An Efficient Optimal Task Allocation and Scheduling Algorithm for Cyclic Synchronous Applications

Hee-Jun Park and Byung Kook Kim
Department of Electrical Engineering
Korea Advanced Institute of Science and Technology
373-1 Kusongdong, Yusong-gu, Taejon, 305-701 KOREA
hjpark@rtcl.kaist.ac.kr, bkkim@ee.kaist.ac.kr

Abstract

We present an efficient optimal algorithm that allocates and schedules cyclic synchronous tasks into fully connected processors. We consider applications with cyclic synchronous tasks with heavy communication traffic, which run on multiprocessors with fully connected communication network. We suggest the computing period as the performance measure to minimize the overall computation time. We use individual start policy for task scheduling, and also introduce concepts and characteristics of the local period and the global period. To solve the complicated optimal scheduling problem in an efficient way, we propose a new spatial scheduling technique using scheduling space which represents all possible schedules in multi-dimensional space. By using spatial searching and enhanced branch-and-bound technique, the optimal task allocation and schedule which minimizes the computing period can be found efficiently. Various examples and scheduling results show the efficiency of the proposed algorithm.

1. Introduction

As the number of processing elements increases, fine grained multiprocessors reveal more complicated problems which decrease efficiency of parallel processing such as bus arbitration for bus-based architectures [1], network routing control for message-passing architectures [2], and cache consistency [3].

Recent advances in semiconductor technology and high-performance microprocessors such as deep pipelining execution, scoreboarding, very-long-instruction-word (VLIW), and on-chip parallel processors enable developing small-scale multiprocessing systems composed of very powerful processors which provide many convenient features suitable for implementing parallel computers. For example, TMS320C40 [4] from Texas Instruments and ADSP-21060 [5] from Analog Devices have many high-speed communication ports and their compilers have special parallel processing libraries. For on-chip parallel processors, a single TMS320C80 [6] contains five powerful, fully programmable processors and is capable of over two billion operations per second (BOPS). These examples show that small-scale multiprocessing systems or on-chip parallel processors became off-the-shelf and available for large-scale applications that had been executed by massively parallel computers. Hence, development of task allocation and scheduling techniques suitable for small-scale multiprocessing systems or on-chip parallel processors became more indispensable.

Task scheduling methods are typically classified into several subcategories as follows. Static scheduling [9][10][11] balances the workload at compile time with a predictable environment, while dynamic scheduling [7][8][12] performs scheduling techniques concurrently at runtime which applies to unpredictable environment. Since the execution time and communication time of most computation intensive applications -- like simulation and plant control -- are accurately measurable on digital signal processors which have predictable architectures, the tasks can be allocated and scheduled off-line, which reduces run-time overhead.

The inter-processor communication is also one of important components since communication delays decrease the overall performance of parallel processing, especially when a lot of data should be transferred among tasks. Some papers take account of communication behaviors. Veltman [15] defined a model that allows communication delays between precedence-related tasks, and proposed a classification of various submodels. More accurate models was proposed in [13] that handle sequential I/O and program execution as well as parallel I/O and program execution within a processor. However, the algorithm did not consider the contention problem on communication links of multiprocessor system and thus may produce an unrealistic schedule. Krishnan presented a modified version of the algorithm which considers the contention problem [14].

In this paper, we propose an efficient and systematic algorithm that provides the optimal allocation and schedule for cyclic synchronous tasks with heavy inter-task communications on fully connected network, which

0-7695-0306-3/99 $10.00 © 1999 IEEE

can be easily implemented in small-scale multiprocessing systems or on-chip parallel processors.

Most of task allocation and scheduling algorithms define the cost function as the sum of communication time and execution finish time of applications. We propose the *computing period* of overall systems as scheduling performance measure, which is suitable to minimize the overall repetitive computation time. The previous researches like [11], [13], and [14] assumed that all processors start execution at the same time. However, some tasks can start earlier than others if all input data are already available. We use such an *individual start* method to decrease the computing period.

This paper is organized as follows: Section 2 explains computational environment of target multiprocessor architectures, describes new execution and communication model for cyclic synchronous tasks, and defines the optimal scheduling problem. Section 3 presents overview of the new scheduling algorithm. The algorithm can be divided into two parts. Task allocating algorithm using branch-and-bound with efficient forward searching is presented in Section 4, and a spatial approach to optimal scheduling is described in Section 5 using the concept of local period and global period. Experimental results that reveal the efficiency of our algorithm are presented in Section 6. Finally, Section 7 summarizes our conclusion.

2. Parallel Processing Models and Problem Statement

In this section, we present a realistic parallel processing model and state our problem.

2.1. Hardware Environment

In fully connected multiprocessors, each processor has communication links by which data can be directly transferred to destinations without any complicated control like routing or bus arbitration. Since all processors are closely connected within one board or one chip, communication sequences are assumed to be completely reliable and predictable.

2.2. Execution Model for Synchronous Applications

Synchronous data flow (SDF) graph is introduced and widely used for describing applications and developing algorithms [16][17][18]. Fig. 1 depicts an example of cyclic SDF graph, and Fig. 2 shows its execution process that we will consider. For example, simulation of fossil power plants can be divided into several tasks, which are synchronously executed and outputs are updated. By investigating this characteristics of application, we can assume that each arc of this SDF graph has unit delay, hence that m-th data consumed by task B will be the $(m-1)$-th data produced by task A when an arc is connected from task A to task B. The process interaction pattern of synchronous applications is described in [19]. We will develop allocation and scheduling techniques for this execution model which is suitable for plant simulation.

By analyzing real non-terminating application model, we can say that *simultaneous start* of processors is not efficient: Tasks which do not receive inputs from outer world but from the previous iteration of the other tasks can start immediately whenever all inputs are available and the processor is idle. We can increase the utilization of processors by adopting *individual start*, and the computing period rather than the completion time can be shortened.

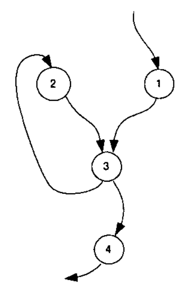

Fig. 1. A cyclic synchronous data flow graph.

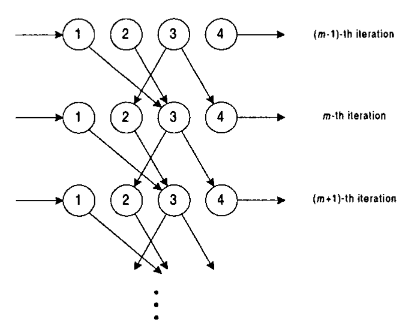

Fig. 2. Delayed execution model of the cyclic SDF graph.

Fig. 3 depicts two scheduling examples of the SDF graph in Fig. 1, which shows improvement of the computing period by individual start. If tasks 1, 3, and 4 are allocated to processor 1 and task 2 is allocated to processor 2. The simultaneous start in Fig. 3(a) is scheduled to start first tasks of two processors at the same time t_0 while Fig. 3(b) shows individual start that the first two tasks start at different time, t_1 and t_2. Let G_1 and G_2 denote the computing period in each case. G_1 is same as total processing time or completion time, which was defined as the cost function in [13] and [14]. However, as we can see in Fig. 3(b), individual start allows overlapping different iterations at the same time, and computing period can be shortened to G_2 by individual start. We adopt this individual start policy and present a method that find the best start-time differences between processors, which minimizes the computing period.

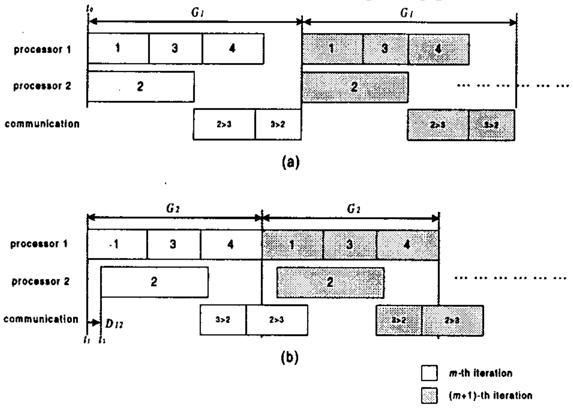

Fig. 3. Computing period. (a) simultaneous start; (b) individual start.

2.3. The Optimal Scheduling Problem

The SDF graph of an application is defined by a set of n_t tasks $T = \{ T_i \}$, $i = 1, 2, \cdots, n_t$ with execution time E_i for task i, and communication costs $M = \{ M_{i,j} \}$, $i = 1, 2, \cdots, n_t$, $j = 1, 2, \cdots, n_t$, $i \neq j$ where $M_{i,j}$ denotes the communication time from task i to task j in the case that they are allocated into different processors. A schedule with n_p processors is uniquely defined by execution sequence of tasks S_k and individual start time t_k on each processor k, $k = 1, \cdots, n_p$.

For example, $S_3 = \{ 1\ 5\ 3\ 4\ 7 \}$ implies that processor 3 executes tasks in the order of $T_1 \rightarrow T_5 \rightarrow T_3 \rightarrow T_4 \rightarrow T_7$. Let $D_{k,l}$ denote the start-time difference of processor k and processor l, i.e.,

$$D_{k,l} = -D_{l,k} = t_l - t_k. \tag{1}$$

S_k and $D_{k,l}$ for $1 \leq k, l \leq n_p$ can be represented by a matrix form such as

$$S = \{ S_1, S_2, S_3, \cdots, S_{n_p} \} \tag{2}$$

$$D = \begin{bmatrix} 0 & D_{1,2} & \cdots & D_{1,n_p} \\ D_{2,1} & 0 & \cdots & D_{2,n_p} \\ \vdots & \vdots & \ddots & \vdots \\ D_{n_p,1} & D_{n_p,2} & \cdots & 0 \end{bmatrix}. \tag{3}$$

Let $L_{k,l}(S, D_{k,l})$ denote the *local period* of processors k and l in the case that we only consider tasks of processors k and l, and communications between them. Let $G(S, D)$ be the *global period* representing the computing period of the overall parallel processing system. The global period is determined by the longest local period, since the parallel processing system should synchronously execute assigned tasks. Hence, the global period becomes a function of S and D such that

$$G(S, D) =$$

$$\max \begin{bmatrix} L_{1,2}(S, D_{1,2}) & L_{1,3}(S, D_{1,3}) & \cdots & L_{1,n_p}(S, D_{1,n_p}) \\ 0 & L_{2,3}(S, D_{2,3}) & \cdots & L_{2,n_p}(S, D_{2,n_p}) \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & L_{n_p-1,n_p}(S, D_{n_p-1,n_p}) \end{bmatrix}. \tag{4}$$

The scheduling problem is to find S and D which minimizes the global period satisfying execution constraints. The optimal scheduling problem can be described as follows.

Problem :

Given n_t tasks with E_i and $M_{i,j}$ on n_p processors, find the minimum global period $G(S, D)$ with a feasible schedule described by execution sequence S and start-time difference D.

3. Overview of the Algorithm

The optimal scheduling is a complicated optimization problem composed of both integer variable set S and continuous variable matrix D. We suggest an algorithm with three phases.

Phase 1. Task allocation iteration with branch-and-bound technique.

Phase 2. Check lower bound of allocation to skip Phase 3.

Phase 3. Task scheduling to find the optimal global period.

The first phase allocates tasks into processors using branch-and-bound technique and searches for the better execution sequence S, which will be explained in Section 4.1. The second phase checks the lower bound of child branches at each node by examining feasibility using a simple integer programming and decides whether the branch will be fathomed or not as explained in Section 4.2. The third phase computes the local periods and the global period, and then use a spatial searching technique to find the best start-time differences D in the given task allocation. This phase will be explained in Section 5.

4. Allocating and Sequencing Tasks

We utilized a branch-and-bound method [20][22] to find the optimal S^*, and developed a separated tree structure for allocating and sequencing tasks. The allocating and sequencing tree structures in [13] and [14] have many redundant nodes that imply exactly the same solution, so that the inefficient tree structures slow down searching speed. To resolve this problem, we separate the tree into two parts – allocating tree and sequencing tree. The overall tree structure is shown in Fig. 4.

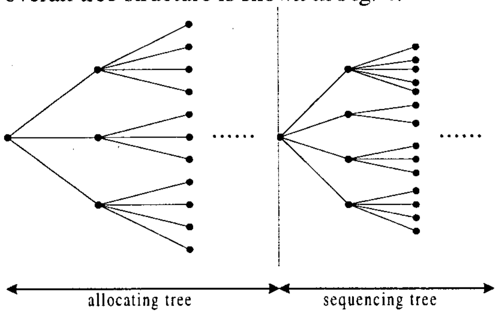

Fig. 4. Allocating and sequencing tree.

4.1. Allocating Tree

Let σ_j denote the total execution time of tasks allocated to processor j such that

$$\sigma_j = \sum_{i=1}^{n_t} \delta_{i,j} E_i \tag{5}$$

where $\delta_{i,j} = 1$ if task i is allocated to processor j; otherwise, $\delta_{i,j} = 0$. Assume that there are f tasks T_1, T_2, $\cdots$, T_f which are not allocated yet, and then we can know the possibility of scheduling from the following integer programming problem:

$$\text{Minimize} \sum_{i=1}^{f} \sum_{j=1}^{n_p} \delta_{i,j} \tag{6}$$

subject to

$$\begin{aligned}
&\delta_{1,1} + \cdots + \delta_{1,n_p} \geq 1 \quad && E_1\delta_{1,1} + \cdots + E_f\delta_{f,1} + \sigma_1 \leq G_w^* \\
&\delta_{2,1} + \cdots + \delta_{2,n_p} \geq 1 \quad && E_1\delta_{1,2} + \cdots + E_f\delta_{f,2} + \sigma_2 \leq G_w^* \\
&\quad\vdots && \quad\vdots \\
&\delta_{f,1} + \cdots + \delta_{f,n_p} \geq 1 \quad && E_1\delta_{1,n_p} + \cdots + E_f\delta_{f,n_p} + \sigma_{n_p} \leq G_w^* .
\end{aligned}$$

where G_w^* denotes the minimum global period in the current stage w of the searching process. The first f constraints represent that all tasks should be allocated to processors. The next n_p constraints represent that execution time of each processor should be smaller than G_w^*. If there is no feasible solution for the problem, all allocations of child nodes can not be scheduled below the current global period G_w^* in any case, hence all child nodes are bounded.

4.2. Sequencing Tree

For a given task allocation at a leaf node of the allocating tree, sequencing process begins with a sequencing tree. In the tree, branching represents that task execution sequence of a processor is determined. For instance, there exist four-time branching from the root of the sequencing tree (leaf node of the allocating tree) to the leaf node of the sequencing tree in case of four processors.

5. The Spatial Approach to the Optimal Scheduling

The third phase of our algorithm which finds the optimal D for a given task allocation S will be explained.

5.1. Local Period

The local period $L_{k,l}$ for processor pair (k,l) and communication link k-l can be defined by execution time of each processor and communication delay as

$$\begin{aligned}
L_{k,l}(S, D_{k,l}) &= \max[\textit{Exec. lower bound},\ \textit{Comm. lower bound}] \\
&= \max[L_{k,l}^e(S),\ L_{k,l}^c(S, D_{k,l})].
\end{aligned} \tag{7}$$

$L_{k,l}^e(S)$ is not a function of $D_{k,l}$ and is given by

$$L_{k,l}^e(S) = \max[\sum_{i=1}^{n_t} \delta_{i,k} E_i,\ \sum_{i=1}^{n_t} \delta_{i,l} E_i\ ]. \tag{8}$$

If all communication delays are completely absorbed into execution time for all $-\infty < D_{k,l} < \infty$, then $L_{k,l}(S, D_{k,l})$ becomes $L_{k,l}^e(S)$ and remains a constant value. However, in many cases, communication delays are not negligible and contentions occur, and hence $L_{k,l}(S, D_{k,l})$ becomes to be determined by $L_{k,l}^c(S, D_{k,l})$. $L_{k,l}^c(S, D_{k,l})$ can be calculated by defining ready, scheduled, completed, and due times as well as scheduling communications. Hence, we assume that a communication line is like a FIFO buffer, and we perform FIFO scheduling. Each communication has a ready time, at which the output data is generated from its source task and become available for transfer. A due time is a point, at which the destination task of the communication starts, so that the communication should be completed beforehand. If ready times and due times of all communications are given, we can schedule communications and define scheduled times and completed times with FIFO scheduling.

Let T_k and T_l be the tasks that are first executed on processor k and processor l respectively. Then, $D_{k,l} = start_{l,m} - start_{k,m}$ where $start_{k,m}$ denotes start time of task k on m-th iteration, and $end_{k,m}$ denotes end time of task k on m-th iteration. Let $C_1, \cdots, C_{n_c}$ be all communications on link k-l. Let α_i denote the source task number of C_i and β_i denote the destination task number of C_i. Then, ready time $r_{i,m}$ and due time $d_{i,m}$ for C_i on m-th iteration are given by

$$r_{i,m} = end_{\alpha_i,m} \tag{9}$$

$$d_{i,m} = start_{\beta_i,m+1} = start_{\beta_i,m} + L_{init} \tag{10}$$

for $i = 1, 2, \cdots, n_c$. Initial local period L_{init} is defined as a sufficiently large value compared to $L^e_{k,l}(S)$. Given ready time and due time, we perform FIFO scheduling for $C_1, \cdots, C_{n_c}$. By definition of scheduled time $s_{i,m}$ and completed time $e_{i,m}$,

$$e_{i,m} = s_{i,m} + M_{\alpha_i,\beta_i}. \tag{11}$$

where M_{α_i,β_i} denotes the communication time from task α_i to task β_i.

Fig. 5 shows an example of ready, scheduled, completed, and due times for a communication.

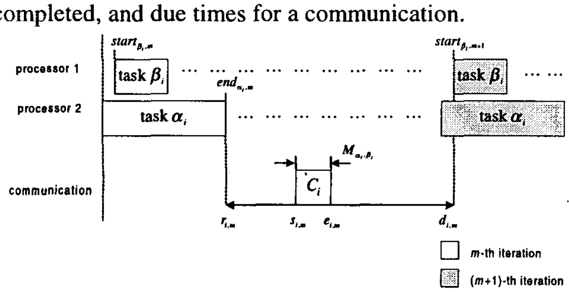

Fig. 5. Ready, scheduled, completed, and due time.

$L^c_{k,l}(S, D_{k,l})$ can be calculated by subtracting the minimum difference of due times and completed times from L_{init} as follows:

$$L^c_{k,l}(S, D_{k,l}) = L_{init} - \min[d_{1,m} - e_{1,m}, \cdots, d_{n_c,m} - e_{n_c,m}] \tag{12}$$

We define a portion in which the order of ready ~ ready and ready ~ completed does not vary. $L^c_{k,l}(S, D_{k,l})$ for $-\infty < D_{k,l} < \infty$ is easily obtained by dividing $D_{k,l}$ into several portions like Fig. 6 and considering the characteristics of the function. Let $C_1, \cdots, C_{n_k}$ denote communications that are transferred from processor k to processor l on link k-l and $C_{n_k+1}, \cdots, C_{n_c}$ denotes the communications that are transferred vice versa, where $n_c = n_k + n_l$. Divided portions can be obtained by an algorithm, which is shown in Fig. 7 where function $p(x)$ is defined as

$$p(x) = \begin{cases} 0 & \text{for } x \le 0 \\ x & \text{otherwise} \end{cases}. \tag{13}$$

This algorithm starts from initial D' and then checks portion boundaries on which order of ready ~ ready and ready ~ completed change as Table 1 where $1 \le i \le n_k$, $n_k + 1 \le j \le n_c$. The number of finite portions $n_b + 1$ and the boundaries of portions $D_{k,l}[0], \cdots, D_{k,l}[n_b - 1]$ are obtained by this algorithm.

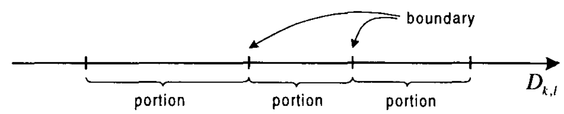

Fig. 6. Portion and boundary on $D_{k,l}$.

Table 1. Three types of boundary on $D_{k,l}$.

Boundary type	Order change at the boundary
A	$r_{i,m} - r_{j,m} > 0 \;\leftrightarrow\; r_{i,m} - r_{j,m} < 0$
B	$r_{i,m} - e_{j,m} > 0 \;\leftrightarrow\; r_{i,m} - e_{j,m} < 0$
C	$e_{i,m} - r_{j,m} > 0 \;\leftrightarrow\; e_{i,m} - r_{j,m} < 0$

1. Set the initial D' which is satisfying
   $\max[e_{n_k+1,m}, \cdots, e_{n_c,m}] < \min[r_{1,m}, \cdots, r_{n_k,m}]$.
2. $n_b \leftarrow 0$
3. do while ($\min[r_{n_k+1,m}, \cdots, r_{n_c,m}] \le \max[e_{1,m}, \cdots, e_{n_k,m}]$)
   Run FIFO scheduling for $D' < D_{k,l} < D' + \varepsilon$ where ε is positive and $\varepsilon \to 0$
   $\Delta D_{i,j} \leftarrow \min[\,p(r_{i,m} - r_{j,m}),\ p(r_{i,m} - e_{j,m}),\ p(e_{i,m} - r_{j,m})\,]$
   $\Delta D \leftarrow \min_{\forall i,j} \Delta D_{i,j}$, where $i = 1, \cdots, n_k$, $j = n_k + 1, \cdots, n_c$
   $D_{k,l}[n_b] \leftarrow D' + \Delta D$
   $D' \leftarrow D' + \Delta D$
   $n_b{+}{+}$
   end do

Fig. 7. The portion dividing algorithm.

Theorem 1: $L^c_{k,l}(S, D_{k,l})$ can be represented as a form of *constant* or *constant* $+ D_{k,l}$ or *constant* $- D_{k,l}$ and remains the same form in a portion.

Theorem 2: $L^c_{k,l}(S, D_{k,l})$ has no local minima for $-\infty < D_{k,l} < \infty$.

(Proofs omitted due to space limitation.)

From Theorems 1 and 2, and Eq. (7), it is shown that the local period, $L_{k,l}(S, D_{k,l})$ is piecewise linear and has no local minima for $D_{k,l}$.

5.2. Global Period and Optimization Methodology in Scheduling Space

If all local periods of processor-pairs are found, the global period is determined by the longest local period because applications allocated into processors are synchronously executed. From $D_{k,l} = D_{h,l} - D_{h,k}$, global period of Eq. (4) becomes function of $n_p - 1$ independent variables $D_{1,2}, D_{1,3}, \cdots, D_{1,n_p}$ such that

$$G(S,D) = \max \begin{bmatrix} L_{1,2}(S,D_{1,2}) & L_{1,3}(S,D_{1,3}) & \cdots & L_{1,n_p}(S,D_{1,n_p}) \\ 0 & L_{2,3}(S,D_{1,3}-D_{1,2}) & \cdots & L_{2,n_p}(S,D_{1,n_p}-D_{1,2}) \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & L_{n_p-1,n_p}(S,D_{1,n_p}-D_{1,n_p-1}) \end{bmatrix} \tag{14}$$

Let a *scheduling space* denotes the $n_p - 1$ dimension space whose axes are $D_{1,2}, D_{1,3}, \cdots, D_{1,n_p}$. Then, a schedule can be mapped to a point in the scheduling space, and this mapping is one-to-one. Hence, scheduling space represents all feasible schedules for a given task allocation. The scheduling space can be divided into two areas with respect to the global period, G'. The first is called *schedulable area* that satisfies $G(S,D) \le G'$, so that it is possible to make the schedule whose global period is not larger than G' in the schedulable area. The other is the exterior of schedulable area where $G(S,D) > G'$. Scheduling space has following two properties.

- More than two separated schedulable area can not exist since $G(S,D)$ has no local minima for D.
- There is no schedule whose global period is not larger than G', if and only if scheduling area does not exist for given G'.

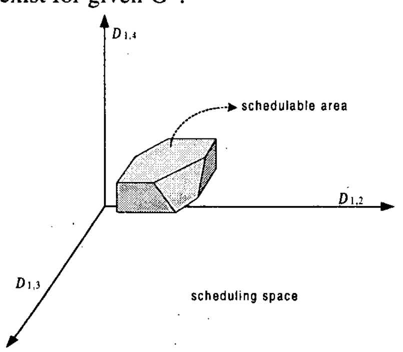

Fig. 8. Scheduling space and schedulable area.

Schedulable area can not be defined if there is a link whose minimum local period is larger than G'. Otherwise, schedulable area can be defined by $n_p(n_p-1)/2$ inequalities:

$$\begin{matrix} L_{1,2}(S,D_{1,2}) \le G' & \cdots & L_{1,n_p}(S,D_{1,n_p}) \le G' \\ \vdots & \ddots & \vdots \\ 0 & 0 & L_{n_p-1,n_p}(S,D_{1,n_p}-D_{1,n_p-1}) \le G' \end{matrix} \tag{15}$$

Since local periods are piecewise-linear and has no local minima for $D_{k,l}$, local periods are represented as a table of $D_{k,l}$ and $L_{k,l}(S,D_{k,l})$. Domains of $D_{1,2}, D_{1,3}, \cdots, D_{1,n_p}$ that satisfy Eq. (15) can be easily found such that

$$\begin{matrix} b^l_{1,2} \le D_{1,2} \le b^r_{1,2} & \cdots & b^l_{1,n_p} \le D_{1,n_p} \le b^r_{1,n_p} \\ \vdots & \ddots & 0 \\ 0 & \cdots & b^l_{n_p-1,n_p} \le D_{1,n_p} - D_{1,n_p-1} \le b^r_{n_p-1,n_p} \end{matrix} . \tag{16}$$

Substitute non-negative variables $D_{0,1}, D_{0,2}, \cdots, D_{0,n_p}$ for $D_{1,2}, D_{1,3}, \cdots, D_{1,n_p}$, then we can get

$$\begin{matrix} b^l_{1,2} \le D_{0,2} - D_{0,1} \le b^r_{1,2} & \cdots & b^l_{1,n_p} \le D_{0,n_p} - D_{0,1} \le b^r_{1,n_p} \\ \vdots & \ddots & \vdots \\ 0 & 0 & b^l_{n_p-1,n_p} \le D_{0,n_p} - D_{0,n_p-1} \le b^r_{n_p-1,n_p} \end{matrix} . \tag{17}$$

Eq. (17) can be represented with the standard linear programming form.

$$\begin{gathered} \mathbf{Ax} \ge \mathbf{b} \\ \mathbf{x} \ge \mathbf{0} \end{gathered} \tag{18}$$

where

$$\mathbf{A} = \begin{bmatrix} -1 & 1 & 0 & 0 & \cdots \\ 1 & -1 & 0 & 0 & \cdots \\ -1 & 0 & 1 & 0 & \cdots \\ 1 & 0 & -1 & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

$$\mathbf{x} = [D_{0,1}, D_{0,2}, D_{0,3}, \cdots, D_{0,n_p}]'$$

$$\mathbf{b} = [b^l_{1,2}, -b^r_{1,2}, b^l_{1,3}, -b^r_{1,3}, b^l_{1,4}, -b^r_{1,4} \cdots]'$$

We can know whether scheduling area exists or not, by appending artificial variables and solving artificial optimization problem of linear programming [21]. The minimum G', which has schedulable area, becomes the optimal global period and all points in the schedulable area are optimal schedules. The optimal schedules are found by a simple binary search technique. Let G_{low} and

G_{high} denotes lower and upper bounds of this binary searching. They are initially given as $G_{low} = \max_{\forall k,\forall l}\left[\min_{-\infty<D_{k,l}<\infty} L_{k,l}(S, D_{k,l})\right]$ and $G_{high} = G(S,0)$. Let G_{opt} denote the global period associated with the best feasible solution in a particular stage of the algorithm, and then the boundary of G_{opt} becomes narrower and narrower while running this algorithm until the optimal schedules are found. The description of this algorithm is shown in Fig. 9.

1. $G_{low} \leftarrow \max_{\forall k,\forall l}\left[\min_{-\infty<D_{k,l}<\infty} L_{k,l}(S, D_{k,l})\right]$, $G_{high} \leftarrow G(S,0)$
2. do while ($G_{high} - G_{low} \geq 1$)
   - $G' \leftarrow (G_{high} - G_{low}) / 2$
   - if scheduling is feasible for G'
     - $G_{high} \leftarrow G'$
   - else
     - $G_{low} \leftarrow G'$
   - end do
3. if ($\lceil G_{low} \rceil = \lceil G_{high} \rceil$) then
   - $G_{opt} \leftarrow \lceil G_{high} \rceil$

   else
   - if scheduling is feasible for $\lceil G_{low} \rceil$ then
     - $G_{opt} \leftarrow \lceil G_{low} \rceil$
   - else
     - $G_{opt} \leftarrow \lceil G_{high} \rceil$
   - end if

   end if

Fig. 9. The optimization algorithm in scheduling space.

6. Experimental Results

The task allocating algorithms were programmed in C language on a pentium PC that is frequently used to develop and compile DSP applications. For an example application, a simulation of fossil power generation plants which are introduced by [23] was employed. This simulation contains a lot of floating-point operations like partial differential equations and ordinary differential equations. The application can be divided into tasks which represent parts of power generation plants. Fig. 10 shows the task graph of power generation plants, and detailed information about execution times and communication times are shown in Appedix.

We measured the processing time for finding the optimal allocation. To analyze the algorithm for various communication traffics, we ran the algorithm as to three cases of communication rate – actual speed (Table 3), ten times faster (Table 2), and ten times slower (Table 4). These results indicate that as communication traffic becomes heavier, the algorithm takes more time due to complexity of communication scheduling. However, the maximum processing time in these examples is 38 seconds, which is acceptable since allocation and scheduling is applied off-line.

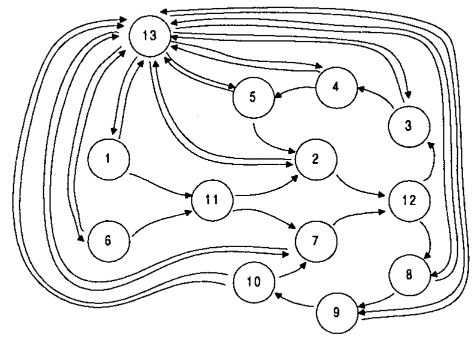

Fig. 10. Task graph of fossil power plants.

Table 2. Experiment results for communication rate x 10 (light communication traffic).

Exp. #	Number of processors	Processing time (seconds)	Number of calculated leaves	Global period
1	2	15	14808	1054
2	3	10	37020	716
3	4	5	54296	532

Table 3. Experiment results for communication rate x 1.

Exp. #	Number of processors	Processing time (seconds)	Number of calculated leaves	Global period
4	2	30	29616	1083
5	3	18	69104	729
6	4	38	345520	550

Table 4. Experiment results for communication rate x 0.1 (heavy communication traffic).

Exp. #	Number of processors	Processing time (seconds)	Number of calculated leaves	Global period
7	2	29	29616	3106
8	3	34	155484	1792
9	4	37	404752	1410

7. Conclusion

New spatial approach to optimal task allocation and scheduling problem is presented with the global period as our performance index. We defined the global period, and pointed out that the global period can be enhanced by individual start policy. To solve such a complicated optimization problem, local periods of communication links are calculated first, and then optimal start times of processors are found using scheduling space and linear programming. The experimental results show that our algorithm is fast enough and practical for real applications.

References

[1] Kai Hwang, *Advanced Computer Architecture*, McGraw-Hill, 1993.

[2] Ronald P. Bianchini, John Paul Shen, "Interprocessor Traffic Scheduling Algorithm for Multiple-Processor Networks", *IEEE trans. Computers*, vol. C-36, no. 4, pp. 396-409, 1987.

[3] David Cullar, Jaswinder Pal Singh, and Anoop Gupta, *Parallel Computer Architecture*, Morgan Kaufmann, 1997.

[4] *TMS320C4x Parallel Processing Development System Technical Reference*, Texas Instruments, 1993.

[5] *ADSP-2106x SHARC User's Manual*, Analog Devices, 1995.

[6] *TMS320C8x System-Level Synopsis*, Texas Instruments, 1995.

[7] Kumar K. Goswami, Murthy Devarakonda, and Ravishankar K. Iyer, "Prediction-Based Dynamic Load-Sharing Heuristics", *IEEE Trans. Parallel and Distributed Systems*, vol. 4, no. 6, pp. 638-648, 1993.

[8] Min-You Wu, "On Runtime Parallel Scheduling for Processor Load Balancing", *IEEE trans. Parallel and Distributed Systems*, vol. 8, no. 2, pp. 173-186, 1997.

[9] Edward Ashford Lee and David G. Messerschmitt, "Static Scheduling of Synchronous Data Flow Programs for Digital Sugnal Processing", *IEEE trans. Computers*, vol. C-36, no. 1, pp. 24-35, 1987.

[10] C. V. Ramamoorthy, K. M. Chandy, and Mario J. Gonzalez, "Optimal Scheduling Strategies in a Multiprocessor System", *IEEE trans. Computers*, vol. C-21, no. 2, pp. 137-146, 1972.

[11] Gilbert C. Sih and Edward A. Lee, "A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures", *IEEE trans. Parallel and Distributed Systems*, vol. 4, no. 2, pp. 175-187, 1993.

[12] Davari, S., and S. K. Dhall, "An On Line Algorithm for Real-Time Tasks Allocation", *Proc. IEEE Real-Time Systems Symp.*, pp. 194-200, IEEE, Los Alamitos, CA, 1986.

[13] Konstantinos Konstantinides, Ronald T. Kaneshiro, and Jon R. Tani, "Task Allocation and Scheduling Models for Multiprocessor Disgital Signal Processing", *IEEE trans. Acoustics, Speech, and Signal Processing*, vol. 38, no. 12, pp. 2151-2161, 1990.

[14] C. S. R. Krishnan, D. Antony Louis Piriyakumar, and C. Siva Ram Murthy, "A Note on Task Allocation and Scheduling Models for Multiprocessor Digital Signal Processing", *IEEE trans. Signal Processing*, vol. 43, no. 3, pp. 802-805, 1995.

[15] B. Veltman and B. J. Lageweg, "Multiprocessor Scheduling with Communication Delays", *Parallel Computing*, vol. 16, pp. 173-182, 1990.

[16] J. B. Dennis, "Data Flow Supercomputers", *Computer*, vol. 13, Nov. 1980.

[17] W. B. Ackerman, "Data Flow Languages", *Computer*, vol. 15, Feb. 1982.

[18] Watson and J. Gurd, "A Practical Data Flow Computer", *Computer*, vol. 15, Feb. 1982.

[19] Nayeem Islam, *Distributed Objects: Methodologies for Customizing Systems Software*, IEEE Computer Society Press, 1996.

[20] Stanley Zionts, *Linear and Integer Programming*, Prentice-Hall, 1974.

[21] David G. Luenberger, *Linear and Nonlinear Programming*, Addison-Wesley, 1984.

[22] R. S. Garfinkel and G. L. Nemhauser, *Integer Programming*, Wiley, 1972.

[23] W. Ordys, A. W. Pike, M. A. Johnson, R. M. Katebi, and M. J. Grimble, *Modeling and Simulation of Power Generation Plants*, Springer-Verlag, 1994.

Appendix

Information about the task graph of power generation plants:

Task #	Task Name	Task #	Task Name
1	Gas turbine-1	8	Steam turbine-2
2	Boiler-1	9	Condenser-2
3	Steam turbine-1	10	Feedwater-2
4	Condenser-1	11	Gas damper
5	Feedwater-1	12	Steam spilt
6	Gas turbine-2	13	Controller
7	Boiler-2		

$$E = \{ E_1, E_2, E_3, \cdots, E_{13} \}$$
$$= \{164, 421, 77, 145, 213, 164, 421, 77, 145, 213, 24, 24, 10\}$$

$M_{1,11} = 38$, $M_{1,13} = 20$, $M_{2,12} = 40$, $M_{2,13} = 20$,

$M_{3,4} = 36$, $M_{3,13} = 16$, $M_{4,5} = 32$, $M_{4,13} = 12$,

$M_{5,2} = 34$, $M_{5,13} = 14$, $M_{6,11} = 38$, $M_{6,13} = 20$,

$M_{7,12} = 40$, $M_{7,13} = 20$, $M_{8,9} = 36$, $M_{8,13} = 16$,

$M_{9,10} = 32$, $M_{9,13} = 12$, $M_{10,7} = 34$, $M_{10,13} = 14$,

$M_{11,2} = 42$, $M_{11,7} = 42$, $M_{12,3} = 26$, $M_{12,8} = 26$,

$M_{13,1} = 16$, $M_{13,2} = 20$, $M_{13,3} = 14$, $M_{13,4} = 12$, $M_{13,5} = 14$,

$M_{13,6} = 16$, $M_{13,7} = 20$, $M_{13,8} = 14$, $M_{13,9} = 12$, $M_{13,10} = 14$

The other communication costs are zero.