Non-approximability of the Bulk Synchronous Task Scheduling Problem

Noriyuki Fujimoto and Kenichi Hagihara

Graduate School of Information Science and Technology,
Osaka University
1-3, Machikaneyama, Toyonaka, Osaka, 560-8531, Japan
{fujimoto, hagihara}@ist.osaka-u.ac.jp

Abstract. The mainstream architecture of a parallel machine with more than tens of processors is a distributed-memory machine. The bulk synchronous task scheduling problem (BSSP, for short) is an task scheduling problem for distributed-memory machines. This paper shows that there does not exist a ρ-approximation algorithm to solve the optimization counterpart of BSSP for any $\rho < \frac{6}{5}$ unless $\mathcal{P} = \mathcal{NP}$.

1 Introduction

Existing researches on the task scheduling problem for a distributed-memory machine (**DMM** for short) simply model DMMs as the parallel machines with large communication delays [2,12,13]. In contrast to this, in the papers [4,5,7], one noticed the following things by both analysis of architectural properties of DMMs and experiments to execute parallel programs which corresponds to schedules generated by existing task scheduling algorithms:

- It is essential to task scheduling for a DMM to consider the software overhead in communication, even if a DMM is equipped with a dedicated communication co-processor per processor.
- Existing task scheduling algorithms would ignore the software overhead.
- 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, in the papers [4,5,6,7], one proposed an optimization problem named the bulk synchronous task scheduling problem (**BSSPO** for short), i.e., the problem of finding a bulk synchronous schedule with small makespan. Formally, BSSPO is an optimization problem which restricts output, rather than input, of the general task scheduling problem with communication delays. A bulk synchronous schedule is a restricted schedule which has the following features:

- The well-known parallel programming technique to reduce the software overhead significantly , called message aggregation [1], can be applied to the parallel program which corresponds to the schedule.

B. Monien and R. Feldmann (Eds.): Euro-Par 2002, LNCS 2400, pp. 225–233.

– Makespan of the schedule approximates well the execution time of the parallel program applied message aggregation.

Hence a good BSSPO algorithm generates a schedule which becomes a fast parallel program on a DMM.

In this paper, we consider non-approximability of BSSPO. The decision counterpart of BSSPO (**BSSP**, for short) is known to be $\mathcal{NP}$-complete even in the case of unit time tasks and positive integer constant communication delays [6]. For BSSPO, two heuristic algorithms [4,5,7] for the general case and several approximation algorithms [6] for restricted cases are known. However, no results are known on non-approximability of BSSPO. This paper shows that there does not exist a ρ-approximation algorithm to solve BSSPO for any $\rho < \frac{6}{5}$ unless $\mathcal{P} = \mathcal{NP}$.

The remainder of this paper is organized as follows. First, we give some definitions in Section 2. Next, we review a bulk synchronous schedule in Section 3. Then, we prove non-approximability of BSSPO in Section 4. Last, in Section 5, we summarize and conclude the paper.

2 Preliminaries and Notation

A parallel computation is modeled as a task graph [3]. A **task graph** is represented by a weighted 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). 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 $\lambda(u)$ unit times. 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 $\tau(u, v)$ unit times if p is not equal to q. If p and 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 [14]. 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 $v \in V$, $q (1 \leq q \leq p)$ 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 three conditions R1 to R3 is called **feasible** (In the following of this paper, we abbreviate a feasible 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 $t \leq t' \leq t + \lambda(v)$.

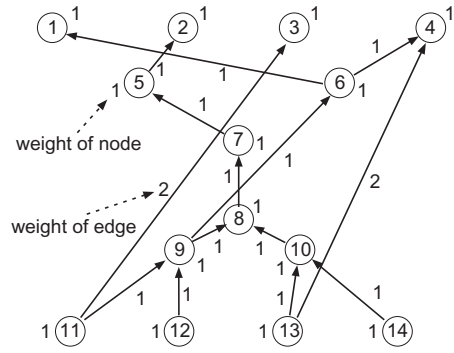

Fig. 1. An example of a task graph

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 $t' \leq t - \lambda(u)$ and $q = q'$, or with $t' \leq t - \lambda(u) - \tau(u, v)$ and $q \neq q'$.

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 **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.

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 $t_1 < t_2$,

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

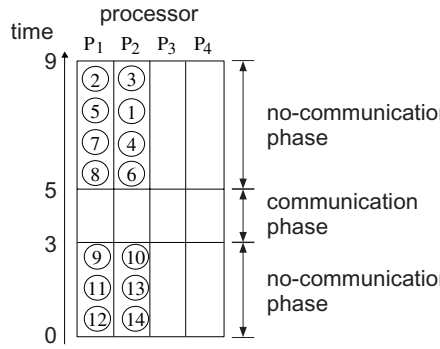

Fig. 2. An example of a bulk synchronous schedule

Notation $S[t_1, t_2]$ represents the set of all the triples such that both the starting time and the completion time of the task in a triple 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 $t' \leq t - \lambda(u)$ and $q = q'$, or with $t' \leq t_1 - \lambda(u) - \tau(u, v)$ and $q \neq q'$.

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 $\{(u, v)|(u, v) \in E, \langle u, q, t\rangle \in S[t_1, t_2], \langle v, q', t'\rangle \in S[t_3, t_4], \langle u, q', t''\rangle \notin S, q \neq q', t'' \leq t' - \lambda(u)\}$. 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 $S[t_3, t_4]$. Note that task duplication [8] is considered in the definition of $Comm(S, t_1, t_2, t_3, t_4)$. 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 two no-communication phases.

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

C2 For any i, j $(1 \leq i < j \leq m)$, $ct_i + \tau_{suff}(Comm(S, st_i, ct_i, st_j, ct_j)) \leq st_j$

Note that C2 considers communications between not only consecutive no-communication phases but also non consecutive ones. 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 Non-approximability of BSSP

4.1 An Overview of Our Proof

In this section, we prove that a ρ-approximation algorithm for BSSPO does not exist for any $\rho < \frac{6}{5}$ unless $\mathcal{P} = \mathcal{NP}$. For this purpose, we use the following lemma [11].

Lemma 1. *Consider a combinatorial minimization problem for which all feasible solutions have non-negative integer objective function value. Let k be a fixed positive integer. Suppose that the problem of deciding if there exists a feasible solution of value at most k is $\mathcal{NP}$-complete. Then, for any $\rho < (k+1)/k$, there does not exist a ρ-approximation algorithm unless $\mathcal{P} = \mathcal{NP}$.*

To extract our non-approximability result using Lemma 1, we prove $\mathcal{NP}$-completeness of BSSP in the case of a given fixed constant communication delay c and makespan at most $3+2c$ (3BSSP(c), for short) by reducing to 3BSSP(c) the unit time precedence constrained scheduling problem in the case of makespan at most 3 [10] (3SP, for short). These problems are defined as follows:

- 3BSSP(c) where c is a constant communication delay (positive integer).
  Instance: A task graph G such that all the weights of nodes are unit and all the weights of edges are the same as c, a number p of available processors
  Question: Is there a bulk synchronous schedule S_{BSP} whose makespan is at most $3 + 2c$?
- 3SP
  Instance: A task graph G such that all the weights of nodes are unit and all the weights of edges are the same as zero, a number p of available processors
  Question: Is there a schedule S whose makespan is at most 3 ?

$\mathcal{NP}$-completeness of 3SP was proved by Lenstra and Rinnooy Kan [10]. In the following, we denote an instance of 3BSSP(c) (3SP, resp.) as (G, p, c) ((G, p), resp.).

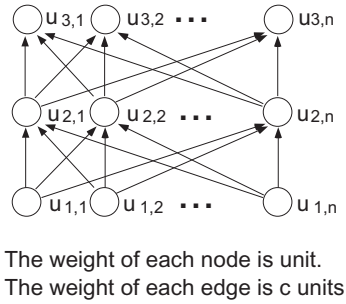

Fig. 3. A ladder graph $LG(n, c)$

4.2 A Ladder Graph and Its Bulk Synchronous Schedule

A **ladder graph** $LG(n, c)$ is a task graph such that $V = \{u_{i,j} | 1 \leq i \leq 3, 1 \leq j \leq n\}$, $E = \{(u_{i,j}, u_{i+1,k}) | 1 \leq i < 3, 1 \leq j \leq n, 1 \leq k \leq n\}$, $\lambda(v) = 1$ for any $v \in V$, and $\tau(e) = c$ for any $e \in E$. Fig. 3 shows a ladder graph $LG(n, c)$. Then, the following lemma follows.

Lemma 2. *For any positive integer c, any bulk synchronous schedule for a ladder graph $LG(3+2c, c)$ onto at least $(3+2c)$ processors within deadline $(3+2c)$ consists of three no-communication phases with one unit length.*

Proof. Let D be $3+2c$. Let S_{BSP} be a bulk synchronous schedule for a ladder graph $LG(D, c)$ onto p $(\geq D)$ processors within deadline D. Any $u_{i+1,j} (1 \leq i < 3, 1 \leq j \leq D)$ cannot construct a no-communication phase with all of $\{u_{i,k} | 1 \leq k \leq D\}$ because the computation time $(D+1)$ of these nodes on one processor is greater than the given deadline D. That is, any $u_{i+1,j} (1 \leq i < 3, 1 \leq j \leq D)$ must communicate with at least one of $\{u_{i,k} | 1 \leq k \leq D\}$. Hence, there exists a sequence $\{u_{1,k_1}, u_{2,k_2}, u_{3,k_3}\}$ of nodes such that $u_{i+1,k_{i+1}}$ communicates with u_{i,k_i} for any $i (1 \leq i < 3)$. This means that S_{BSP} includes at least two communication phases. On the other hand, S_{BSP} cannot include more than two communication phases because the deadline D is broken. Therefore, S_{BSP} includes just two communication phases. Consequently, S_{BSP} must consist of just three no-communication phases with one unit length. One of schedules possible as S_{BSP} is $\{\langle u_{i,j}, j, (i-1)(c+1)\rangle | 1 \leq i \leq 3, 1 \leq j \leq D\}$ (See Fig. 4). □

4.3 A Polynomial-Time Reduction

Now, we show the reduction from 3SP to 3BSSP(c). Let $(G = (V_G, E_G, \lambda_G, \tau_G), p)$ be an instance of 3SP. Let c be any positive integer. Let $LG(3 + 2c, c) = (V_{LG}, E_{LG}, \lambda_{LG}, \tau_{LG})$ be a ladder graph. Let G' be a task graph $(V_G \cup V_{LG}, E_G \cup E_{LG}, \lambda_{G'}, \tau_{G'})$ where $\lambda_{G'}(v) = 1$ for any $v \in V_G \cup V_{LG}$, and $\tau_{G'}(e) = c$ for any $e \in E_G \cup E_{LG}$.

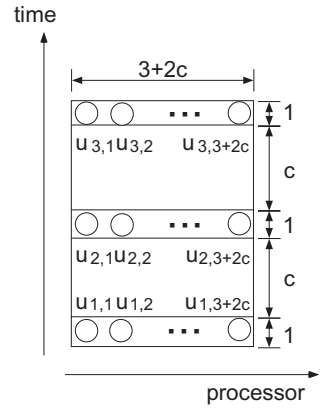

Fig. 4. A bulk synchronous schedule of a ladder graph $LG(3+2c, c)$ onto at least $(3+2c)$ processors within deadline $(3+2c)$

Lemma 3. *The transformation from an instance (G, p) of 3SP to an instance $(G', p+3+2c, c)$ of 3BSSP(c) is a polynomial transformation such that (G, p) is a yes instance iff $(G', p+3+2c, c)$ is a yes instance.*

Proof. If (G, p) is a "yes" instance of 3SP, then let S be a schedule for (G, p). A set $\{\langle v, q, t(c+1)\rangle | \langle v, q, t\rangle \in S\} \cup \{\langle u_{i,j}, p+j, (i-1)(c+1)\rangle | 1 \leq i \leq 3, 1 \leq j \leq 3+2c\}$ of triples is a bulk synchronous schedule for $(G', p+3+2c)$ with three no-communication phases and two communication phases (See Fig. 5).

Conversely, if $(G', p+3+2c, c)$ is a "yes" instance of 3BSSP(c), then let S'_{BSP} be a schedule for $(G', p+3+2c, c)$. From Lemma 2, $LG(3+2c, c)$ must be scheduled into a bulk synchronous schedule which consists of just three no-communication phases with one unit length. Therefore, whole S'_{BSP} must consists of just three no-communication phases with one unit length. Hence, S'_{BSP} must become a schedule as shown in Fig. 5. A subset $\{\langle v, q, t\rangle | \langle v, q, t(c+1)\rangle \in S'_{BSP}, 1 \leq q \leq p\}$ of S'_{BSP} is a schedule for (G, p). □

Theorem 1. *For any positive integer c, 3BSSP(c) is $\mathcal{NP}$-complete.*

Proof. Since BSSP(c) is $\mathcal{NP}$-complete [6], it is obvious that 3BSSP(c) is in $\mathcal{NP}$. Hence, from Lemma 3, the theorem follows. □

Let BSSPO(c) be the optimization counterpart of BSSP(c).

Theorem 2. *Let c be any positive integer. Then, a ρ-approximation algorithm for BSSPO(c) does not exist for any $\rho < \frac{4+2c}{3+2c}$ unless $\mathcal{P} = \mathcal{NP}$.*

Proof. From Theorem 1 and Lemma 1, the theorem follows. □

Theorem 3. *A ρ-approximation algorithm for BSSPO does not exist for any $\rho < \frac{6}{5}$ unless $\mathcal{P} = \mathcal{NP}$.*

Proof. From Theorem 2, a ρ'-approximation algorithm for BSSPO(1) does not exist for any $\rho' < \frac{6}{5}$ unless $\mathcal{P} = \mathcal{NP}$. If a ρ-approximation algorithm A for

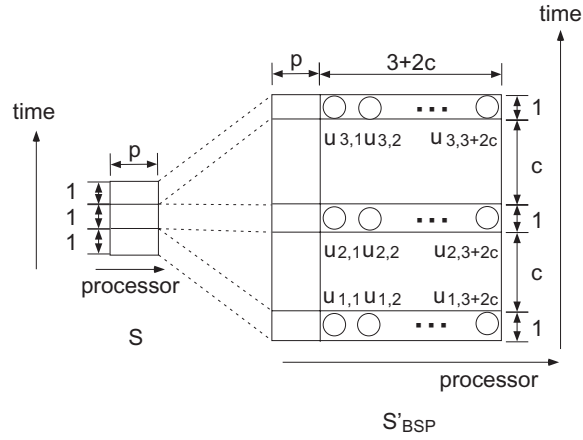

Fig. 5. A yes instance to a yes instance correspondence

BSSPO exists for some $\rho < \frac{6}{5}$, A can be used as a ρ-approximation algorithm for BSSPO(1). Hence, the theorem follows. □

5 Conclusion and Future Work

For the bulk synchronous task scheduling problem, we have proved that there does not exist a ρ-approximation algorithm for any $\rho < \frac{6}{5}$ unless $\mathcal{P} = \mathcal{NP}$.

In order to prove that, we have showed that generating a bulk synchronous schedule of length at most 5 is $\mathcal{NP}$-hard. However, the complexity of the problem for a schedule of length at most 4 is unknown. The $\mathcal{NP}$-hardness means non-approximability stronger than our result. So, one of the future work is to clear the complexity like Hoogeveen et al.'s work [9] for the conventional (i.e., not bulk synchronous) task scheduling problem.

Acknowledgement

This research was supported in part by the Kayamori Foundation of Informational Science Advancement.

References

1. Bacon, D.F. and Graham, S.L. and Sharp, O.J.: Compiler Transformations for High-Performance Computing, ACM computing surveys, Vol.26, No.4 (1994) 345-420
2. Darbha, S. and Agrawal, D. P.: Optimal Scheduling Algorithm for Distributed-Memory Machines, IEEE Trans. on Parallel and Distributed Systems, Vol.9, No.1 (1998) 87-95

3. El-Rewini, H. and Lewis, T.G. and Ali, H.H.: TASK SCHEDULING in PARALLEL and DISTRIBUTED SYSTEMS, PTR Prentice Hall (1994)
4. Fujimoto, N. and Baba, T. and Hashimoto, T. and Hagihara, K.: A Task Scheduling Algorithm to Package Messages on Distributed Memory Parallel Machines, Proc. of 1999 Int. Symposium on Parallel Architectures, Algorithms, and Networks (1999) 236-241
5. Fujimoto, N. and Hashimoto, T. and Mori, M. and Hagihara, K.: On the Performance Gap between a Task Schedule and Its Corresponding Parallel Program, Proc. of 1999 Int. Workshop on Parallel and Distributed Computing for Symbolic and Irregular Applications, World Scientific (2000) 271-287
6. Fujimoto, N. and Hagihara, K.: NP-Completeness of the Bulk Synchronous Task Scheduling Problem and Its Approximation Algorithm, Proc. of 2000 Int. Symposium on Parallel Architectures, Algorithms, and Networks (2000) 127-132
7. Fujimoto, N. and Baba, T. and Hashimoto, T. and Hagihara, K.: On Message Packaging in Task Scheduling for Distributed Memory Parallel Machines, The International Journal of Foundations of Computer Science, Vol.12, No.3 (2001) 285-306
8. Kruatrachue, B., "Static task scheduling and packing in parallel processing systems", Ph.D. diss., Department of Electrical and Computer Engineering, Oregon State University, Corvallis, 1987
9. Hoogeveen, J. A., Lenstra, J. K., and Veltman, B.: "Three, Four, Five, Six or the Complexity of Scheduling with Communication Delays", Oper. Res. Lett. Vol.16 (1994) 129-137
10. Lenstra, J. K. and Rinnooy Kan, A. H. G.: Complexity of Scheduling under Precedence Constraints, Operations Research, Vol.26 (1978) 22-35
11. Lenstra, J.K. and Shmoys, D. B.: Computing Near-Optimal Schedules, Scheduling Theory and its Applications, John Wiley & Sons (1995) 1-14
12. Palis, M. A. and Liou, J. and Wei, D. S. L.: Task Clustering and Scheduling for Distributed Memory Parallel Architectures, IEEE Trans. on Parallel and Distributed Systems, Vol.7, No.1 (1996) 46-55
13. Papadimitriou, C. H. and Yannakakis, M.: Towards An Architecture-Independent Analysis of Parallel Algorithms, SIAM J. Comput., Vol.19, No.2 (1990) 322-328
14. Thurimella, R. and Yesha, Y.: A scheduling principle for precedence graphs with communication delay, Int. Conf. on Parallel Processing, Vol.3 (1992) 229-236
15. Valiant, L.G.: A Bridging Model for Parallel Computation, Communications of the ACM, Vol.33, No.8 (1990) 103-111