DAG Scheduling in the BSP Model

Pál András Papp[0009−0005−6667−802X], Georg Anegg[0000−0002−5730−5812], and Albert-Jan N. Yzelman[0000−0001−8842−3689]

Computing Systems Lab, Huawei Zurich Research Center
{pal.andras.papp,albertjan.yzelman}@huawei.com

Abstract. We study the problem of scheduling an arbitrary computational DAG on a fixed number of processors while minimizing the makespan. While previous works have mostly studied this problem in fairly restricted models, we define and analyze DAG scheduling in the Bulk Synchronous Parallel (BSP) model, which is a well-established parallel computing model that captures the communication cost between processors much more accurately. We provide a taxonomy of simpler scheduling models that can be understood as variants or special cases of BSP, and discuss how the properties and optimum cost of these models relate to BSP. This essentially allows us to dissect the different building blocks of the BSP model, and gain insight into how these influence the scheduling problem.
We then analyze the hardness of DAG scheduling in BSP in detail. We show that the problem is solvable in polynomial time for some very simple classes of DAGs, but it is already NP-hard for in-trees or DAGs of height 2. We also prove that in general DAGs, the problem is APX-hard: it cannot be approximated to a $(1 + \epsilon)$-factor in polynomial time for some specific $\epsilon > 0$. We then separately study the subproblem of scheduling communication steps, and we show that the NP-hardness of this problem depends on the problem parameters and the communication rules within the BSP model. Finally, we present and analyze a natural formulation of our scheduling task as an Integer Linear Program.

Keywords: Bulk synchronous parallel · NP-hard · APX-hard · ILP.

1 Introduction

The optimal scheduling of complex workloads is a fundamental problem not only in computer science, but also in other areas like logistics or operations research. In a computational context, the most natural application of scheduling is when we have a complex computation consisting of many different subtasks, and we want to execute this on a parallel (multi-processor or multi-core) architecture, while minimizing the total time required for this. Unsurprisingly, this topic has been extensively studied since the 1960s, and has gained even more importance recently with the widespread use of manycore architectures.

In these scheduling problems, a computational task is represented as a Directed Acyclic Graph (DAG), where each node corresponds to an operation or subtask, and each directed edge (u, v) indicates a dependency relation, i.e. that the processing of node u has to be finished before the processing of node v begins, since the output of operation u is needed as an input for operation v. This DAG model of general computations is not only prominent in scheduling, but also in further topics such as pebble games.

However, from a complexity-theoretic perspective, the scheduling of general DAGs is already a hard problem even in very simple settings, e.g. even in models that heavily simplify or completely ignore the communication costs between processors, which is the main bottleneck in many computational tasks in practice. Due to this, previous theoretical works have mostly focused on analyzing the complexity of scheduling in these rather simple models, and while more realistic models were sometimes introduced, the theoretical properties of scheduling in these more realistic models received little attention.

On the other hand, the parallel computing community has developed far more advanced models to accurately quantify the real cost of parallel algorithms in practice. One of the most notable is the Bulk Synchronous Parallel (BSP) model, which is still relatively simple, but provides a delicate cost function that captures the volume of communicated data and the synchronization costs in a given parallel schedule. BSP (and similar models) are fundamental tools for evaluating and comparing concrete practical implementations of parallel algorithms. However, previous theoretical works on BSP only focus on finding and analyzing parallel schedules for specific algorithms, and do not study BSP as a model for scheduling general DAGs, i.e. an arbitrary computational task.

Our goal in this paper is to bridge this gap between theory and practice to some extent, and understand the fundamental theoretical properties of DAG scheduling in the BSP model. This more detailed model results in a more complex scheduling problem with some entirely new aspects compared to classical models; as such, understanding its key properties is a crucial step towards designing efficient parallel schedules for computations in practice. Our hope is that our insights inspire a new line of work on the theoretical study of scheduling in more advanced parallel computing models that are developed and applied on the practical side. More specifically, our main contributions are as follows:

(i) We first define DAG scheduling in the BSP model. We then provide a taxonomy of scheduling models from previous works, and show that many of

these can be understood as a special case of BSP. Analyzing the relations between these models essentially allows us to gain insight into how the different aspects of BSP affect the scheduling problem.

(ii) We then analyze the complexity of BSP scheduling, with the goal of understanding when (i.e. for which kind of DAGs) it becomes NP-hard. We show that the problem is still solvable in polynomial time for DAGs that consist of several connected chains, but it is already NP-hard for slightly more complex classes of DAGs, such as in-trees or DAGs of height 2.

(iii) We show that for general DAGs, the problem is NP-hard to even approximate to a $(1+\epsilon)$ factor, for some $\epsilon > 0$. In order words, BSP scheduling is APX-hard: it allows no polynomial-time approximation scheme unless P=NP.

(iv) We separately analyze the subproblem of scheduling communication steps, assuming that the rest of the schedule is already fixed. We discuss the complexity of this subproblem for several different variants of BSP.

(v) Finally, we present and analyze a natural formulation of the BSP scheduling task as an Integer Linear Programming (ILP) problem.

Our main technical contributions are points (ii)–(iv) above; however, the remaining points also provide valuable insight into the problem. The proof details and some further model discussion are deferred to the full version of the paper [42].

2 Related Work

DAG scheduling is a fundamental problem in computer science, and has been studied extensively since the 1970s. The first papers considered a simple setting where communication between processors is free, which essentially corresponds to the PRAM model. The numerous results in this model include polynomial algorithms for $P=2$ processors [5,45,14], polynomial algorithms for special classes of DAGs [15,11,10], hardness results for $P=\infty$ [50,3,25], and results for weighted DAGs [24,37,4]. The results on some of these topics, e.g. approximation algorithms, are still rapidly improving in recent years [47,28,16,30]. On the other hand, some basic questions are still open even in this fundamental model: e.g. it is still not known whether scheduling for some fixed $P>2$ is NP-hard.

A more realistic version of this model was introduced in the late 1980s [40,54], where there is a fixed communication delay between processors. There are also numerous algorithms and hardness results for this setting, in particular for unit-length delays [19,21] or infinitely many processors [33,21,38]. The approximability of the optimal solution is also a central question in this model that receives significant attention even in recent years [23,31,8,32,7].

This communication delay model is still unrealistic, as it allows an unlimited amount of data to be sent in a single time unit. To our knowledge, the only more sophisticated DAG scheduling model which measures communication volume is the recently introduced single-port duplex model [39]. However, this model has not been studied from a theoretical perspective, and in contrast to BSP, it cannot be extended by some real-world aspects such as synchronization costs.

There are also numerous extensions of these models with further aspects, e.g. heterogeneous processors or deadlines for each task [26,27,48,29,9,43].

On the other hand, BSP has been introduced as a prominent model of parallel computing in 1990 [51], and studied extensively ever since [34,2,46]. The model has also found its way into various applications, most notably through the BSPlib standard library [20] and its different implementations [56,55]. Fundamental results on BSP include the analysis of prominent algorithms in this model [35,36] or the extension of BSP to multi-level architectures [52,53]. However, the BSP model (and similar models with even more parameters, such as LogP [6]) have mostly been used so far to analyze the computational costs of specific parallel implementations of concrete algorithms. In contrast to this, in our work, we apply BSP as a general model to evaluate the scheduling of any DAG.

3 Model and Background

Computational tasks are modelled as a *Directed Acyclic Graph* G, with the set of nodes (subtasks) denoted by V, and the set of directed edges (dependencies) by E. We use u and v to denote individual nodes, and n to denote the number of nodes $|V|$. An edge (u, v) indicates that subtask u has to be finished before the computation of subtask v begins. We also use $[k]$ as a shorthand notation for the integer set $\{1, ..., k\}$.

We assume that we have $P \in O(1)$ identical processors, and our goal is to execute all nodes of G on these, while minimizing the total time this takes.

The BSP model. BSP is a very popular model for the design and evaluation of parallel algorithms. To our knowledge, general DAG scheduling has not been studied in this model before, but extending the interpretation of BSP to this setting is rather straightforward.

In contrast to classical models where nodes are assigned to concrete points in time, the BSP model instead divides the execution of nodes into larger batches, so-called *supersteps*. Each superstep consists of two phases, in the following order:

1. *Computation phase*: each processor may execute an arbitrary number of computation steps, but no communication between the processors is allowed.
2. *Communication phase*: processors can communicate an arbitrary number of values to each other, but no computation is executed.

The motivation behind supersteps is to encourage sending data in large batches, since in practice, communication often has a large fixed cost (e.g. synchronization, network initialization) that is independent of data volume.

Formal definition. Let us denote the number of supersteps in our schedule by S. While S can be freely chosen in a schedule, we provide our definitions for a fixed S for simplicity. A *BSP schedule* with S supersteps consists of:

- An assignment of nodes to processors $\pi : V \to [P]$ and to supersteps $\tau : V \to [S]$. For simplicity, we introduce the notation $H^{(s,p)} = \{v \in V \mid \pi(v) = p, \tau(v) = s\}$ for the set of nodes assigned to processor p and superstep s. We can imagine the nodes of $H^{(s,p)}$ to be executed in an arbitrary (but topologically correct) order on p in superstep s.
- A set Γ of 4-tuples $(v, p_1, p_2, s) \in V \times [P] \times [P] \times [S]$, indicating that the output of node v is sent from processor p_1 to processor p_2 in the communication phase of superstep s. In this base variant of BSP, we only include p_1 in these 4-tuples for clarity, but we always assume $p_1 = \pi(v)$, i.e. the value is sent from the processor where it was computed.

A valid BSP schedule must satisfy the following conditions:

(i) A node v can only be computed if all of its predecessors are available, i.e. they were computed on processor $\pi(v)$ in an earlier (or the same) superstep, or sent to $\pi(v)$ before the given superstep. That is, for all $(u, v) \in E$, if $\pi(u) = \pi(v)$ then we must have $\tau(u) \leq \tau(v)$, and if $\pi(u) \neq \pi(v)$ then we must have $(u, \pi(u), \pi(v), s) \in \Gamma$ for some $s < \tau(v)$.

(ii) We only communicate values that are already computed: if $(v, p_1, p_2, s) \in \Gamma$, then $p_1 = \pi(v)$, and $\tau(v) \leq s$.

Cost function. The computation phase can be executed in parallel on the different processors, so its cost (the amount of time it takes) in superstep $s \in [S]$ is the largest amount of computation executed on any of the processors. More formally, the *work cost* of superstep s (first for a given processor $p \in [P]$, and then in general) is defined as

$$C_{work}{}^{(s,p)} = |H^{(s,p)}| \qquad \text{and} \qquad C_{work}{}^{(s)} = \max_{p \in [P]} C_{work}{}^{(s,p)} .$$

Communication costs, on the other hand, are governed by two further problem parameters: $g \in \mathbb{N}$ is the cost of communicating a single unit of data, and $L \in \mathbb{N}$ is the fixed latency cost incurred by each superstep. BSP assumes that different values can be communicated in parallel in general, but any processor can only send and receive a single value in any time unit. As such, BSP considers the number of values sent and received by processor p in superstep s, and then defines the *communication cost* of a superstep (for p, or in general) as the maximum of these; this cost function is called *h-relation*. More formally, let

$$C_{sent}{}^{(s,p)} = |\{(v, p, p', s) \in \Gamma\}| \qquad \text{and} \qquad C_{rec}{}^{(s,p)} = |\{(v, p', p, s) \in \Gamma\}|$$

for some fixed $s \in [S]$ and $p \in [P]$ (over all $v \in V$ and $p' \in [P]$), and then let

$$C_{comm}{}^{(s,p)} = \max(C_{sent}{}^{(s,p)}, C_{rec}{}^{(s,p)}) \quad \text{and} \quad C_{comm}{}^{(s)} = \max_{p \in [P]} C_{comm}{}^{(s,p)} .$$

The cost $C^{(s)}$ of superstep s and the cost C of the entire schedule is defined as:

$$C^{(s)} = C_{work}{}^{(s)} + g \cdot C_{comm}{}^{(s)} + L \qquad \text{and} \qquad C = \sum_{s \in [S]} C^{(s)} .$$

For an example, consider the BSP schedule shown in Figure 1, and let $s=1$, $p_1=1$, $p_2=2$. Here processor p_1 computes 4 nodes, and processor p_2 computes 5 nodes, so ${C_{work}}^{(s,p_1)} = 4$, ${C_{work}}^{(s,p_2)} = 5$, and ${C_{work}}^{(s)} = \max(4,5) = 5$ in the computation phase. In the communication phase, p_1 must send a single value to p_2 (so ${C_{sent}}^{(s,p_1)} = {C_{rec}}^{(s,p_2)} = 1$), while p_2 must send two values to p_1 (${C_{sent}}^{(s,p_2)} = {C_{rec}}^{(s,p_1)} = 2$). This implies ${C_{comm}}^{(s,p_1)} = {C_{comm}}^{(s,p_2)} = \max(2,1) = 2$, and hence ${C_{comm}}^{(s)} = 2$. The total cost of the superstep is $C^{(s)} = 5 + 2 \cdot g + L$. For more details on BSP, we refer the reader to [34,2].

As a scheduling problem. We can now formally define our problem.

Definition 1. *Given an input DAG, the goal of* BSP scheduling *is to find a feasible BSP schedule (π, τ, Γ) as described above, with minimal cost C.*

In the decision version of the problem, we also have a maximal cost parameter C_0, and we need to decide if there is a BSP schedule with cost $C \leq C_0$. For simplicity, we will often focus on the simplest case of $L=0$.

From a complexity perspective, it is important to note that we consider the parameters P, g, L to be small fixed constants (properties of our computing architecture), and not parts of the problem input. We especially emphasize this for P, since in contrast to our work, some others assume that P is an input variable that can be up to linear in n; however, this is unrealistic in most applications, and also makes the problem unreasonably hard even for trivial DAGs. In general, both settings (fixed P and variable P) have been extensively studied before, and are distinguished by "Pm" and "P" in the classical 3-field notation [18].

4 Comparison to Other Models

4.1 Taxonomy of Scheduling Models

In the most basic *classical scheduling* model, nodes are assigned to processors $\pi : V \to [P]$ and time steps $t : V \to \mathbb{Z}^+$, with two simple conditions: $\nexists\, u, v \in V$ with $\pi(u)=\pi(v)$, $t(u)=t(v)$), and $\forall (u,v) \in E$ we have $t(u) < t(v)$. The cost of a schedule here is simply $\max_{v \in V}\, t(v)$. A more realistic version of this model also adds a fixed communication delay g between processors: $\forall (u,v) \in E$, in case if $\pi(u) \neq \pi(v)$, we now need to have $t(v) > t(u) + g$.

BSP has two major differences from this latter *commdelay* model. Firstly, commdelay allows a processor to execute computations and communications simultaneously, while in BSP, these must happen in separate phases. Moreover, commdelay implicitly allows any amount of data to be sent from p_1 to p_2 simultaneously, whereas BSP also considers data volume: sending k values from p_1 to p_2 takes k times as long as a single value. Previous work has briefly considered the extension of commdelay with both of these modifications separately:

– The work of [13] considers a variant of commdelay where computation and communication can only happen in separate phases.

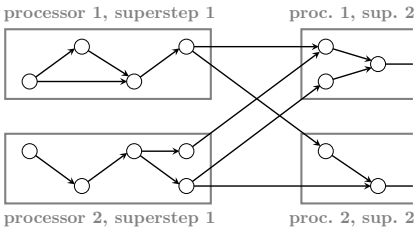

Fig. 1: Example BSP schedule for a DAG. The labelled boxes only represent the computation phases; the supersteps itself also consists of the communication phase that follows.

	free comm. (no cost)	simplified comm. cost (any amount of data in a single step)	exact comm. cost (depends on data volume)
comp. & comm. simultaneously	classical scheduling [5,45, ...]	commdelay [40,23, ...]	single-port duplex [39]
comp. & comm. in separate phases	classical scheduling [5,45, ...]	commdelay with phases [13]	**BSP [novel for DAGs]**

Table 1: Taxonomy of DAG scheduling models. The horizontal axis shows communication cost models, and the vertical axis shows whether simultaneous computation and communication is allowed.

– The work of [39] introduces a *single-port duplex* (SPD) model with communication volume: a schedule here must also specifically assign the send/recieve steps to concrete (disjoint) time intervals for each processor.

The summary of these models in Table 1 shows that BSP scheduling indeed fills a natural place in this taxonomy. Note that in the last column, the difference between SPD and BSP is in fact twofold. Firstly, BSP assumes that at any time, a processor can either compute or communicate, but not both. Secondly, BSP divides the schedule into supersteps, which can be understood as a *barrier synchronization* requirement: to send a value v, we first need a point in time (after computing v) when no processor is computing, to initiate the process of sending v (and possibly other values). By separating these two properties, we can extend our taxonomy into Table 2 to include further model variants.

– If global synchronization is required (so we have supersteps), but processors can compute and communicate simultaneously: the first paper on BSP [51] implicitly assumes this *maxBSP* model variant, defining $C^{(s)}$ as the maximum of ${C_{work}}^{(s)}$ and $g \cdot {C_{comm}}^{(s)} + L$. In contrast, recent textbooks [2] apply our definition from Section 3 with a sum. Note that defining maxBSP for general DAGs requires further consideration, to ensure that the computation and communication phases of a superstep are indeed parallelizable.
– If processors can only either compute or communicate at a given time, but no synchronization is needed: one can interpret this as the $\alpha-\beta$ model [44] with a choice of $\alpha = 0$, although the definition of this model varies (see Appendix A.4 of [42]). This allows e.g. p_1 and p_2 to stop computing and exchange values, while the rest of the processors keep computing in the meantime.

One natural question regarding this taxonomy is how the optimum costs in these models (denoted by OPT) relate to each other for a given DAG. Firstly, note that in any of the models, one of the processors must have a work cost of $\frac{n}{P}$

		free communication (no cost)	simplified comm. cost (any amount of data in a single step)	exact comm. cost (cost depends on data volume)
comp. & comm. simultaneously	no sync	classical scheduling	commdelay	single-port duplex
	sync	classical scheduling	commdelay with sync points	maxBSP
comp. & comm. separately	no sync	classical scheduling	$\alpha-\beta$ with $\beta=0$ (or subset-CD)	$\alpha-\beta$ with $\alpha=0$ (or subset-BSP)
	sync	classical scheduling	commdelay with phases	BSP

Table 2: Extended table of DAG scheduling models. The vertical axis is split according to (i) whether computation and communication are allowed simultaneously, and (ii) whether barrier synchronization is required for communication.

at least; this provides a lower bound. Moreover, executing the entire DAG on a single processor (without communication) always yields a valid solution of cost n, and hence it always gives a factor P approximation of the optimum.

Proposition 1. *We have $\frac{n}{P} \leq OPT \leq n$ in any of these models.*

Next we compare the optimum cost in the two fundamental models, classical scheduling and commdelay, to the optimum in BSP (denoted by OPT_{class}, OPT_{CD} and OPT_{BSP}, respectively, assuming $L=0$ in BSP). These clearly satisfy $\mathrm{OPT}_{class} \leq \mathrm{OPT}_{CD} \leq \mathrm{OPT}_{BSP}$. Finding the maximal difference is more involved; however, note that e.g. Proposition 1 already implies that the optimum costs in any two models differ by at most a factor P.

Lemma 1. *We have $OPT_{BSP} \leq P \cdot OPT_{class}$ for any DAG and parameters P and g. Moreover, $OPT_{CD} \leq (1+g) \cdot OPT_{class}$. These bounds are essentially tight: there are DAG constructions with $\frac{OPT_{CD}}{OPT_{class}} = P$, $\frac{OPT_{BSP}}{OPT_{CD}} = P$, and $\frac{OPT_{CD}}{OPT_{class}} = (1+g-\varepsilon)$ for any $\varepsilon > 0$.*

Due to our focus on BSP, we also analyze the relation between the models in the last column of Table 2. Let us denote their optimum costs by OPT_{SPD}, OPT_{mBSP}, OPT_{β} and OPT_{BSP} from top to bottom, again for $L=0$. The restrictiveness of the models implies $\mathrm{OPT}_{SPD} \leq \mathrm{OPT}_{mBSP} \leq \mathrm{OPT}_{BSP}$ and $\mathrm{OPT}_{SPD} \leq \mathrm{OPT}_{\beta} \leq \mathrm{OPT}_{BSP}$; we complement with some further observations.

Theorem 1. *For any DAG and parameters P, g, the optimal costs in the models of the last column can differ by a factor 2 at most, i.e. $OPT_{BSP} \leq 2 \cdot OPT_{SPD}$. Moreover, we show DAG constructions that prove (for any $\varepsilon > 0$)*

i) a lower bound of $(2-\varepsilon)$ for $\frac{OPT_{\beta}}{OPT_{SPD}}$, $\frac{OPT_{BSP}}{OPT_{SPD}}$, $\frac{OPT_{\beta}}{OPT_{mBSP}}$ and $\frac{OPT_{BSP}}{OPT_{mBSP}}$,

ii) a looser lower bound of $(\frac{3}{2}-\varepsilon)$ for $\frac{OPT_{mBSP}}{OPT_{SPD}}$, $\frac{OPT_{BSP}}{OPT_{\beta}}$ and $\frac{OPT_{mBSP}}{OPT_{\beta}}$.

On a high level, $\mathrm{OPT}_{BSP} \leq 2 \cdot \mathrm{OPT}_{SPD}$ follows from a simple conversion of an SPD schedule to BSP. The lower bounds in i) use a construction that allows

to parallelize computation and communication almost perfectly, while ii) uses a DAG where different processors would ideally always send data at different times, so requiring barrier synchronization notably increases the optimum.

4.2 Communication Models Within BSP

Even within BSP, there are different options to model the communication rules, and these seemingly small changes can have a significant effect on the model.

For instance, our base assumed that $\pi(v) = p_1$ for all $(v, p_1, p_2, s) \in \Gamma$ for simplicity, i.e. values are sent from the processor where they were computed. However, to transfer a value from p_1 to p_2, one might as well send it from p_1 to a third processor p_3 first, and then from p_3 to p_2. Moreover, there are simple examples where such *free data movement* between processors can indeed result in a lower communication cost altogether. Consider the schedule in Figure 2 with $P = 3$ processors; this has a node that is computed on p_3 in superstep 1, but later only needed on p_1 in superstep 3. With direct transfer, we can send this data from p_3 to p_1 in either superstep 1 or 2; however, p_1 must already receive a value in superstep 1, and p_3 must send a value in superstep 2, so both options increase the cost in one of the supersteps. On the other hand, with free data movement, we can send the value from p_3 to p_2 in superstep 1, and then from p_2 to p_1 in superstep 2, without increasing the cost in either superstep.

Another interesting question occurs when processor p wants to send a single value v to multiple other processors $p_1, ..., p_k$ in the same superstep. In our base model, this requires a separate entry (v, p, p_i, s) for all $i \in [k]$, and hence contributes k units to the send cost ${C_{send}}^{(s,p)}$. This is a reasonable assumption e.g. if the communication topology is a clique, and thus p needs to send this value over k distinct network links. However, in e.g. a star-shaped communication topology, it can be more reasonable to only charge a single unit of send cost for this, i.e. to assume that data transfers are *broadcast operations*, and their values can be received by any number of processors.

These options can be combined to form 4 different *communication models* within BSP. We name these models in Table 3. Our results in Sections 7 and 8 show that these models can indeed influence some properties of the problem.

5 NP-hardness

Regarding the complexity of BSP scheduling, it is not surprising that the problem is NP-hard in general DAGs. However, this raises a natural follow-up question, which has also been studied in simpler models: in which subclasses of DAGs is the problem still solvable in polynomial time, and when does it become NP-hard?

The simplest non-trivial subclass is *chain DAGs*, where both the indegree and outdegree of nodes is at most 1. This subclass has been analyzed in different scheduling models [12,48], including a proof of NP-hardness in BSP [17] if we assume a compressed input representation (see [42] for a discussion). We also consider an extension of this subclass: we say that a DAG is a *connected chain*

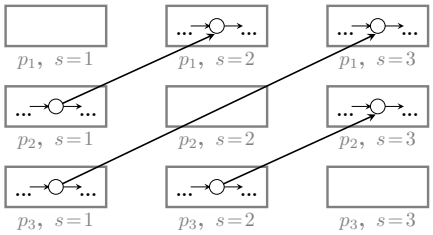

Fig. 2: An example BSP schedule where free data movement allows for a lower communication cost than direct data transfer.

	Singlecast	Broadcast
Direct transfer	**DS model** CS: *open problem* ILP: $O(n{\cdot}P{\cdot}S)$ *vars*	**DB model** CS: *NP-hard* ILP: $O(n{\cdot}P{\cdot}S)$ *vars*
Free data movement	**FS model** CS: *NP-hard* ILP: $O(n{\cdot}P^2{\cdot}S)$ *vars*	**FB model** CS: *NP-hard* ILP: $O(n{\cdot}P{\cdot}S)$ *vars*

Table 3: Communication models *within BSP*, and their properties shown in Sections 7 and 8. Our main results (Theorems 2–5) hold in all of these models.

DAG if it can be obtained by adding an extra source node v_0 to a chain DAG, and drawing an edge from v_0 to the first node in every chain. We show that for these simple classes of DAGs, the optimal BSP schedule can still be found in polynomial time. The key observation is that the optimal BSP schedule in chain DAGs always consists of at most P supersteps; this allows us to find the optimum through a rather complex dynamic programming approach.

Theorem 2. *The BSP scheduling problem can be solved in polynomial time in n for chain DAGs and connected chain DAGs.*

On the other hand, it turns out that BSP scheduling already becomes NP-hard for slightly more complex DAGs. In particular, we consider (i) DAGs with height only 2, where *height* is the number of nodes in the longest directed path, and (ii) *in-trees*, which are DAGs where every node has outdegree at most 1. We prove that in these (still relatively simple) classes, the problem is already NP-hard. Since the scheduling problem is known to be polynomially solvable for these classes in simpler models (specifically, for in-trees in classical scheduling [15], and for height-2 DAGs in commdelay [33]), this shows that our more advanced model indeed comes at the cost of increased complexity.

Theorem 3. *BSP scheduling is NP-hard if restricted to DAGs of height* 2*.*

Theorem 4. *BSP scheduling is NP-hard if restricted to in-trees.*

Proof sketch. The proof of Theorem 3 uses a reduction from k-clique, and can be understood as an adaptation of the reduction approach for partitioning in [41] to our setting. Intuitively, the second level of the DAG consists of gadgets representing each node of the input graph, while the first level consists of gadgets representing the edges; each edge gadget is connected to the two incident node gadgets. Our construction ensures that at most k of the node gadgets can be assigned to a given processor p without incurring a too large work cost, and that each edge gadget incurs a communication step exactly if one of its incident node

gadgets is not assigned to p. With the appropriate cost limit C_0, the DAG only admits a valid schedule if there are k original nodes that induce $\binom{k}{2}$ edges.

Theorem 4 is our most technical proof, requiring $P = 16$ processors. Firstly, our choice of C_0 ensures that we can only have $\frac{n}{P}$ computations on any processor, and only d communication steps altogether (for some d). Then on the one hand, our construction contains a large gadget that occupies a single processor p for the entire schedule (otherwise the communication cost is too high), and d further gadgets that each need to send a value to p. Any schedule needs to manage these gadgets carefully to ensure that one of these d values is already available every time we do a communication step; this ensures that the communications can only happen at specific times at the earliest. On the other hand, we use critical paths to ensure that specific nodes need to be computed by a given time step. We then attach further gadgets to given segments of these paths, which forces us to split the work between multiple processors (and thus communicate) to satisfy these deadlines; hence communications must happen at specific times at the latest. Together, these define the exact times when we need to communicate, otherwise the cost exceeds C_0. Due to this, there is essentially only one way to develop a valid schedule in our DAG. The underlying reduction then uses the 3-partition problem with gadgets representing each input number a_i, and ensures that it is only possible to satisfy each communication deadline if the numbers a_i are sorted into triplets that each sum up to a given value.

6 Inapproximability

Given the NP-hardness of the problem, another follow-up question is whether we can at least approximate the optimal solution in polynomial time. However, we show that on general DAGs, BSP scheduling is APX-hard: there is a constant $\epsilon > 0$ such that it is already NP-hard to approximate the optimum cost to a $(1+\epsilon)$ factor. To our knowledge, for the case of constant P, no similar hardness results are known for other DAG scheduling problems.

Theorem 5. *BSP scheduling is APX-hard: there is a constant $\epsilon > 0$ such that it is NP-hard to approximate the optimum to a $(1+\epsilon)$ factor, already for $P = 2$.*

Proof sketch. We provide a reduction from MAX-3SAT(B), i.e. maximizing the satisfied clauses in a 3SAT formula where each variable occurs at most B times; this is already APX-hard for $B \in O(1)$ [49,1]. Our construction consists of a separate gadget for each clause and each variable of the given 3SAT formula. On a high level, the variable gadgets contain two separate subgadgets that we need to assign to the two different processors; this choice corresponds to setting the variable to true or false in the underlying formula. The clause gadgets then contain subgadgets corresponding to the 3 literals in the clause, which need to be assigned according to the truth value chosen in the corresponding variable gadget. Whenever a clause is unsatisfied, the corresponding gadgets in the schedule incur a slightly higher work cost than a satisfied clause. With this, the total

cost of the BSP schedule has a $\Theta(n)$ term that is proportional to the number of unsatisfied clauses, which allows us to complete the reduction.

The technical part of the proof is to show that any reasonable schedule in our DAG is indeed structured as described above, representing a solution of MAX-3SAT(B). Intuitively, our construction ensures that using any other kind of subschedule in our gadgets results in a higher work or communication cost. More formally, we can always apply a sequence of transformation steps to convert any other BSP schedule into a schedule that represents a valid 3SAT assignment, while only decreasing the cost of the solution.

7 Problem within a Problem: Communication Scheduling

Since h-relations are a complex communication metric, it is natural to wonder if the hardness of BSP scheduling lies only within the assignment of nodes to processors and steps (as in simpler models), or if the scheduling of communication steps also adds another layer of complexity on this. More specifically, assume that the nodes are already assigned to processors and supersteps, but we still need to decide when the values are communicated between the chosen processors; is this subproblem easier to solve?

Definition 2. *In the* communication scheduling (CS) problem*, π and τ are already fixed, and we need to find a Γ that minimizes the resulting BSP cost.*

We assume that π and τ allow a feasible communication schedule, i.e. $\forall (u, v) \in E$, we have $\tau(u) \leq \tau(v)$ if $\pi(u) = \pi(v)$, and $\tau(u) < \tau(v)$ if $\pi(u) \neq \pi(v)$.

As a heuristic approach, one might consider an eager or lazy communication policy, i.e. to communicate each value immediately after it is computed (eager), or only in the last possible superstep before it is needed (lazy). However, a both of these can easily be suboptimal; see the full version for a concrete example [42].

This shows that CS is indeed an interesting theoretical problem. Furthermore, from the practical side, CS has significantly less degree of freedom, so a heuristic approach could aim to first find a good initial schedule, and then try to improve this by fixing π and τ, and reducing communication cost separately.

For the simplest case of $P = 2$, a greedy approach already finds the optimal solution (in any model of Table 3, since these are all equivalent for $P = 2$).

Lemma 2. *The CS problem can be solved in polynomial time for $P = 2$.*

Interestingly, for general P, the hardness of CS may depend on the communication model: we prove that CS is already NP-hard in the DB, FS or FB models, but we leave it as an open question whether it is also NP-hard in DS.

Theorem 6. *The CS problem is NP-hard in the DB, FS and FB models.*

Proof sketch. The proof uses a reduction from 3D-matching: given sets X, Y, Z of size N, and M triplets from $X \times Y \times Z$, we need to select N triplets that form a disjoint cover. We convert this into a CS problem where each triplet is

represented by a value which needs to be sent from p_0 to several other processors, with the concrete deadlines depending on the given triplet. The construction consists of two parts. The first part allows us to send exactly $(M - N)$ values from p_0 to all other processors (the triplets that are not chosen) within the allowed cost. The second part then allows us to send N further values from p_0 to the other processors, but only if the corresponding triplets are disjoint.

We also note that if we consider a natural extension of the problem with node weights (see the full version for details), then it becomes relatively simple to reduce CS to standard packing problems, already for $P = 2$ processors.

Lemma 3. *With communication weights, CS is already NP-hard for* $P = 2$.

8 Formulation as an ILP Problem

Finally, we discuss a straightforward approach to formulate BSP scheduling as an Integer Linear Programming (ILP) problem. Since today's ILP solvers can often solve even fairly large instances in reasonable time, this naive ILP formulation may already be a viable approach in several applications, especially if the computation is modelled as a DAG at relatively coarse granularity. A similar approach with ILP solvers has already provided remarkable empirical results for various combinatorial problems [22]. Our ILP formulation also generalizes very naturally to several model extensions, e.g. with node weights.

Proposition 2. *BSP scheduling can be formulated as an ILP on* $O(n \cdot P \cdot S)$ *variables in models DS, DB and FB, and on* $O(n \cdot P^2 \cdot S)$ *variables in model FS.*

References

1. Berman, P.R., Scott, A., Karpinski, M.: Approximation hardness and satisfiability of bounded occurrence instances of sat. Tech. rep., SIS-2003-269 (2003)
2. Bisseling, R.H.: Parallel Scientific Computation: A Structured Approach Using BSP. Oxford University Press, USA (2020)
3. Bodlaender, H.L., Fellows, M.R.: W [2]-hardness of precedence constrained k-processor scheduling. Operations Research Letters **18**(2), 93–97 (1995)
4. Chen, L., Jansen, K., Zhang, G.: On the optimality of approximation schemes for the classical scheduling problem. In: Proceedings of the 25th annual ACM-SIAM symposium on Discrete algorithms (SODA). pp. 657–668. SIAM (2014)
5. Coffman, E.G., Graham, R.L.: Optimal scheduling for two-processor systems. Acta informatica **1**(3), 200–213 (1972)
6. Culler, D., Karp, R., Patterson, D., Sahay, A., Schauser, K.E., Santos, E., Subramonian, R., Von Eicken, T.: Logp: Towards a realistic model of parallel computation. In: Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming (PPoPP). pp. 1–12 (1993)
7. Davies, S., Kulkarni, J., Rothvoss, T., Sandeep, S., Tarnawski, J., Zhang, Y.: On the hardness of scheduling with non-uniform communication delays. In: ACM-SIAM Symposium on Discrete Algorithms (SODA). pp. 316–328. SIAM (2022)

8. Davies, S., Kulkarni, J., Rothvoss, T., Tarnawski, J., Zhang, Y.: Scheduling with communication delays via lp hierarchies and clustering. In: 61st Annual Symposium on Foundations of Computer Science (FOCS). pp. 822–833. IEEE (2020)
9. Davies, S., Kulkarni, J., Rothvoss, T., Tarnawski, J., Zhang, Y.: Scheduling with communication delays via lp hierarchies and clustering ii: weighted completion times on related machines. In: ACM-SIAM Symposium on Discrete Algorithms (SODA). pp. 2958–2977. SIAM (2021)
10. Dolev, D., Warmuth, M.K.: Scheduling precedence graphs of bounded height. Journal of Algorithms **5**(1), 48–59 (1984)
11. Dolev, D., Warmuth, M.K.: Profile scheduling of opposing forests and level orders. SIAM Journal on Algebraic Discrete Methods **6**(4), 665–687 (1985)
12. Du, J., Leung, J.Y., Young, G.H.: Scheduling chain-structured tasks to minimize makespan and mean flow time. Information and Computation **92**(2), 219–236 (1991)
13. Fujimoto, N., Hagihara, K.: On approximation of the bulk synchronous task scheduling problem. IEEE Transactions on Parallel and Distributed Systems **14**(11), 1191–1199 (2003)
14. Gabow, H.N.: An almost-linear algorithm for two-processor scheduling. Journal of the ACM **29**(3), 766–780 (1982)
15. Garey, M., Johnson, D., Tarjan, R., Yannakakis, M.: Scheduling opposing forests. SIAM Journal on Algebraic Discrete Methods **4**(1), 72–93 (1983)
16. Garg, S.: Quasi-ptas for scheduling with precedences using LP hierarchies. In: 45th International Colloquium on Automata, Languages, and Programming (ICALP 2018). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2018)
17. Goldman, A., Mounié, G., Trystram, D.: Near optimal algorithms for scheduling independent chains in bsp. In: Proceedings. 5th International Conference on High Performance Computing. pp. 310–317. IEEE (1998)
18. Graham, R.L., Lawler, E.L., Lenstra, J.K., Kan, A.R.: Optimization and approximation in deterministic sequencing and scheduling: a survey. In: Annals of discrete mathematics, vol. 5, pp. 287–326. Elsevier (1979)
19. Hanen, C., Munier, A.: An approximation algorithm for scheduling dependent tasks on m processors with small communication delays. In: Proceedings 1995 INRIA/IEEE Symposium on Emerging Technologies and Factory Automation. ETFA'95. vol. 1, pp. 167–189. IEEE (1995)
20. Hill, J.M., McColl, B., Stefanescu, D.C., Goudreau, M.W., Lang, K., Rao, S.B., Suel, T., Tsantilas, T., Bisseling, R.H.: BSPlib: The BSP programming library. Parallel Computing **24**(14), 1947–1980 (1998)
21. Hoogeveen, J., Lenstra, J.K., Veltman, B.: Three, four, five, six, or the complexity of scheduling with communication delays. Operations Research Letters **16**(3), 129–137 (1994)
22. Jenneskens, E.L., Bisseling, R.H.: Exact k-way sparse matrix partitioning. In: 2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). pp. 754–763. IEEE (2022)
23. Kulkarni, J., Li, S., Tarnawski, J., Ye, M.: Hierarchy-based algorithms for minimizing makespan under precedence and communication constraints. In: Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). pp. 2770–2789. SIAM (2020)
24. Lenstra, J.K., Kan, A.R., Brucker, P.: Complexity of machine scheduling problems. In: Annals of discrete mathematics, vol. 1, pp. 343–362. Elsevier (1977)
25. Lenstra, J.K., Rinnooy Kan, A.: Complexity of scheduling under precedence constraints. Operations Research **26**(1), 22–35 (1978)

26. Lenstra, J.K., Shmoys, D.B., Tardos, É.: Approximation algorithms for scheduling unrelated parallel machines. Mathematical programming **46**(1), 259–271 (1990)
27. Leung, J.Y.T., Young, G.H.: Minimizing total tardiness on a single machine with precedence constraints. ORSA Journal on Computing **2**(4), 346–352 (1990)
28. Levey, E., Rothvoss, T.: A (1+ epsilon)-approximation for makespan scheduling with precedence constraints using lp hierarchies. In: Proceedings of the 48th annual ACM symposium on Theory of Computing (STOC). pp. 168–177 (2016)
29. Li, S.: Scheduling to minimize total weighted completion time via time-indexed linear programming relaxations. In: IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS). pp. 283–294. IEEE Computer Society (2017)
30. Li, S.: Towards PTAS for precedence constrained scheduling via combinatorial algorithms. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA). pp. 2991–3010. SIAM (2021)
31. Liu, Q.C., Purohit, M., Svitkina, Z., Vee, E., Wang, J.R.: Scheduling with communication delay in near-linear time. In: 39th International Symposium on Theoretical Aspects of Computer Science (STACS 2022). Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2022)
32. Maiti, B., Rajaraman, R., Stalfa, D., Svitkina, Z., Vijayaraghavan, A.: Scheduling precedence-constrained jobs on related machines with communication delay. In: 61st Annual Symposium on Foundations of Computer Science (FOCS). pp. 834–845. IEEE (2020)
33. Markenscoff, P., Li, Y.Y.: Scheduling a computational dag on a parallel system with communication delays and replication of node execution. In: Proceedings of the 7th International Parallel Processing Symposium. pp. 113–117. IEEE (1993)
34. McColl, B.: Mathematics, Models and Architectures, p. 6–53. Cambridge University Press (2021)
35. McColl, W.F.: Scalable computing. Computer Science Today pp. 46–61 (1995)
36. McColl, W.F., Tiskin, A.: Memory-efficient matrix multiplication in the BSP model. Algorithmica **24**(3), 287–297 (1999)
37. Mnich, M., Wiese, A.: Scheduling and fixed-parameter tractability. Mathematical Programming **154**(1), 533–562 (2015)
38. Munier, A., König, J.C.: A heuristic for a scheduling problem with communication delays. Operations Research **45**(1), 145–147 (1997)
39. Özkaya, M.Y., Benoit, A., Uçar, B., Herrmann, J., Çatalyürek, Ü.V.: A scalable clustering-based task scheduler for homogeneous processors using DAG partitioning. In: IEEE International Parallel and Distributed Processing Symposium (IPDPS). pp. 155–165. IEEE (2019)
40. Papadimitriou, C., Yannakakis, M.: Towards an architecture-independent analysis of parallel algorithms. In: Proceedings of the 20th annual ACM symposium on Theory of computing (STOC). pp. 510–513 (1988)
41. Papp, P.A., Anegg, G., Yzelman, A.J.N.: Partitioning hypergraphs is hard: Models, inapproximability, and applications. In: Proceedings of the 35th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA). pp. 415–425 (2023)
42. Papp, P.A., Anegg, G., Yzelman, A.N.: DAG scheduling in the BSP model. arXiv preprint arXiv:2303.05989 (2024)
43. Rajaraman, R., Stalfa, D., Yang, S.: Approximation algorithms for scheduling under non-uniform machine-dependent delays. arXiv preprint arXiv:2207.13121 (2022)
44. Sanders, P., Mehlhorn, K., Dietzfelbinger, M., Dementiev, R.: Sequential and Parallel Algorithms and Data Structures. Springer (2019)

45. Sethi, R.: Scheduling graphs on two processors. SIAM Journal on Computing **5**(1), 73–82 (1976)
46. Skillicorn, D.B., Hill, J., McColl, W.F.: Questions and answers about BSP. Scientific Programming **6**(3), 249–274 (1997)
47. Svensson, O.: Conditional hardness of precedence constrained scheduling on identical machines. In: Proceedings of the 42nd ACM symposium on Theory of computing (STOC). pp. 745–754 (2010)
48. Timkovsky, V.G.: Identical parallel machines vs. unit-time shops and preemptions vs. chains in scheduling complexity. European Journal of Operational Research **149**(2), 355–376 (2003)
49. Trevisan, L.: Non-approximability results for optimization problems on bounded degree instances. In: Proceedings of the thirty-third annual ACM symposium on Theory of computing. pp. 453–461 (2001)
50. Ullman, J.D.: Np-complete scheduling problems. Journal of Computer and System sciences **10**(3), 384–393 (1975)
51. Valiant, L.G.: A bridging model for parallel computation. Communications of the ACM **33**(8), 103–111 (1990)
52. Valiant, L.G.: A bridging model for multi-core computing. In: European Symposium on Algorithms (ESA). pp. 13–28. Springer (2008)
53. Valiant, L.G.: A bridging model for multi-core computing. Journal of Computer and System Sciences **77**(1), 154–166 (2011)
54. Veltman, B., Lageweg, B., Lenstra, J.K.: Multiprocessor scheduling with communication delays. Parallel computing **16**(2-3), 173–182 (1990)
55. Yzelman, A., Bisseling, R.H.: An object-oriented bulk synchronous parallel library for multicore programming. Concurrency and Computation: Practice and Experience **24**(5), 533–553 (2012)
56. Yzelman, A., Bisseling, R.H., Roose, D., Meerbergen, K.: MulticoreBSP for C: a high-performance library for shared-memory parallel programming. International Journal of Parallel Programming **42**(4), 619–642 (2014)