NP-Completeness of the Bulk Synchronous Task Scheduling Problem and Its Approximation Algorithm

Noriyuki Fujimoto and Kenichi Hagihara
Graduate School of Engineering Science, Osaka University
Toyonaka, 560-8531, Japan
{fujimoto, hagihara}@ics.es.osaka-u.ac.jp
TEL : +81-6-6850-6598, FAX : +81-6-6850-6599

Abstract

The bulk synchronous task scheduling problem (BSSP) is known as an effective task scheduling problem for distributed-memory machines, but the time complexity of BSSP is unknown. This paper presents a proof of NP-completeness of BSSP even in the case of unit time tasks and positive integer constant communication delays. This paper also gives an approximation algorithm for BSSP in several restricted cases.

1 Introduction

Existing researches on the task scheduling problem for distributed-memory machines (**DMM** for short) simply model DMMs as the parallel machines with large communication delay [11, 10, 3]. In contrast to this, the literatures[5, 6] showed the following things:

- It is essential to task scheduling for DMM to consider the software overhead in communication and message packaging, even if a DMM is equipped with a dedicated communication co-processor per processor.
- Existing task scheduling algorithms would ignore the software overhead and do not consider message packaging.
- For the above reasons, it is hard for existing algorithms to generate schedules which become fast parallel programs on a DMM.

To remedy this situation, the literatures[5, 6] proposed the bulk synchronous task scheduling problem (**BSSP** for short), i.e. the problem of finding a bulk synchronous schedule with small makespan, and also proposed heuristic algorithms for BSSP. However, the time complexity of BSSP is unknown and the above heuristic algorithms has no performance guarantee.

In this paper, we first present a proof of NP-completeness of BSSP even for task graphs such that the execution time of any task is unit and the communication delay of any edge is a positive integer constant. Next, we give a method to convert a general schedule S into a bulk synchronous one with twice the makespan of S for such a task graph and a specified number of processors. Several restricted cases are known where optimal or near-optimal schedules are obtained for the general task scheduling problem. We lead an approximation algorithm with the performance guarantee for BSSP in such cases by combining the method with these existing algorithm.

2 Preliminaries and Notation

A parallel computation is modeled as a task graph[4]. A **task graph** is represented by a weighted DAG (Directed Acyclic Graph) $G = (V, E, \lambda, \tau)$, where V is a set of nodes, E is a set of directed edges, λ is a function from a node to the weight of the node, and τ is a function from a directed edge to the weight of the edge. We write a directed edge from a node u to a node v as (u, v). If $(u, v) \in E$, v (u, resp.) is called a **successor** (**predecessor**, resp.) of u (v, resp.). A node without a successor (predecessor, resp.) is called an **exit node** (**entry node**, resp.). We write the set of all the predecessors of v as $Pred(v)$. A node in a task graph represents a task in the parallel computation. We write a task represented by a node u as T_u. The value $\lambda(u)$ means that the execution time of T_u is $\lambda(u)$ time units. An edge (u, v) means that the computation of T_v needs the result of the computation of T_u. The value $\tau(u, v)$ means that interprocessor communication delay from the processor p which computes T_u to the processor q which computes T_v is at most $\tau(u, v)$ time units if p is not equal to q. If p and

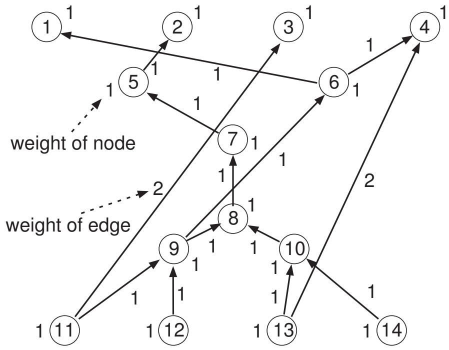

Figure 1. An example of a task graph

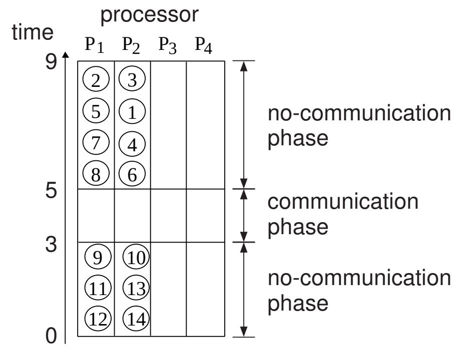

Figure 2. A bulk synchronous schedule

q are identical, no interprocessor communication delay is needed.

Thurimella gave the definition of a schedule in the case that $\lambda(v)$ is equal to a unit time for any v and $\tau(u,v)$ is a constant independent of u and v[13]. For general task graphs, we define a schedule as extension of Thurimella's definition as follows. For a given number p of available processors, a **schedule** S of a task graph $G = (V, E, \tau, \lambda)$ for p is a finite set of triples $\langle v, q, t\rangle$, where $v \in V$, $q(1 \le q \le p)$ is the index of a processor, and t is the **starting time** of task T_v. A triple $\langle v, q, t\rangle \in S$ means that the processor q computes the task T_v between time t and time $t + \lambda(v)$. We call $t+\lambda(v)$ the **completion time** of the task T_v. A schedule which satisfies the following four conditions R1 to R4 is called **valid** (In the following of this paper, we abbreviate a valid schedule as a schedule.):

- R1 For each $v \in V$, there is at least one triple $\langle v, q, t\rangle \in S$.
- R2 There are no two triples $\langle v, q, t\rangle, \langle v', q, t'\rangle \in S$ with $t \le t' \le t + \lambda(v)$.
- R3 If $(u, v) \in E$ and $\langle v, q, t\rangle \in S$, then there exists a triple $\langle u, q', t'\rangle \in S$ either with $t' \le t - \lambda(u)$ and $q = q'$, or with $t' \le t - \lambda(u) - \tau(u, v)$ and $q \ne q'$.
- R4 If $(u, v) \in E$ and $\langle u, q, t\rangle \in S$, then there exists a triple $\langle v, q', t'\rangle \in S$ either with $t' \ge t + \lambda(u)$ and $q = q'$, or with $t' \ge t + \lambda(u) + \tau(u, v)$ and $q \ne q'$.

Informally, the above rules can be stated as follows. The rule $R1$ enforces each task T_v to be executed at least once. The rule $R2$ says that a processor can execute at most one task at any given time. The rule $R3$ states that any task must receive the required data (if exist) before its starting time. The rule $R4$ prohibits the existence of a task such that its result is not used for any task to be computed. The **makespan** of S is $\max\{t+\lambda(v)|\langle v, q, t\rangle \in S\}$. An **optimal schedule** is a schedule with the smallest makespan among all the schedules.

A schedule within a factor of α of optimal is called an **α-optimal schedule**.
A **ρ-approximation algorithm** is a polynomial-time algorithm that always finds a ρ-optimal schedule; the value ρ is referred to as the **performance guarantee** of the algorithm[9].

3 Review of a Bulk Synchronous Schedule

As shown in Fig. 2, a bulk synchronous schedule is a schedule such that no-communication phases and communication phases appear alternately (In a general case, no-communication phases and communication phases appear repeatedly). Informally, a no-communication phase is a set of task instances in a time interval such that the corresponding program executes computations only. A communication phase is a time interval such that the corresponding program executes communications only. A bulk synchronous schedule is similar to BSP (Bulk Synchronous Parallel) computation proposed by Valiant [15] in that local computations are separated from global communications. A no-communication phase corresponds to a super step of BSP computation. In the following, we first define a no-communication phase and a communication phase. Then, we define a bulk synchronous schedule using them.

Let S be a schedule of a task graph $G = (V, E, \lambda, \tau)$ for a number p of available processors. We define the following notation: For S, t_1, and t_2 with $t_1 < t_2$,

$$S[t_1, t_2] = \{\langle v, q, t\rangle \in S | t_1 \le t \le t_2 - \lambda(v)\}$$

Notation $S[t_1, t_2]$ represents the set of all the triples such that both the starting times and the completion times are between t_1 and t_2. A set $S[t_1, t_2] \subseteq S$ of triples is called a **no-communication phase** of S iff the following condition holds.

- C1 If $(u, v) \in E$ and $\langle v, q, t\rangle \in S[t_1, t_2]$, then there exists a triple $\langle u, q', t' \rangle \in S$ either with $t' \le t - \lambda(u)$ and $q = q'$, or with $t' \le t_1 - \lambda(u) - \tau(u,v)$ and $q \ne q'$.

The condition C1 means that each processor needs no interprocessor communication between task instances in $S[t_1, t_2]$ since all the needed results of tasks are either computed by itself or received from some processor before t_1.

Let $S[t_1, t_2]$ be a no-communication phase. Let t_3 be $\min\{t | \langle u, q, t \rangle \in (S - S[0, t_2])\}$. Assume that a no-communication phase $S[t_3, t_4]$ exists for some t_4. We say that $S[t_1, t_2]$ and $S[t_3, t_4]$ are **consecutive no-communication phases**. We intend that in the execution of the corresponding program each processor sends the results, which are computed in $S[t_1, t_2]$ and are required in $S[t_3, t_4]$, as packaged messages at t_2 and receives all the needed results in $S[t_3, t_4]$ as packaged messages at t_3. A **communication phase** between consecutive no-communication phases is the time interval where each processor executes communications only. To reflect such program's behavior in the time interval on the model between consecutive no-communication phases, we assume that the result of $\langle u, q, t \rangle \in S[t_1, t_2]$ is sent at t_2 even in case of $t + \lambda(u) < t_2$ although the model assumes that the result is always sent at $t + \lambda(u)$. Let $Comm(S, t_1, t_2, t_3, t_4)$ be $\{(u, v) | (u, v) \in E, \langle u, q, t \rangle \in S[t_1, t_2], \langle v, q', t' \rangle \in S[t_3, t_4], \langle u, q', t'' \rangle \notin S, q \ne q', t'' \le t' - \lambda(u)\}$. A set $Comm(S, t_1, t_2, t_3, t_4)$ of edges corresponds to the set of all the interprocessor communications between task instances in $S[t_1, t_2]$ and task instances in $S[t_3, t_4]$. We define the following notation: For $C \subseteq E$,

$$\tau_{suff}(C) = \begin{cases} 0 & if\ C = \emptyset \\ \max\{\tau(u,v) | (u,v) \in C\} & otherwise \end{cases}$$

Consider simultaneous sendings of all the results in C. The value $\tau_{suff}(C)$ represents the elapsed time on the model till all the results are available to any processor. So, the value $\tau_{suff}(Comm(S, t_1, t_2, t_3, t_4))$ represents the minimum communication delay on the model between the consecutive no-communication phases.

We say S is a **bulk synchronous schedule** iff S can be partitioned into a sequence of no-communication phases $\langle S[st_1, ct_1], S[st_2, ct_2], \cdots, S[st_m, ct_m] \rangle$ ($m \ge 1$) which satisfies the following condition C2.

C2 For any $i (1 \le i < m)$, $ct_i + \tau_{suff}(Comm(S, st_i, ct_i, st_{i+1}, ct_{i+1})) \le st_{i+1}$

Fig. 2 shows an example of a bulk synchronous schedule $\langle S[0,3], S[5,9] \rangle$ of the task graph in Fig. 1 for four processors. The set $Comm(S, 0, 3, 5, 9)$ of edges is $\{(9,6), (10,8), (11,3)\}$. The edge with maximum weight of all the edges in $Comm(S, 0, 3, 5, 9)$ is $(11, 3)$. So, the weight of the edge $(11, 3)$ decides that $\tau_{suff}(Comm(S, 0, 3, 5, 9))$ is two.

4 A Proof of NP-completeness of BSSP

We prove that BSSP is NP-complete even in the case of unit time tasks and positive integer constant communication delays (**BSSP**c for short) by reducing the unit time precedence constrained scheduling problem[14] (**SP** for short) to BSSPc. These problems are:

BSSPc Instance: A task graph G_{1c} such that all the weight of nodes are unit time and all the weight of edges are positive integer c, a number p of available processors, a deadline D (positive integer);
Question: Is there a bulk synchronous schedule S_{BSP} whose makespan is less than or equal to D ?

SP Instance: A task graph G_{10} such that all the weight of nodes are unit time and all the weight of edges are zero, a number p of available processors, a deadline D (positive integer);
Question: Is there a schedule S whose makespan is less than or equal to D ?

Theorem 1 *BSSPc is NP-complete.*

Proof Let T be a 3D boolean array such that $\langle v, p, t \rangle \in S \Leftrightarrow T[v, p, t] = true$. Let n be the number of all the nodes in a given task graph. We can assume $1 \le p \le n$ and $1 \le v \le n$ without loss of generality. If $D \ge n$, then the answer for BSSPc (G, p, D) is yes since the makespan of a trivial schedule for one processor (Compute each task in topological order. Note that the trivial schedule is a bulk synchronous schedule with a no-communication phase only) is n. Therefore, we can assume $0 \le t < n$ without loss of generality. There are 2^{n^3} cases to decide T. BSSPc is in NP, since we can verify in polynomial time whether each case is valid as a bulk synchronous schedule and its makespan is less than or equal to a given deadline.

Let (G_{10}, p, D) be an instance of SP. For (G_{10}, p, D), form an instance (G'_{1c}, p', D') of BSSPc as follows:

- $D' = D + (D-1)c$
- $p' = p + 1 + D'$
- Let G'_{1c} be a task graph such that the following nodes and edges are added to G_{10} and all the weight of nodes (edges, resp.) are unit time (c, resp. where c is an arbitrary positive integer) (See Fig. 3). Note that each additional node is not connected to any node in G_{10}.
  - additional nodes
    $v_i (1 \le i \le D)$ and $u_{i,j} (1 \le i < D, 1 \le j \le D')$
  - additional edges
    (v_i, v_{i+1}) $(1 \le i < D)$ and $(u_{i,j}, v_{i+1})$ $(1 \le i < D, 1 \le j \le D')$

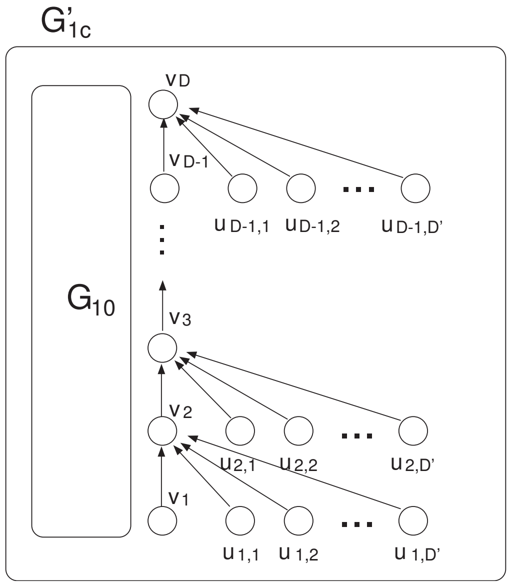

Figure 3. An input task graph for reduction

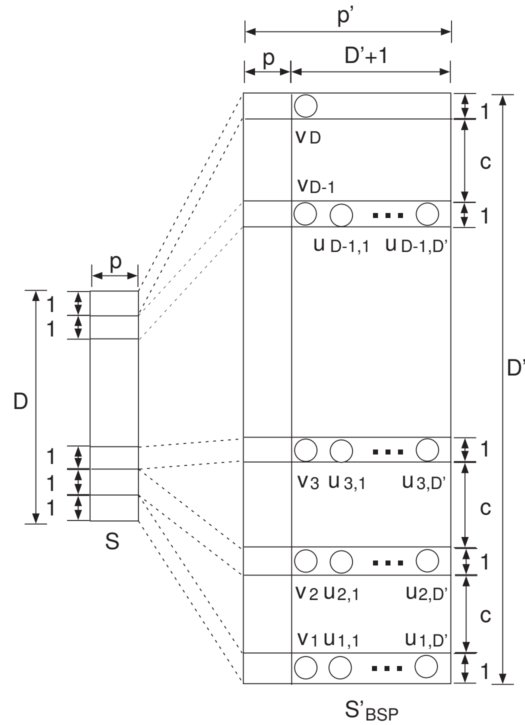

Figure 4. An yes instance to an yes instance correspondence

If (G_{10}, p, D) is an "yes" instance of SP, then let S be a schedule for (G_{10}, p, D). A set $S'_{BSP} = \{\langle v, q, t(c+1)\rangle | \langle v, q, t\rangle \in S\} \cup \{\langle v_i, p+1, (i-1)(c+1)\rangle | 1 \le i \le D\} \cup \{\langle u_{i,j}, p+1+j, (i-1)(c+1)\rangle | 1 \le i < D, 1 \le j \le D'\}$ of triples is a bulk synchronous schedule for (G'_{1c}, p', D') with D no-communication phases and $(D-1)$ communication phases (See Fig. 4).

Conversely, if (G'_{1c}, p', D') is an "yes" instance of BSSPc, then let S'_{BSP} be a schedule for (G'_{1c}, p', D'). Any $v_{i+1}(1 \le i < D)$ cannot construct a no-communication phase with all of $\{u_{i,j} | 1 \le j \le D'\}$ because the computation time $(D'+1)$ of the no-communication phase is greater than the given deadline D'. That is, any $v_{i+1}(1 \le i < D)$ must communicate with at least one of $\{u_{i,j} | 1 \le j \le D'\}$. This means that S'_{BSP} includes at least $(D-1)$ communication phases. On the other hand, S'_{BSP} cannot include more than $(D-1)$ communication phases because $(D'-D)/c = D-1$. Therefore, S'_{BSP} includes just $(D-1)$ communication phases. Consequently, S'_{BSP} includes just $(D-1)$ communication phases and just D no-communication phases with one unit long. Hence, S'_{BSP} would become a schedule as shown in Fig. 4. A subset $S = \{\langle v, q, t\rangle | \langle v, q, t(c+1)\rangle \in S'_{BSP}, 1 \le q \le p\}$ of S'_{BSP} is a schedule for (G_{10}, p, D).

5 An Approximation Algorithm

5.1 A Conversion Method of a General Schedule into a Bulk Synchronous Schedule

In this section, we assume that the weight of any node is unit time and the weight of any edge is a positive integer. Any edge may have different weight from any other edge.

Let c_{min} (c_{max}, resp.) be the smallest (largest, resp.) weight of all the edges. Let S be an arbitrary general schedule. Consider any c_{min} units long time interval I in S. From the condition R3 in Section 2, there exists no interprocessor communication such that both the sending time and the receiving time are in I. Therefore, for any t, $S[t, t+c_{min}]$ satisfies the conditions C1 in Section 3. This observation gives a conversion method of any general schedule into a bulk synchronous schedule, as described below.

By partitioning S into c_{min} units long time intervals which become no-communication phases and setting up c_{max} units long communication phases between consecutive time intervals, we can generate a bulk synchronous schedule $S' = \{\langle v, q, t + \lfloor t/c_{min} \rfloor (c_{min} + c_{max})\rangle | \langle v, q, t\rangle \in S\}$ with c_{min} units long no-communication phases and c_{max} units long communication

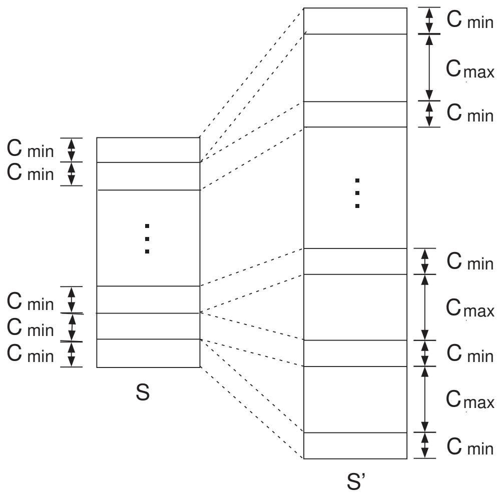

Figure 5. The conversion of a general schedule into a bulk synchronous schedule

phases (See Fig. 5). Clearly, the following lemma holds.

Lemma 1 *The makespan of S' is at most $(1 + c_{max}/c_{min})$ times makespan of S.*

5.2 The Performance Guarantee of the Conversion Method

Consider the general task scheduling problem (**GSP** for short) and BSSP for the same input (a task graph G and a number p of available processors). Let OPT and OPT_{BSSP} be the optimal makespan of GSP and BSSP, respectively. Since the set of all the bulk synchronous schedules for G and p is a subset of the set of all the general schedules for G and p, $OPT \leq OPT_{BSSP}$ holds. Assume that we have a general schedule S whose makespan is at most αOPT (α is a constant). Then, from Lemma 1, our conversion method gives a bulk synchronous schedule S' whose makespan is at most $\alpha OPT(1 + c_{max}/c_{min})$. From $OPT \leq OPT_{BSSP}$, we obtain:

$$\begin{aligned} \frac{\alpha OPT(1 + c_{max}/c_{min})}{OPT_{BSSP}} &\leq \frac{\alpha OPT(1 + c_{max}/c_{min})}{OPT} \\ &= \alpha(1 + c_{max}/c_{min}) \end{aligned}$$

Therefore, the following theorem holds:

Theorem 2 *The conversion method in Section 5.1 generates a schedule with at most $\alpha(1 + c_{max}/c_{min})$ times optimal makespan for BSSP from an α-optimal schedule for GSP.*

In particular, we obtain the following corollary if $c_{min} = c_{max}$ holds.

Corollary 1 *The conversion method in Section 5.1 generates a schedule with at most 2α times optimal makespan for BSSPc from an α-optimal schedule for GSP.*

5.3 Application of the Conversion Method

There are several restricted cases where optimal or near-optimal schedules can be obtained for the general task scheduling problem. In Table 1, we summarize the performance guarantees of our approximation algorithm in cases that we use these existing algorithms as subroutines.

In the following of this section, we define the notions in Table 1. Let $G = (V, E, \lambda, \tau)$ be a task graph. For a node $v \in V$, let $g_1(v)$ and $g_2(v)$ be $\min\{\lambda(u)|(u,v) \in E\}/\max\{\tau(u,v)|(u,v) \in E\}$ and $\min\{\lambda(w)|(v,w) \in E\}/\max\{\tau(v,w)|(v,w) \in E\}$ respectively. The **granularity** of G is given by $g(G) = \min\{\min(g_1(v), g_2(v))|v \in V\}$[8]. A task graph G is **coarse-grained** if $g(G) \geq 1$; otherwise it is **fine-grained**[8]. An **in-tree** (**out-tree**, resp.) is a directed rooted tree in which every edge is directed from a node to its predecessor (successor, resp.)[10]. A **fork-join** is a hybrid of an in-tree and out-tree. It has an entry node which spawns a number of successors. The output edges of the successors connect to either an intermediate node that spawns an other set of successors or an exit node[1]. **Reduction computation** is computation concerning an equation comprised of operations where the number of the arguments is always the same and the associative law is satisfied[7]. A partial order Θ is called **series-parallel** if it can be obtained recursively from singletons by two operations, the **series composition** $\Theta_1 \times \Theta_2$ and the **parallel composition** $\Theta_1 \cup \Theta_2$ of two (series-parallel) suborders Θ_1, Θ_2 [9]. A partial order is called **interval-order** when its elements can be mapped into intervals on the real line and two elements are related iff the corresponding intervals do not overlap[4]. Let u_1 and u_2 be the predecessors of node v which have the highest and second-highest values of $\{ect(u) + \tau(u,v)|(u,v) \in E\}$, respectively. The following condition is called **TDS**[3]: $\lambda(u_1) \geq \tau(u_2, v)$ if $est(u_1) \geq est(u_2)$ or, $\lambda(u_1) \geq \tau(u_2,v) + est(u_2) - est(u_1)$ if $est(u_1) < est(u_2)$ where $est(v) = 0$ if $Pred(v) = \emptyset$, $est(v) = \min_{u \in Pred(v)} \max_{s \in Pred(v), s \neq u}(ect(u), ect(s) + \tau(s,v))$ if $Pred(v) \neq \emptyset$, $ect(v) = est(v) + \lambda(v)$. Note that condition TDS can be satisfied if the DAG is coarse-grained, but it is not necessary[3].

6 Conclusion

For the bulk synchronous task scheduling problem, we have presented the proof of NP-completeness of the problem and an approximation algorithm for several restricted

Table 1. The performance guarantee of the proposed approximation algorithm for BSSP and BSSPc (all the weight of nodes are unit time)

subroutine	DAG topology	edge weight	# of procs	BSSP guarantee	BSSPc guarantee
[4]	in-tree, out-tree	unit time	2	2	2
[2]	interval-order	unit time	arbitrary	2	2
[7]	reduction comp.	constant	unlimited	2	2
[3]	arbitrary	condition TDS	unlimited	$(1 + c_{max}/c_{min})$	2
[16]	fork, join coarse-grained tree some fine-grained trees	arbitrary	unlimited	$(1 + c_{max}/c_{min})$	2
[1]	out-tree, fork-join unit-height in-tree	arbitrary	unlimited	$(1 + c_{max}/c_{min})$	2
[9]	series-parallel	unit time	arbitrary p	$4 - 2/p$	$4 - 2/p$
[11]	arbitrary	constant	unlimited	4	4
[10]	arbitrary	arbitrary	unlimited	$2(1 + c_{max}/c_{min})$	4
[12]	arbitrary	unit time	arbitrary	6	6
[16]	arbitrary	arbitrary	unlimited	$(1 + 1/g(G))(1 + c_{max}/c_{min})$	$2(1 + 1/g(G))$

cases. In particular, we have presented that α-optimal general schedules can be converted into 2α-optimal bulk synchronous schedules if tasks are unit time and communication delays are a positive integer constant.

Acknowledgement

This research was supported in part by Grant-in-Aid for Scientific Research (C) No.11680357 from the Ministry of Education, Science, Sports and Culture of Japan, Hattori incorporated foundation, and Parallel and Distributed processing Consortium.

References

[1] I. Ahmad and Y. Kwok. On exploiting task duplication in parallel program scheduling. *IEEE Trans. on Parallel and Distributed Systems*, 9(9):872–892, 1998.

[2] H. Ali and H. El-Rewini. The time complexity of scheduling interval orders with communication is polynomial. *Parallel Processing Letters*, 3(1):53–58, 1994.

[3] S. Darbha and D. P. Agrawal. Optimal scheduling algorithm for distributed-memory machines. *IEEE Trans. on Parallel and Distributed Systems*, 9(1):87–95, 1998.

[4] H. El-Rewini, T. Lewis, and H. Ali. *TASK SCHEDULING in PARALLEL and DISTRIBUTED SYSTEMS.* PTR Prentice Hall, 1994.

[5] N. Fujimoto, T. Baba, T. Hashimoto, and K. Hagihara. A task scheduling algorithm to package messages on distributed memory parallel machines. In *Proc. of Int. Symposium on Parallel Architectures, Algorithms, and Networks*, pages 236–241, 1999.

[6] N. Fujimoto, T. Hashimoto, M. Mori, and K. Hagihara. On the performance gap between a task schedule and its corresponding parallel program. In *Proc. of Int. Workshop on Parallel and Distributed Computing for Symbolic and Irregular Applications*, pages 271–287. World Scientific, 2000.

[7] Y. Fukazawa, S. Kurino, and Y. Nonomura. A parallel reduction algorithm with communication delay. In *The Massively Parallel Processing System JUMP-1*, pages 50–60. Ohmsha, Ltd., 1996.

[8] A. Gerasoulis and T. Yang. On the granularity and clustering of directed acyclic task graphs. *IEEE Trans. on Parallel and Distributed Systems*, 4(6):686–701, 1993.

[9] R. H. Möhring, M. W. Schäffter, and A. S. Schulz. Scheduling jobs with communication delay: Using infeasible solutions for approximation. *LNCS1136*, pages 76–90, 1996.

[10] M. A. Palis, J. Liou, and D. S. L. Wei. Task clustering and scheduling for distributed memory parallel architectures. *IEEE Trans. on Parallel and Distributed Systems*, 7(1):46–55, 1996.

[11] C. H. Papadimitriou and M. Yannakakis. Towards an architecture-independent analysis of parallel algorithms. *SIAM J. Comput.*, 19(2):322–328, 1990.

[12] V. J. Rayward-Smith. Net scheduling with unit interprocessor communication delays. *Disc. Appl. Math.*, 18:55–71, 1987.

[13] R. Thurimella and Y. Yesha. A scheduling principle for precedence graphs with communication delay. In *Int. Conf. on Parallel Processing*, volume 3, pages 229–236, 1992.

[14] J. D. Ullman. Np-complete scheduling problems. *Journal of Computer and System Science*, 10:384–393, 1975.

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

[16] T. Yang and A. Gerasoulis. Dsc: Scheduling parallel tasks on an unbounded number of processors. *IEEE Trans. on Parallel and Distributed Systems*, 5(9):951–967, 1994.