scheduled by the communication subsystem for transmitting messages. In the following sections, denote  $\mathcal{G}$  and  $G_k = \sum_{i=0}^k g_i$  as sequence  $(g_0, g_1, g_2, \dots, g_{N-2})$  and the  $k$ th accumulative computational gain, respectively. Note that  $g_{N-1}$  is not included in  $\mathcal{G}$ . It is worth mentioning that the values in the sequence of computational gains  $\mathcal{G}$  are related to the network traffic. Thus, the term *traffic pattern* will be used in stead of the term *computational gain sequence* wherever it is appropriate.

The parameters  $\rho$ ,  $f_k$  and  $g_k$  defined in this paper are summarized as follows. The data items  $d_0, d_1, \dots, d_{r-1}$  stored at the data provider are packed into a sequence of messages  $m_0 m_1 \dots m_{N-1}$  according to a given function  $\rho$ . Message  $m_k$  consists of all the data items  $d_i$ 's, where  $\rho(i) = k$  for  $0 \leq i < r$ . As message  $m_k$  arrives at the computing device, the number of extra executable computations is  $f_k$ . Before message  $m_{k+1}$  arrives, the computing device has  $g_k$  CPU cycles to perform the extra executable computations plus the previous unfinished computations if they exist.

### 3. INCREMENTAL COMPUTING

Overlapping computation with communication is one of the major techniques to increase CPU utilization. In a network computing platform, it can be achieved by communicating messages and performing computations concurrently. For example, in Fig. 2, receiving message  $m_2$  overlaps with the ongoing computations specified by  $f_1$  at the computing device. In the network computing platform, each device competes with others for the shared network bandwidth as well as the data access capability of data repositories. Thus, the messages may arrive at a computing device intermittently. In this case, the CPU utilization may become intolerably low. For example, in Fig. 2, the computing device is idle at the interval between  $t_{02}$  and  $t_{11}$  because the message  $m_1$  arrives at the computing device late. Although overlapping computation with communication is an effective technique to mask the latency, however, the number of computations which wait to be performed between two consecutive messages should be large enough to keep the device busy. To effectively overlap computation with communication, a message sequence should be tailored to tolerate the uncertainty of communication latency.

The computation structure of the task given in Fig. 1 is used as an example to illustrate that various message sequences lead to possibly different execution times. According to Fig. 1, there are 12 computations to be performed. Initially, the data items  $d_0, d_1, d_2, \dots, d_{11}$  are stored at a server and they are sent to the computing device for performing computations concurrently using a sequence of messages. One of the possible sequences is  $m_0 m_1 m_2$ , where  $m_0 = \{d_0 d_1 d_2 d_3\}$ ,  $m_1 = \{d_4 d_5 d_6 d_7\}$  and  $m_2 = \{d_8 d_9 d_{10} d_{11}\}$  as shown in Fig. 3 (a). The computational fillets triggered by the sequence are  $f_0 = 3, f_1 = 5$  and  $f_2 = 4$ . Let the interval between  $m_0$  and  $m_1$  be 6 CPU cycles and the other interval between  $m_1$  and  $m_2$  be 4 CPU cycles. That is, the computational gains are  $g_0 = 6$  and  $g_1 = 4$ . Assume one computation takes one CPU cycle and the starting time of execution is the time when the *first* message has been completely received by the computing device. Based on the assumption, the computing device takes 15 CPU cycles to complete the 12 computations and its CPU utilization is 80%. Another possible sequence is  $m'_0 m'_1 m'_2$ , where  $m'_0 = \{d_4 d_5 d_6 d_7\}$ ,  $m'_1 = \{d_0 d_1 d_2 d_3\}$  and  $m'_2 = \{d_8 d_9 d_{10} d_{11}\}$  as shown in Fig. 3 (b). The computational fillets triggered by the sequence are  $f_0 = 1, f_1 = 7$  and  $f_2 = 4$ . Let the arrival intervals of the consecutive messages be the same as the previous case. Then, the computing device takes 17 CPU cycles to complete the 12 computations and its CPU utilization is 71%. It is obvious that the first sequence leads to a better CPU utilization compared with the second sequence. However, the first sequence is not a best sequence. Another better sequence  $m''_0 m''_1 m''_2$  can be found, where  $m''_0 = \{d_0 d_1 d_6 d_7\}$ ,  $m''_1 = \{d_2 d_3 d_8 d_9\}$  and  $m''_2 = \{d_4 d_5 d_{10} d_{11}\}$  as shown in Fig. 3 (c). The computational fillets triggered by the sequence are  $f_0 = 6, f_1 = 4$  and  $f_2 = 2$ . Let the arrival intervals of the consecutive messages be the same as the previous cases. Then, the computing device takes 12 CPU cycles to complete the 12 computations and its CPU utilization is 100%. In the third case, the device keeps on performing computations without being idle. It is easy to verify that, for a given number of received messages, the third sequence makes the computing device accumulate more triggered computations than the other two. That is  $F_i'' \geq F_i \geq F_i'$  for  $0 \leq i \leq 2$ .

Network computing that employs incremental computing technique to mask latency can be described by a pair of sequences  $\mathcal{F}$  and  $\mathcal{G}$ . An incremental computing process is defined as follows.

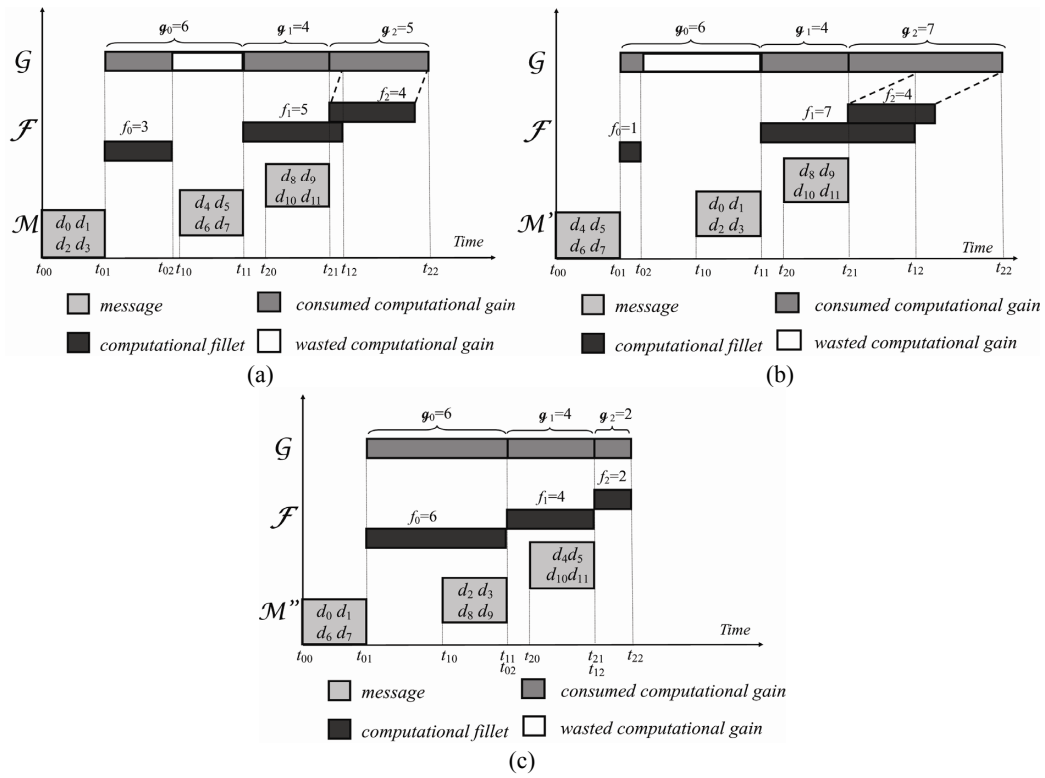


Fig. 3. Illustration of three different message sequences.

**Definition 1** In a network computing platform, the computing process in which a computing device employs available CPU cycles greedily to perform triggered computations is defined as incremental computing process. An incremental computing process is denoted by a 2-tuple  $(\mathcal{F}, \mathcal{G})$ .

Assume a server sends a sequence of messages  $m_0 m_1 \dots m_i \dots m_{N-1}$  to a computing device. The computational fillets of the messages are  $f_0 f_1 f_2 \dots f_i \dots f_{N-1}$  and their corresponding computational gains are  $g_0 g_1 g_2 \dots g_i \dots g_{N-1}$ . Two cases can be observed for  $f_i$  and  $g_i$ :  $g_i \leq f_i$  or  $g_i > f_i$ . For example, in Fig. 2,  $g_1$  is less than  $f_1$ . In this case, the execution of the triggered computations of  $f_i - g_i$  CPU cycles are delayed and will be performed later. Thus,  $f_i - g_i$  CPU cycles should be taken from  $g_{i+1} g_{i+2} \dots g_{N-2} g_{N-1}$  to complete part of the computations specified by  $f_i$ . For example, in Fig. 2, some of the CPU cycles specified by  $g_2$  are used to complete part of the computations specified by  $f_1$ . The other case:  $g_i$  is larger than  $f_i$ . In this case,  $g_i - f_i$  CPU cycles are wasted or used to perform the previous triggered computations that cannot get sufficient CPU cycles from their corresponding computational gain. For example, in Fig. 2,  $g_0$  is larger than  $f_0$ . Thus, the computing device is idle at the interval between time  $t_{02}$  and  $t_{11}$ . Then, the time to finish the task is delayed due to the wasted CPU cycles. The gap of a computational fillet  $f_i$  and its corresponding computational gain  $g_i$  is a performance measure which indicates how efficiently a computational gain is employed. Thus, the  $i$ th gain utilization index ( $gui$ ) denoted as  $\delta_i$  is defined

based on the difference of  $g_i$  and  $f_i$ . That is,  $\delta_i = g_i - f_i$ . Furthermore, the  $k$ th accumulative *gui*  $\Delta_k$  is defined as  $\sum_{i=-1}^k \delta_i$ . The pattern of a *gui* sequence indicates the CPU utilization of an incremental computing process. Based on the  $\delta_i$ , we define the *characteristic* sequence of an incremental computing process as follows:

**Definition 2** The characteristic sequence of  $(\mathcal{F}, \mathcal{G})$  is defined as  $(\delta_0, \delta_1, \delta_2, \dots, \delta_{N-2})$ . It is denoted as  $\mathcal{P}(\mathcal{F}, \mathcal{G})$ .

The execution time of a task denoted as  $\Phi(\mathcal{F}, \mathcal{G})$  is defined to be  $\sum_{i=0}^{N-1} g_i$ . The time to complete the partial computations specified by  $(f_0, f_1, f_2, \dots, f_k)$  is denoted as  $\Phi(\mathcal{F}_k, \mathcal{G}_{k-1})$ . Denote  $E(\mathcal{F}, \mathcal{G})$  as the CPU utilization of running  $(\mathcal{F}, \mathcal{G})$  and it is defined as  $F_{N-1}/\Phi(\mathcal{F}, \mathcal{G})$ . In Table 1,  $f_i$ ,  $\delta_i$  and  $\Delta_i$  for an incremental computing process  $(\mathcal{F}, \mathcal{G})$  are given as an example, where  $\mathcal{F} = (f_0, f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8)$  and  $\mathcal{G} = (g_0, g_1, g_2, g_3, g_4, g_5, g_6, g_7)$ . Note that, based on the definition of the computational gains, the value of  $g_8$  depends on sequences  $\mathcal{F}$  and  $\mathcal{G}$ . Thus, we need to calculate  $g_8$  using  $\mathcal{F}$  and  $\mathcal{G}$  before we can derive  $\Phi(\mathcal{F}, \mathcal{G})$ . In section 4, a theorem is given for deriving  $\Phi(\mathcal{F}, \mathcal{G})$  without knowing  $g_8$  in advance.

**Table 1. Parameters of an incremental computing process  $(\mathcal{F}, \mathcal{G})$ .**

$i$	0	1	2	3	4	5	6	7	8
$f_i$	2	9	1	4	3	5	9	4	2
$F_i$	2	11	12	16	19	24	33	37	39
$g_i$	2	7	6	2	5	7	7	2	6
$\delta_i$	0	-2	5	-2	2	2	-2	-2	4
$\Delta_i$	0	-2	3	1	3	5	3	1	5
$E(\mathcal{F}, \mathcal{G})$	88.6								

Denote  $\mathcal{S}_T$  as the set of computational fillet sequences for performing task  $T$  using a sequence of messages of a given size  $k$ . Since the number of input data items of task  $T$  is fixed, thus, the sequences in  $\mathcal{S}_T$  have the same number of computational fillets. Note that it is not necessary for  $\mathcal{S}_T$  to include all the combinations of the computational fillet sequences for performing task  $T$ . In the following definition, a special sequence of computational fillets is defined. Assume the accumulative computational fillets for sequences  $\mathcal{F}$  and  $\mathcal{F}'$  are denoted as  $F_k$  and  $F'_k$ , respectively.

**Definition 3** Let  $\mathcal{F}$  be a member of  $\mathcal{S}_T$ . If  $F_i \geq F'_i$  for  $0 \leq i < N$ , where  $\mathcal{F}'$  is any other member of  $\mathcal{S}_T$ , then  $\mathcal{F}$  is a dominant sequence in  $\mathcal{S}_T$ .

It will be shown, in section 4, that the execution time of a dominant sequence is less than that of any other sequence in  $\mathcal{S}_T$ .

## 4. GROUNDWORK

Theorems guiding the design of efficient applications and serving as the base for developing the algorithms of finding a dominant message sequence are given in this sec-

tion. Let  $\sigma$  be a permutation function. The following notations are given to simplify mathematical expressions. Assume  $\mathcal{L} = (l_0, l_1, \dots, l_a, \dots, l_b, \dots, l_{N-1})$ . Sequence  $\mathcal{L}_\sigma$  is formed by permuting the elements in  $\mathcal{L}$  using function  $\sigma$ .  $\mathcal{L}_{a,b}$  is a subsequence of  $\mathcal{L}$  which consists of  $l_a \dots l_b$ . If  $a = 0$ ,  $\mathcal{L}_b$  is used instead of  $\mathcal{L}_{0,b}$ . For example, if  $\mathcal{F} = (f_0, \dots, f_a, \dots, f_b, \dots, f_{N-1})$ , then  $\mathcal{F}_\sigma = (f'_0 \dots f'_{N-1})$  where  $f'_i = f_{\sigma(i)}$ ,  $\mathcal{F}_{a,b} = (f_a \dots f_b)$  and  $\mathcal{F}_b = (f_0 \dots f_b)$ . The time needed to complete the computations triggered by the received messages is given in Theorem 1. The correctness of the theorem is based on Lemma 1. Lemma 1 gives the recurrence relation for the time to complete the computations triggered by messages  $m_0 m_1 \dots m_{b+1}$ . In the lemma, the expressions at the righthand sides of the cases 1 and 2 consist of two terms. In the first case, the first term is the CPU cycles needed to complete the computations triggered by messages  $m_0 m_1 \dots m_{k+1}$  under a given traffic pattern  $\mathcal{G}_k$ , where  $k$  is the largest integer such that  $\Delta_k = \max\{\Delta_{-1}, \Delta_0, \Delta_1, \dots, \Delta_{b-1}\}$ . The second term is the total computations triggered by the remaining messages  $m_{k+2}, \dots, m_{b+1}$ . In the second case, the first term is the CPU cycles needed to complete the computations triggered by messages  $m_0 m_1 \dots m_b$  under a given traffic pattern  $\mathcal{G}_{b-1}$ . The second term is  $f_{b+1}$  plus extra CPU cycles. The number of extra CPU cycles is the interval of time at which the CPU is idle in waiting for the forthcoming message  $m_{b+1}$ .

**Lemma 1** Let  $k, k < b$ , be the largest integer such that  $\Delta_k = \max_{i=-1}^{b-1} \{\Delta_i\}$ , where  $b \leq |\mathcal{F}| - 2$ . Thus, we have:

- (1) If  $\Delta_b \leq \Delta_k$ , then  $\Phi(\mathcal{F}_{b+1}, \mathcal{G}_b) = \Phi(\mathcal{F}_{k+1}, \mathcal{G}_k) + \sum_{i=k+2}^{b+1} f_i$ .
- (2) If  $\Delta_b > \Delta_k$ , then  $\Phi(\mathcal{F}_{b+1}, \mathcal{G}_b) = \Phi(\mathcal{F}_b, \mathcal{G}_{b-1}) + (f_{b+1} + \Delta_b - \Delta_k)$ .

**Proof:** Cases (1) and (2) are proved separately. In the proof of this theorem,  $k < b$  and  $k$  is the largest integer such that  $\Delta_k = \max_{i=-1}^{b-1} \{\Delta_i\}$ .

**Case 1:**  $\Delta_b \leq \Delta_k$ . It implies  $\Delta_j - \Delta_k \leq 0$  for  $k+1 \leq j \leq b$ . Denote  $G_{k+1,j} = \sum_{i=k+1}^j g_i$  and  $F_{k+1,j} = \sum_{i=k+1}^j f_i$ . Since  $\Delta_j - \Delta_k = G_{k+1,j} - F_{k+1,j}$ , we have:  $G_{k+1,j} - F_{k+1,j} \leq 0$ , for  $k+1 \leq j \leq b$ . It implies that the available CPU cycles specified by  $g_{k+1}, g_{k+2}, \dots, g_j$  are not enough to perform computations needed by  $f_{k+1}, f_{k+2}, \dots, f_j$ . The statement leads to the following equation:

$$\Phi(\mathcal{F}_{b+1}, \mathcal{G}_b) = \Phi(\mathcal{F}_{k+1}, \mathcal{G}_k) + \sum_{i=k+2}^{b+1} f_i.$$

**Case2:**  $\Delta_b > \Delta_k$ . It implies  $\Delta_j - \Delta_k \leq 0$  for  $k+1 \leq j < b$  and  $\Delta_b - \Delta_k > 0$ . Thus, the following equations hold

$$\Delta_j - \Delta_k = G_{k+1,j} - F_{k+1,j} \leq 0 \text{ for } k+1 \leq j < b \text{ and} \quad (1)$$

$$\Delta_b - \Delta_k = \underbrace{(G_{k+1,b-1} - F_{k+1,b-1})}_B + \underbrace{(g_b - f_b)}_C > 0. \quad (2)$$

From Eqs. (1) and (2), it is known that terms  $B \leq 0$  and  $A > 0$ . Thus,  $C > 0$  must be true. It implies that some CPU cycles provided by  $g_b$  are used to perform computations

defined by the computational fillets  $f_{k+1}f_{k+2}\dots f_{b-1}$  that are short of CPU cycles. That is, the extra CPU cycles specified by term  $C$  can be used to perform the computations specified by term  $B$ . It is the number of computations which have not been performed when message  $m_b$  arrives at the computing device. However, there are still  $B + C$  wasted cycles which cannot be used by the forthcoming computational fillet  $f_{b+1}$ . Note that  $B + C = \Delta_b - \Delta_k$ . The statement leads to the following equation:

$$\Phi(\mathcal{F}_{b+1}, \mathcal{G}_b) = \Phi(\mathcal{F}_b, \mathcal{G}_{b-1}) + (f_{b+1} + \Delta_b - \Delta_k). \quad \square$$

Note that  $\mathcal{G}_{j-1}$  can be considered as the sequences of arrival intervals of the first  $j$  messages. The execution time of an incremental computing process is given in Theorem 1 which is derived based on Lemma 1. The theorem states the time needed to complete the computations specified by messages  $m_0m_1\dots m_j$  with intervals  $\mathcal{G}_{j-1}$  is the number of  $F_j$  cycles plus the maximum of  $\{0, \Delta_0, \Delta_1, \Delta_2, \dots, \Delta_{j-1}\}$ .

**Theorem 1**  $\Phi(\mathcal{F}_j, \mathcal{G}_{j-1}) = F_j + \max_{i=-1}^{j-1} \{\Delta_i\}$ .

**Proof:** Mathematical induction is used to show the correctness of Theorem 1.

**Basis:** According to the definition of computational gain,  $\mathcal{G}_1 = ()$ . It is easy to verify that  $\Phi(\mathcal{F}_0, \mathcal{G}_1) = F_0$  and  $\Phi(\mathcal{F}_1, \mathcal{G}_0) = F_1 + \max\{0, \Delta_0\}$ . It leads to that  $(\Phi(\mathcal{F}_j, \mathcal{G}_{j-1}) = F_j + \max_{i=-1}^{j-1} \{\Delta_i\})$  for  $j = 0, 1$ . Thus, it is true for  $j = 0, 1$ .

**Hypothesis:**  $\Phi(\mathcal{F}_j, \mathcal{G}_{j-1}) = F_j + \max_{i=-1}^{j-1} \{\Delta_i\}$  for  $j \leq b$  is true.

**Induction:** Considering  $j = b + 1$ , we want to prove:  $\Phi(\mathcal{F}_{b+1}, \mathcal{G}_b) = F_{b+1} + \max_{i=-1}^b \{\Delta_i\}$  is also true.

Let  $k, k < b$ , be the largest integer such that  $\Delta_k = \max_{i=-1}^{b-1} \{\Delta_i\}$ . Two cases need to be considered:  $\Delta_b \leq \Delta_k$  and  $\Delta_b > \Delta_k$ .

•  $\Delta_b \leq \Delta_k$ : Based on Lemmas 1 and Hypothesis, we have:

$$\begin{aligned} \Phi(\mathcal{F}_{b+1}, \mathcal{G}_b) &= \Phi(\mathcal{F}_{k+1}, \mathcal{G}_k) + \sum_{i=k+2}^{b+1} f_i && \text{Based on Lemma 1} \\ &= F_{k+1} + \max_{i=-1}^k \{\Delta_i\} + \sum_{i=k+2}^{b+1} f_i && \text{Based on Hypothesis} \\ &= F_{b+1} + \max_{i=-1}^k \{\Delta_i\} \\ &= F_{b+1} + \max_{i=-1}^b \{\Delta_i\}. \end{aligned}$$

Thus, the first case is proved.

•  $\Delta_b > \Delta_k$ : Based on Lemma 1 and Hypothesis, we have:

$$\begin{aligned}
 \Phi(\mathcal{F}_{b+1}, \mathcal{G}_b) &= \Phi(\mathcal{F}_b, \mathcal{G}_{b-1}) + f_{b+1} + \Delta_b - \Delta_k && \text{Based on Lemma 1} \\
 &= F_b + \max_{i=-1}^{b-1} \{\Delta_i\} + f_{b+1} + \Delta_b - \Delta_k && \text{Based on Hypothesis} \\
 &= F_{b+1} + \Delta_b \\
 &= F_{b+1} + \max_{i=-1}^b \{\Delta_i\}.
 \end{aligned}$$

Thus, the second case is proved. □

The following corollary is a direct result of Theorem 1. It is the formula to calculate the time needed to complete the incremental computing process  $(\mathcal{F}, \mathcal{G})$ .

**Corollary 1** Given an incremental computing process  $(\mathcal{F}, \mathcal{G})$  and  $|\mathcal{F}| = N$ , then, the execution time  $\Phi(\mathcal{F}, \mathcal{G})$  is  $F_{N-1} + \max_{i=-1}^{N-2} \{\Delta_i\}$ .

**Proof:** Since  $\mathcal{F} = (f_0, f_1, \dots, f_{N-1})$ , it is obvious according to Theorem 1. □

The execution time of the incremental computing process illustrated in Table 1 can be derived using Corollary 1. Its execution time is  $F_8 + \max_{i=-1}^7 \{\Delta_i\} = 44$ . Let  $\mathcal{F}$  and  $\mathcal{F}'$  be members of  $\mathcal{S}_T$ . The maximal accumulative *gui* of  $(\mathcal{F}, \mathcal{G})$  and  $(\mathcal{F}', \mathcal{G}')$  are  $\Delta_k$  and  $\Delta'_{k'}$ , respectively. The comparison on the execution times of the incremental computing processes with various maximal accumulative *gui*'s is given in Corollary 2. The corollary states that the incremental computing process with a smaller maximal accumulative *gui* can finish earlier than those with larger maximal accumulative *gui*'s.

**Corollary 2** If  $\mathcal{F}, \mathcal{F}' \in \mathcal{S}_T$  and  $\Delta_k < \Delta'_{k'}$ , then  $\Phi(\mathcal{F}, \mathcal{G}) < \Phi(\mathcal{F}', \mathcal{G}')$ .

**Proof:** According to Theorem 1, the execution time of  $(\mathcal{F}, \mathcal{G})$  is  $F_{N-1} + \max_{i=-1}^{N-2} \{\Delta_i\}$  and the execution time of  $(\mathcal{F}', \mathcal{G}')$  is  $F'_{N-1} + \max_{i=-1}^{N-2} \{\Delta'_i\}$ .  $\mathcal{F}$  and  $\mathcal{F}' \in \mathcal{S}_T$  implies  $F_{N-1} = F'_{N-1}$ . Since  $F_{N-1} = F'_{N-1}$  and  $\Delta_k < \Delta'_{k'}$ , it leads to  $\Phi(\mathcal{F}, \mathcal{G}) < \Phi(\mathcal{F}', \mathcal{G}')$ . □

Let  $A$  be an increasing sequence  $(a_0, a_1, \dots, a_i, a_{i+1}, \dots, a_{N-1})$  in which  $a_{i+1}$  is no less than  $a_i$ , for all  $i$ . Lemma 2 gives a fundamental property of an increasing sequence which is used as the base in proving Theorem 2. Denote the  $\ell$ th prefix sum of  $A$  as  $\sum_{i=0}^{\ell} a_i$ . The lemma states that the  $\ell$ th prefix sum of  $A$  is no larger than the  $\ell$ th prefix sum of  $A_\sigma$  which is formed by permuting the elements in  $A$  using the permutation function  $\sigma$ .

**Lemma 2** Let  $A = (a_0, a_1, \dots, a_{N-1})$  and  $A_\sigma = (a'_0, a'_1, \dots, a'_{N-1})$ . If  $A$  is an increasing sequence, then  $\sum_{i=0}^{\ell} a_i \leq \sum_{i=0}^{\ell} a'_i$ , for all  $\ell, 0 \leq \ell < N$ .

**Proof:** Let set  $S = \{a_0, a_1, a_2, \dots, a_\ell\}$  and  $S_\sigma = \{a'_0, a'_1, a'_2, \dots, a'_\ell\}$ . Denote  $\bar{S}$  as the complement of  $S$ . That is  $\bar{S} = \{a_{\ell+1}, a_{\ell+2}, a_{\ell+3}, \dots, a_{N-1}\}$ . It is obvious that for any  $x \in S$  and  $y \in \bar{S}$ , we have:  $x \leq y$ ; otherwise,  $A$  cannot be an increasing sequence. Notations  $S'$  and  $S'_\sigma$  are used to represent the sets which are formed by excluding the common

elements from  $S$  and  $S_\sigma$ , respectively. That is,  $S' = S - (S \cap S_\sigma)$  and  $S'_\sigma = S_\sigma - (S \cap S_\sigma)$ . Since  $S'_\sigma \subseteq (S \cup \bar{S})$  and  $S \cap S'_\sigma = \emptyset$ , we have  $S'_\sigma \subseteq \bar{S}$ . It implies that for any  $x \in S'$  and  $y \in S'_\sigma$ ,  $x \leq y$ . In addition,  $|S'| = |S'_\sigma|$  because of  $|S| = |S_\sigma|$ . Thus,  $\sum_{x \in S'} x \leq \sum_{y \in S'_\sigma} y$ . It leads to  $\sum_{x \in S} x \leq \sum_{y \in S_\sigma} y$ . Thus, this lemma is proved.  $\square$

Theorem 2 compares the execution times for the two processes: one with an increasing characteristic sequence and the other with the permuted version of the sequence denoted as  $\mathcal{P}(\mathcal{F}, \mathcal{G})_\sigma$ . Note that the characteristic sequence  $\mathcal{P}(\mathcal{F}, \mathcal{G})_\sigma$  is formed by permuting the elements of the characteristic sequence  $\mathcal{P}(\mathcal{F}, \mathcal{G})$ .

**Theorem 2** If  $\mathcal{P}(\mathcal{F}, \mathcal{G})$  is an increasing sequence, then  $\Phi(\mathcal{F}, \mathcal{G}) \leq \Phi(\mathcal{F}, \mathcal{G})_\sigma$ .

**Proof:** Since  $\mathcal{P}(\mathcal{F}, \mathcal{G})$  is an increasing sequence, then  $\delta_{i+1} \geq \delta_i$  for all  $i$ . Let  $\mathcal{P}(\mathcal{F}, \mathcal{G})_\sigma = (\delta'_0, \delta'_1, \delta'_2, \dots, \delta'_{N-1})$ , where  $\delta'_i = \delta_{\sigma(i)}$ . Denote  $\sum_{i=0}^b \delta'_i = \Delta'_b$ . Let  $k$  and  $k'$  be the largest integers such that  $\Delta_k = \max_{i=1}^{N-2} \{\Delta_i\}$  and  $\Delta'_{k'} = \max_{i=1}^{N-2} \{\Delta'_i\}$ . From Lemma 2, it is known  $\Delta_i \leq \Delta'_{k'}$  for all  $i$ ,  $0 \leq i < N$ . Thus,  $\Delta \leq \Delta_k \leq \Delta'_{k'}$ . It implies  $\Phi(\mathcal{F}, \mathcal{G}) \leq \Phi(\mathcal{F}, \mathcal{G})_\sigma$ .  $\square$

Incremental computing processes  $(\mathcal{F}, \mathcal{G})$  and  $(\mathcal{F}', \mathcal{G}')$  are *fairly treated* if  $g_i = g'_i$  for all  $0 \leq i \leq N-2$ . Incremental computing processes  $(\mathcal{F}, \mathcal{G})$  and  $(\mathcal{F}', \mathcal{G}')$  are *unfairly treated* if there exists an  $i$ ,  $0 \leq i \leq N-2$ , such that  $g_i \neq g'_i$ . Corollaries 3 and 4 are statements regarding fairly treated processes and unfairly treated processes, respectively. Corollary 3 states that for a given traffic pattern  $\mathcal{G}$  of fixed message arrival intervals, the time to complete the task specified by a decreasing sequence  $\mathcal{F}$  is no more than that specified by its permuted version.

**Corollary 3** Let  $(\mathcal{F}, \mathcal{G})$  and  $(\mathcal{F}_\sigma, \mathcal{G})$  be fairly treated. If  $\mathcal{F}$  is a decreasing sequence and  $g_i = g_j$  for  $0 \leq i, j \leq N-2$ , then  $\Phi(\mathcal{F}, \mathcal{G}) \leq \Phi(\mathcal{F}_\sigma, \mathcal{G})$ .

**Proof:** Since  $\mathcal{F}$  is a decreasing sequence and  $g_i = g_j$  for all  $0 \leq i, j \leq N-2$ , we have: (1)  $\mathcal{P}(\mathcal{F}, \mathcal{G})$  is an increasing sequence and (2)  $\mathcal{P}(\mathcal{F}_\sigma, \mathcal{G}) = \mathcal{P}(\mathcal{F}, \mathcal{G})_\sigma$ . Based on Theorem 2,  $\Phi(\mathcal{F}, \mathcal{G}) \leq \Phi(\mathcal{F}_\sigma, \mathcal{G})$  is true.  $\square$

Corollary 4 states that if each of the incoming messages triggers a fixed number of computations then the time to complete the task under the traffic pattern  $\mathcal{G}$  of increasing arrival intervals is no more than that under the permuted version of  $\mathcal{G}$ .

**Corollary 4** Let  $(\mathcal{F}, \mathcal{G})$  and  $(\mathcal{F}, \mathcal{G}_\sigma)$  be unfairly treated. If  $\mathcal{G}$  is an increasing sequence and  $f_i = f_j$  for  $0 \leq i, j \leq N-1$ , then  $\Phi(\mathcal{F}, \mathcal{G}) \leq \Phi(\mathcal{F}, \mathcal{G}_\sigma)$ .

**Proof:** Since  $\mathcal{G}$  is an increasing sequence and  $f_i = f_j$  for all  $0 \leq i, j \leq N-1$ , we have: (1)  $\mathcal{P}(\mathcal{F}, \mathcal{G})$  is an increasing sequence, and (2)  $\mathcal{P}(\mathcal{F}, \mathcal{G}_\sigma) = \mathcal{P}(\mathcal{F}, \mathcal{G})_\sigma$ . Based on Theorem 2,  $\Phi(\mathcal{F}, \mathcal{G}) \leq \Phi(\mathcal{F}, \mathcal{G}_\sigma)$  is true.  $\square$

Corollaries 3 and 4 consider either the elements in  $\mathcal{F}$  have the same value or the elements in  $\mathcal{G}$  have the same value. The following theorem considers the general case the values of the elements in  $\mathcal{G}$  and  $\mathcal{F}$  are variant.

**Theorem 3** Let  $(\mathcal{F}, \mathcal{G})$  and  $(\mathcal{F}_\sigma, \mathcal{G})$  be fairly treated. If  $\mathcal{F}$  is a decreasing sequence then  $\Phi(\mathcal{F}, \mathcal{G}) \leq \Phi(\mathcal{F}_\sigma, \mathcal{G})$ .

**Proof:** Let  $\mathcal{F} = (f_0, f_1, \dots, f_{N-1})$  and  $\mathcal{F}_\sigma = (f'_0, f'_1, f'_2, \dots, f'_{N-1})$ . Based on  $\mathcal{F}$  and  $\mathcal{F}_\sigma$ ,  $\tilde{\mathcal{F}}$  and  $\tilde{\mathcal{F}}_\sigma$  are formed by attaching a negative sign to each of the elements in  $\mathcal{F}$  and  $\mathcal{F}_\sigma$ . That is,  $\tilde{\mathcal{F}} = (-f_0, -f_1, \dots, -f_{N-1})$  and  $\tilde{\mathcal{F}}_\sigma = (-f'_0, -f'_1, -f'_2, \dots, -f'_{N-1})$ . Since  $\mathcal{F}$  is a decreasing sequence,  $\tilde{\mathcal{F}}$  is an increasing sequence. According to Lemma 2,  $\sum_{i=0}^\ell (-f_i) \leq \sum_{i=0}^\ell (-f'_i)$  for all  $0 \leq \ell < N$ . It implies that  $\sum_{i=0}^b (g_i - f_i) \leq \sum_{i=0}^b (g_i - f'_i)$  for all  $b$ . Let  $\Delta_b = \sum_{i=0}^b (g_i - f_i)$  and  $\Delta'_b = \sum_{i=0}^b (g_i - f'_i)$ . Thus,  $\Delta_i \leq \Delta'_i$  for all  $0 \leq i \leq N-2$ . It leads to that  $F_{N-1} + \max_{i=-1}^{N-2} \{\Delta_i\} \leq F_{N-1} + \max_{i=-1}^{N-2} \{\Delta'_i\}$ . Thus, the statement:  $\Phi(\mathcal{F}, \mathcal{G}) \leq \Phi(\mathcal{F}_\sigma, \mathcal{G})$  is true.  $\square$

The following theorem considers the execution times of different message sequences in  $\mathcal{S}_T$  which are sent to a computing device for performing task  $T$ . Let the  $i$ th accumulative fillets of  $(\mathcal{F}, \mathcal{G})$  and  $(\mathcal{F}', \mathcal{G})$  be denoted as  $F_i$  and  $F'_i$ . Theorem 4 states that for a given traffic pattern  $\mathcal{G}$ , the incremental computing process which accumulates more computations than the other processes at any moment can finish its task earlier.

**Theorem 4**  $\mathcal{F}$  and  $\mathcal{F}' \in \mathcal{S}_T$ . If  $F_i \geq F'_i$  for all  $0 \leq i \leq N-2$ , then  $\Phi(\mathcal{F}, \mathcal{G}) \leq \Phi(\mathcal{F}', \mathcal{G})$ .

**Proof:** Since  $F_i \geq F'_i$  for all  $0 \leq i \leq N-2$ , it implies  $G_i - F_i \leq G_i - F'_i$ , for all  $0 \leq i \leq N-2$ . Thus,  $\Delta_i \leq \Delta'_i$  for all  $0 \leq i \leq N-2$ . According to Theorem 1, the statement:  $\Phi(\mathcal{F}, \mathcal{G}) \leq \Phi(\mathcal{F}', \mathcal{G})$  is true.  $\square$

The following corollary is a direct result of Theorem 4. Corollary 5 states that a dominant sequence has the shortest execution time for any given traffic pattern  $\mathcal{G}$ .

**Corollary 5** If  $\mathcal{F}$  is a dominant sequence in  $\mathcal{S}_T$ , then  $\Phi(\mathcal{F}, \mathcal{G}) \leq \Phi(\mathcal{F}', \mathcal{G})$ , for any other  $\mathcal{F}' \in \mathcal{S}_T$ .

**Proof:** Since  $\mathcal{F}, \mathcal{F}' \in \mathcal{S}_T$  and  $\mathcal{F}$  is dominant, then  $F_i \geq F'_i$  for  $0 \leq i \leq N-2$ . Based on Theorem 4, this corollary is proved.  $\square$

Theorem 5 shows that the execution time of the task specified by an increasing computational fillet sequence is no more than that of the same task specified by a decreasing computational fillet sequence.

**Theorem 5**  $\mathcal{F}$  and  $\mathcal{F}' \in \mathcal{S}_T$ . If  $\mathcal{F}$  and  $\mathcal{F}'$  are decreasing and increasing sequences, respectively, then  $\Phi(\mathcal{F}, \mathcal{G}) < \Phi(\mathcal{F}', \mathcal{G})$ .

**Proof:** Firstly, the proof of the correctness on the claim:  $F_i \geq F'_i$  for all  $0 \leq i < N$  is conducted. Since  $\mathcal{F}$  is a decreasing sequence and  $\mathcal{F}'$  is an increasing sequence,  $f_0 \geq f'_0$  must be true. Otherwise,  $\mathcal{F}$  and  $\mathcal{F}'$  cannot be both in  $\mathcal{S}_T$ . So,  $F_0 \geq F'_0$ . Assume the claim is not true for some  $i > 0$ . Thus, a smallest integer  $h$ , for  $1 \leq h \leq N-2$ , such that  $F_h < F'_h$  can be found. It implies that  $F_h - F_{h-1} < F'_h - F'_{h-1}$ . It leads to that  $f_h < f'_h$ . Since  $\mathcal{F}$  is a decreasing sequence and  $\mathcal{F}'$  is an increasing sequence, it implies that  $f_{N-1} \leq f_{N-2} \leq \dots \leq f_{h+1} \leq f_h <$

$f'_h \leq f'_{h+1} \dots \leq f'_{N-2} \leq f'_{N-1}$ . Thus,  $\sum_{i=h+1}^{N-1} f_i < \sum_{i=h+1}^{N-1} f'_i$ . It implies that  $F_h + \sum_{i=h+1}^{N-1} f_i < F'_h + \sum_{i=h+1}^{N-1} f'_i$ . That is,  $F_{N-1} < F'_{N-1}$ . It is a contradiction. Thus, it leads to the correctness of the claim:  $F_i \geq F'_i$  for all  $0 \leq i \leq N-2$ . Since  $F_i \geq F'_i$  for all  $0 \leq i \leq N-2$ ,  $\Phi(\mathcal{F}, \mathcal{G}) \leq \Phi(\mathcal{F}', \mathcal{G})$  is true according to Theorem 4.  $\square$

## 5. FINDING A DOMINANT SEQUENCE

Given a set  $S$  of messages, the *Dominant Subsequence* problem (DSP) is to find a subset  $S'$  of  $k$  messages such that the number of executable computations triggered by  $S'$  is no less than that triggered by any other subset of  $k$  messages. To show DSP is  $\mathcal{NP}$ -hard, the *Heaviest Subgraph* problem (HSP) [18] is introduced first. For an undirected graph  $G = (V, E)$  with nonnegative edge weights  $w_{ij}$  for  $(v_i, v_j) \in E$  and an integer  $k \leq N = |V|$ , the HSP problem is to determine a subset  $V'$  of  $k$  vertices such that the weight of the subgraph induced by  $V'$  is maximized. Let  $c_{ij}$  denote the number of executable computations triggered by a pair of messages  $m_i$  and  $m_j$  after the pair of messages have been completely received by the computing device. Given a set of  $N$  messages, the computations performed by a task is defined as follows:  $c_{ij} = w_{ij}$  if  $w_{ij} \neq 0$ ; otherwise  $c_{ij} = 0$ . It is obvious that HSP is polynomial-time reducible to DSP. Thus, DSP is  $\mathcal{NP}$ -hard. In this section, assume the number of data items  $r$  is a multiple of  $k$ , where  $k$  is the number of data items in a message. Based on the theorems developed in section 4, algorithms which find a well organized message sequence in  $O\left(\left(\frac{r}{k}\right)^{k+1}\right)$  and  $O\left(\frac{r!}{(k!)^{\frac{r}{k}}}\right)$  comparison steps are given for sending  $r$  input data items using messages of a given size  $k$ .

Theorem 4 states that a dominant message sequence minimizes the time of executing a task across a network. According to Definition 3, a dominant sequence  $\mathcal{F}$  of  $\mathcal{S}_T$  must satisfy  $F_i \geq F'_i$  for all  $0 \leq i \leq N-2$ , where  $\mathcal{F}'$  is any other sequence in  $\mathcal{S}_T$ . Assume  $\mathcal{S}_T$  consists of all the possible computational fillet sequences of task  $T$  and there is at least one dominant sequence in  $\mathcal{S}_T$ . The following steps are given to find a dominant sequence from a special type of  $\mathcal{S}_T$  in which  $F_i \neq F'_i$  for all  $0 \leq i \leq N-2$ . Let  $S$  and  $S'$  be sets of data items. The data items needed to perform task  $T$  are  $d_0, d_1, \dots, d_{r-1}$  and each message consists of  $k$  data items.

1. Initially,  $i \leftarrow 0$ ,  $S' \leftarrow \emptyset$  and  $S \leftarrow \{d_0, d_1, \dots, d_{r-1}\}$ .
2. Choose  $k$  data items  $d_{j'_0}, d_{j'_1}, \dots, d_{j'_{k-1}}$  from set  $S$  such that  $F_i$  is largest.
3. Pack the chosen data items into message  $m_i$  and define  $\rho(j'_h) = i$  for  $0 \leq h < k$ .
4.  $S \leftarrow S - \{d_{j'_0}, d_{j'_1}, \dots, d_{j'_{k-1}}\}$  and  $S' \leftarrow S' \cup \{d_{j'_0}, d_{j'_1}, \dots, d_{j'_{k-1}}\}$ .
5.  $i \leftarrow i + 1$ . Go to step 2 for the next iteration if there is any data item left in set  $S$ .

The uncertainties of communication latency can be effectively masked by sending the computational fillet with the largest value first. Thus, step 2 is to make the largest amount of computations triggered as early as possible by packing the computation-correlated data items into one message. The computational fillets with large values can keep CPU busy even though the next message is blocked by heavy network traffic. In step 2, the number of triggered computations is calculated based on the currently chosen data

items and the data items in  $S'$ . Then, the number is compared with the temporary largest number which is the  $F_i$  of another set of  $k$  data items previously chosen from set  $S$ . To select  $k$  appropriate data items from set  $S$  at iteration  $i$ ,  $C_k^{r-ki}$  comparisons are needed. Since the algorithm is proposed to find a dominant sequence from a special type of  $\mathcal{S}_T$ , only one message of  $k$  data items which leads to the maximal number of triggered computations can be found at step 2. Thus, the algorithm finds a dominant sequence in  $\sum_{i=0}^{\frac{r}{k}-1} C_k^{r-ki}$  comparison steps. That is, it takes  $O(\left(\frac{r}{k}\right)^{k+1})$  comparison steps to find a dominant sequence from  $\mathcal{S}_T$  for a given size  $k$ .

```

Algorithm FindSequence( $\{d_0, d_1, \dots, d_{r-1}\}, CS, k$ )
{
   $i \leftarrow 0$ ;
   $S_0^{ki} \leftarrow \{d_0, d_1, \dots, d_{r-1}\}$ ;
   $\hat{S}_0^{ki} \leftarrow \emptyset$ ;
   $\Lambda_i \leftarrow \{(S_0^{ki}, \hat{S}_0^{ki}, 0)\}$ ;
  repeat
     $\Lambda_{i+1} \leftarrow \emptyset$ ;
    for each  $(S_j^{ki}, \hat{S}_j^{ki}, z_j^{ki}) \in \Lambda_i$ ;
       $\Lambda_{i+1} \leftarrow \Lambda_{i+1} \cup \Upsilon(S_j^{ki}, \hat{S}_j^{ki}, CS, k)$ ;
     $\Lambda_{i+1} \leftarrow \text{ExtractMultipleMax}(\Lambda_{i+1})$ ;
     $i \leftarrow i + 1$ ;
  until  $(i = r/k)$ ;
  return  $(\Lambda_i)$ ;
}
end FindSequence

```

Fig. 4. Algorithm for finding a dominant sequential in  $\mathcal{S}_T$ .

Assume  $\mathcal{S}_T$  consists of all the possible message sequences for performing task  $T$  and there is at least one dominant sequence in  $\mathcal{S}_T$ . A general algorithm which finds a dominant sequence is given in Fig. 4. The algorithm takes  $\{d_0, d_1, \dots, d_{r-1}\}$ ,  $CS$  and  $k$  as its input parameters.  $\{d_0, d_1, \dots, d_{r-1}\}$  is the set of input data items sent from a server to a computing device.  $CS$  is a collection of 2-tuples  $(S, F_S)$ , where  $S$  is a subset of the input data items and  $F_S$  is the number of total executable computations after the computing device has received all the data items in  $S$ .  $k$  is the number of data items in a message. Without loss of generality, assume  $r$  is a multiple of  $k$ . Three variables are used in the algorithm:  $S_j^{ki}$ ,  $\hat{S}_j^{ki}$  and  $z_j^{ki}$ .  $S_j^{ki}$  is the set of data items left in the server after messages  $m_0, m_1, \dots, m_{i-1}$  have been sent.  $\hat{S}_j^{ki}$  is the set of data items received by the computing device after messages  $m_0, m_1, \dots, m_{i-1}$  have been sent.  $z_j^{ki}$  is the accumulated triggered computations when the computing device receives messages  $m_0, m_1, \dots, m_{i-1}$ . Note that  $j$  is the index for  $S_j^{ki}$ ,  $\hat{S}_j^{ki}$  and  $z_j^{ki}$ . In the algorithm, function  $\Upsilon$  takes  $S_j^{ki}$ ,  $\hat{S}_j^{ki}$ ,  $CS$  and  $k$  as its input. Its output is a set of triples  $\{(S_0^{ki+k}, \hat{S}_0^{ki+k}, z_0^{ki+k}), (S_1^{ki+k}, \hat{S}_1^{ki+k}, z_1^{ki+k}), \dots, (S_l^{ki+k}, \hat{S}_l^{ki+k}, z_l^{ki+k}), \dots\}$ . The function chooses data items  $d_{j'_0}, d_{j'_1}, \dots, d_{j'_{k-1}}$  from  $S_j^{ki}$ , where  $d_{j'_0},$

$d_{j_1}, \dots, d_{j_{k-1}}$  are the candidates to form a new message. Then, set is formed by excluding the data items  $d_{j_0}, d_{j_1}, \dots, d_{j_{k-1}}$  from  $S_j^{ki}$  and set  $\hat{S}_l^{ki+k}$  is formed by including the data items of  $\hat{S}_j^{ki}$  and  $d_{j_0}, d_{j_1}, \dots, d_{j_{k-1}}$ . Based on the data items in set  $\hat{S}_j^{ki+k}$ ,  $z_j^{ki+k}$  can be calculated. In the algorithm, set  $\Lambda_{i+1}$  temporarily stores the output of  $\Upsilon(S_j^{ki}, \hat{S}_j^{ki}, CS, k)$  for all  $j$ . Let  $z_{\max}^{ki+k} = \max\{z_l^{ki+k} \mid (S_l^{ki+k}, \hat{S}_l^{ki+k}, z_l^{ki+k}) \in \Lambda_{i+1}\}$ . Then, the triples with their  $z_l^{ki+k} = z_{\max}^{ki+k}$  are extracted from  $\Lambda_{i+1}$  using function *ExtractMultipleMax* and all the elements in  $\Lambda_{i+1}$  are replaced by those triples with their  $z_l^{ki+k} = z_{\max}^{ki+k}$ . At the end of iteration  $i$ , only the triples with the maximal  $z_j^{ki+k}$  are kept in  $\Lambda_{i+1}$ . According to the definition of a dominant sequence, algorithm *FindSequence* finds dominant sequences for performing task  $T$  and it finds dominant sequences in  $\prod_{i=0}^{k-1} C_k^{r-ki}$  comparison steps. That is, it takes  $O\left(\frac{r!}{(k!)^{\frac{r}{k}}}\right)$  comparison steps to find dominant sequences. The algorithm employs the branch-and-bound technique to focus on searching dominant sequences by pruning the subsequences with smaller  $z_l^{ki+k}$ . Thus, the complexity of the number of comparison steps can be lower than  $O\left(\frac{r!}{(k!)^{\frac{r}{k}}}\right)$ . If the number of triples  $(S_l^{ki+k}, \hat{S}_l^{ki+k}, z_l^{ki+k})$ 's with their third term  $z_l^{ki+k}$ 's equal to  $z_{\max}^{ki+k}$  is no more than a constant  $c$  at the end of iteration  $i$ , for all  $i$ , then the complexity of comparison steps is  $O\left(\left(\frac{r}{k}\right)^{k+1}\right)$ . Based on Corollary 5, the found dominant sequence leads to the minimal execution time compared with the other sequences in  $\mathcal{S}_T$ .

## 6. CONCLUDING REMARKS

In this paper, a network computing model has been proposed for studying the effect of message sequences on a network computing platform. Based on the model, the impact of a message sequence on CPU utilization has been investigated. It has been shown that well organized message sequences have the computing power of a computation provider to be employed efficiently. A novel technique which employs a well organized message sequence to maximize the efficiency of computations has been introduced in this paper. Theorems which profit the find of a well organized message sequence have been given. Finally, algorithms which find a well organized message sequence in  $O\left(\left(\frac{r}{k}\right)^{k+1}\right)$  and  $O\left(\frac{r!}{(k!)^{\frac{r}{k}}}\right)$  comparison steps have also been developed.

## REFERENCES

1. I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid: enabling scalable virtual organizations," *International Journal of Supercomputer Applications*, Vol. 15, 2001, pp. 200-222.
2. S. Chandra, J. R. Larus, and A. Rogers, "Where is time spent in message-passing and shared-memory programs?" in *Proceedings of International Conference on Architectural Support of Programming Languages and Operating Systems*, 1994, pp. 61-73.

3. V. Karamcheti and A. A. Chien, "Software overhead in messaging layers: where does the time go?" in *Proceedings of International Conference on Architectural Support of Programming Languages and Operating Systems*, 1994, pp. 51-60.
4. P. Liu, "Broadcast scheduling optimization for heterogeneous cluster systems," *Journal of Algorithms*, Vol. 42, 2002, pp. 135-152.
5. P. B. Bhat, V. K. Prasanna, and C. S. Raghavendra, "Efficient collective communication in distributed heterogeneous systems," *Journal of Parallel and Distributed Computing*, Vol. 63, 2003, pp. 251-263.
6. S. Ranka, R. V. Shankar, and K. A. Alsabti, "Many-to-many personalized communication with bounded traffic," in *Proceedings of Symposium on the Frontiers of Massively Parallel Computation*, 1995, pp. 20-27.
7. W. Liu, C. L. Wang, and V. K. Prasanna, "Portable and scalable algorithm for irregular all-to-all communication," *Journal of Parallel and Distributed Computing*, Vol. 62, 2002, pp. 1493-1526.
8. T. S. Hsu, J. C. Lee, D. R. Lopez, and W. A. Royce, "Task allocation on a network of processors," *IEEE Transactions on Computers*, Vol. 49, 2000, pp. 1339-1353.
9. W. M. Lin and W. Xie, "Load-skewing task assignment to minimize communication conflicts on network of workstations," *Parallel Computing*, Vol. 26, 2000, pp. 179-197.
10. M. Guo and Y. Pan, "Improving communication scheduling for array redistribution," *Journal of Parallel and Distributed Computing*, Vol. 65, 2005, pp. 553-563.
11. V. Strumpfen and T. L. Casavant, "Exploiting communication latency hiding for parallel network computing: model and analysis," in *Proceedings of International Conference on Parallel and Distributed Systems*, 1994, pp. 622-627.
12. A. Sohn, J. Ku, Y. Kodama, M. Sato, H. Sakane, H. Yamana, S. Sakai, and Y. Yamaguchi, "Identifying the capability of overlapping computation with communication," in *Proceedings of ACM/IEEE Conference Parallel Architectures and Compilation Techniques*, 1996, pp. 133-138.
13. S. Kim and A. V. Veidenbaum, "The effect of limited network bandwidth and its utilization by latency hiding techniques in large-scale shared memory systems," in *Proceedings of International Conference on Parallel Architectures and Compilation Techniques*, 1997, pp. 40-51.
14. B. Falsafi and D. A. Wood, "Scheduling communication on an SMP node parallel machine," in *Proceedings of IEEE International Symposium on High Performance Computer Architecture*, 1997, pp. 128-138.
15. C. C. Lin, "A novel message scheduling paradigm for developing algorithms in network computing platform," in *Proceedings of International Conference on Advanced Information Networking and Applications*, 2003, pp. 650-655.
16. C. C. Lin, "Strategies for achieving high performance incremental computing on a network environment," in *Proceedings of International Conference on Advanced Information Networking and Applications*, Vol. I, 2004, pp. 113-118.
17. C. C. Lin, T. S. Hsu, and D. W. Wang, "Bounds on the client-server incremental computing," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, Vol. E89-A, 2006, pp. 1198-1206.
18. A. Srivastav and K. Wolf, "Finding dense subgraphs with semidefinite programming," *Lecture Notes in Computer Science*, No. 1444, 1998, pp. 181-191.



**Cho-Chin Lin (林作俊)** is an associate professor at the Department of Electronic Engineering, National Ilan University. He received his B.S. degree in Computer Science and Information Engineering from National Taiwan University in 1985. He received his Ph.D. in Computer Engineering from University of Southern California in 1995. His research interests include parallel and distributed computing, high performance computing, and task migration.