Near optimal algorithms for scheduling independent chains in BSP*

Alfredo GOLDMAN,† Grégory MOUNIÉ, Denis TRYSTRAM
LMC – IMAG
BP 53, F-38041 Grenoble Cedex
France

Abstract

The aim of this work is to show that scheduling a set of independent chains on a parallel machine under the BSP model is a difficult optimization problem which can be easily approximated in practice. BSP is a machine independent computational model which is becoming more and more popular [7]. Finding the optimal solution when the number of processors is fixed is shown to be hard. Efficient heuristics including communications are proposed and analyzed. We particularly focus on the influence of synchronization between consecutive supersteps. Simulations of a large number of instances have been carried out to complement the theoretical worst case analysis. They confirm the very good behavior of the algorithm on average.

1 Introduction

This paper studies the problem of scheduling independent chains on identical parallel processors. The chains are composed of tasks that have to be executed sequentially.

The processors are organized as a parallel distributed-memory machine, consisting of m processors of identical speed. We consider a machine-independent programming model based on *BSP* (Bulk Synchronous Parallel) [7]. Two main parameters can be considered in BSP, namely, the global synchronizations and the latency between two consecutive supersteps.

Our goal in this work is to show that efficient scheduling algorithms can be designed under the BSP model. Intrinsically, the problem under the BSP model is more difficult than in the models in which the communications are not necessarily synchronized among supersteps.

The organization of the paper is the following: In section 2, some definitions are introduced and preliminary results for scheduling a set of independent chains on other computational models are discussed. Then, this problem is solved in section 3 for two extreme cases, namely for two processors, and for an unlimited number of processors. Complexity results and approximation algorithms are presented in section 4 for the general case of a fixed number of processors. Then, we propose an algorithm with a limited number of supersteps. Section 5 is devoted to experiments with simulations. Finally, some perspectives for future work are discussed in the conclusion.

*This research is part of the Apache project supported by CNRS, INPG, INRIA and UJF.
†Supported by CNPq. On leave from DCC-IME-USP Brazil

2 Preliminaries

In this section we present the target problem of scheduling chains and recall the definition of BSP. Some preliminary results assuming standard computational models, useful for the understanding of the algorithms, are also described.

2.1 Description of the problem

The problem that we consider in this paper is to schedule k independent chains of tasks on m identical processors under the BSP model. We will call this problem *SIC* for Scheduling Independent Chains. The execution time of each task takes one unit of time (UET assumption). Chains of length n_i will be denoted ch_i (for $1 \leq i \leq k$). We consider the case $k \geq 2$, since otherwise the problem is trivial and the best strategy is to schedule the entire single chain on one processor without communication. The total number of tasks is $n = \sum_{i=1}^{k} n_i$. Without loss of generality, the chains are assumed to be initially sorted in decreasing order of length, that is, $n_1 \geq \ldots \geq n_i \geq \ldots \geq n_k$.

We are interested in minimizing the *makespan*, which is the maximum completion time of the tasks of a parallel algorithm. We present below a preliminary proposition which establishes a straightforward lower bound.

Proposition 1 *The workload for SIC on m processors is at least $WL = \max\{\lceil \frac{n}{m} \rceil, n_1\}$.*

Proof: This bound is a lower bound on the minimum possible time to finish the execution of all chains. The workload is the maximum of the total workload distributed among the processors and the length of the longest chain. ❑

Observe that the optimal workload can not be achieved in every case without splitting chains (which introduces communications between the different parts of a chain). Consider for instance the execution of three chains of same length on two processors as shown in figure 1. (Here and in all other figures, the dashed area represents idle times.) This remark motivates the following definition:

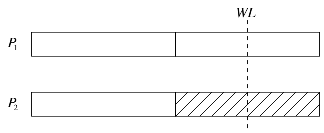

Figure 1. Scheduling of 3 identical chains.

Definition 1 *The best possible schedule, without communication, to complete the execution of all the chains is the schedule with the smallest time. The completion time of this schedule will be denoted ω_o.*

It is straightforward that $\omega_0 \geq WL$.

Finally, it is interesting to observe that in the SIC problem, the task replication is not useful, as each task has at most one task that depends on it.

2.2 Presentation of BSP

BSP (for Bulk Synchronous Parallel) is a computational model introduced for improving the scalability, the portability and the ease of developing application code on distributed-memory parallel systems [7]. It distinguishes between the two key aspects of performance: computation and communication. BSP was introduced in order to isolate the communication problems from the scheduling difficulties. It is becoming more and more popular for theoretical studies, providing a solid foundation for designing parallel algorithms. Some implementations exist on several platforms [3].

2.2.1 Definition

Programming in BSP consists of a succession of supersteps. Each superstep is divided into three activities:
- computation, local computations run in parallel;
- communication, data to be used in the next supersteps;
- synchronization, all processors start the next superstep simultaneously.

The synchronization guarantees the completion of communication and computation of the superstep. There are two mains forms to perform the communication: asynchronously or as an efficient h-relation [2].

There are two intrinsic restrictions in BSP: each processor can send/receive at most a constant number of messages during a superstep, and the messages have a limited size. With these restrictions, using the g and l parameters of the BSP model, it is possible to bound the time for communication and synchronization in a superstep. In all algorithms presented in this paper, each processor sends and receives at most one communication (1-relation), so we can assume that synchronization takes a constant time denoted by C.

The original model allows supersteps where the synchronization is done only among subsets of processors. However, the actual current implementations only provide global synchronization. See [4] for a complete discussion.

There are two main parameters which characterize the model: the time to synchronize the processors, C (a positive real number usually greater than the execution time, $C > 1$) and the minimum time between two consecutive supersteps, denoted λ (of course, $\lambda \geq C$). As said before, to simplify the analysis, for each superstep (except the last one), there is a global synchronization with time C.

2.2.2 Example

The example depicted in figure 2 shows an execution under BSP for a simple precedence task graph. (This is the usual representation of a parallel program: The instructions are represented by nodes, and the precedence relations among the instructions are represented by arcs.) On the Gantt chart, the communications are represented in grey, the synchronization phase in black.

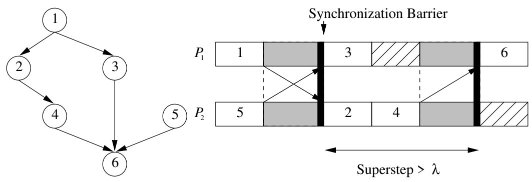

Figure 2. Example of a schedule under BSP.

2.3 SIC under basic computational models

Many papers addressing the problem of scheduling a set of independent chains appear in the literature. It is of some practical interest since it can model the problem of distributing the execution of sequential work (for instance, while using library components).

Most of these papers assume simple computational models. Either communication are ignored, or they are taken into account with the standard scheduling model introduced by Rayward-Smith [6], in which there is no synchronization among the communications, and the transmission of messages can overlap computations. Communications are assumed to be constant and equal to the elementary computational time (UET-UCT assumption).

2.3.1 An asynchronous algorithm

We present in this section a simple algorithm for SIC under the standard asynchronous model of Rayward-Smith. The principle consists of filling the processors one after the other, from left to right, until time WL.

When a chain does not fit entirely on a processor (say, P_1), it is split and a communication is introduced between its two parts. According to the precedence relations: the first part starts on P_2 and the last part finishes on P_1, each part is allocated at each extreme side of the Gantt chart. See figures 3 which illustrate the principle.

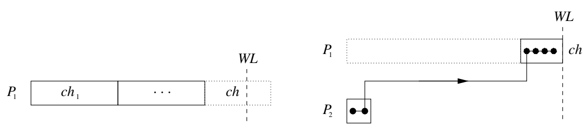

Figure 3. Greedy and split chain allocation.

Require: The chains are sorted in decreasing order
Require: $load(P_j)$ compute the number of tasks allocated to processor P_j
  $i \leftarrow 1, j \leftarrow 1$
  while $i \leq k$ **do** {it remains some chains to allocate}
    while $load(P_j) + n_i \leq WL$ and $i \leq k$ **do**
      allocate ch_i on P_j
      $i \leftarrow i + 1$
    end while
    if $load(P_j) \neq WL i \leq k$ **then**
      allocate first $n_i - (WL - load(P_j))$ tasks of ch_i on P_{j+1}
      allocate last $WL - load(P_j)$ tasks of ch_i on P_j
      $i \leftarrow i + 1$
    end if
    $j \leftarrow j + 1$
  end while

In the standard UET-UCT model, it is straightforward to show that the previous algorithm gives a valid allocation with makespan WL. The key point is that a communication can always be performed between the two parts of a split chain (because it has less than WL tasks).

2.3.2 Our contribution

In this paper, we focus our attention on scheduling specific precedence task graphs which consist of a set of independent chains of UET tasks. This paper is one of the first studies of scheduling under BSP.

Note that some other related work deal with the scheduling problem of specific task graphs such as tree structures and FORK under computational models such as LogP [8, 5].

3 Two preliminary cases

In this section, we present some results for scheduling k independent chains for two extreme cases, namely two processors, and an unlimited number of processors.

3.1 SIC on two processors in BSP

First, we study the problem of scheduling independent chains on two processors. We first show that it is difficult to find a schedule with makespan WL. Then we propose an algorithm that generates a schedule with a makespan no greater than $WL + C$, that is with at most two supersteps.

3.1.1 Theoretical analysis

Lemma 1 *Determining the existence of a schedule of independent chains on two identical processors with makespan WL is NP-complete.*

Proof: In the presence of communications, any schedule will have a makespan greater than WL, so we look for a schedule within only one superstep.

The problem is in NP, as a non-deterministic algorithm only needs to guess a subset CH of the set of chains, and to check in polynomial time that the sum of the chain lengths in subset CH is WL. We recall that the following *Partition* problem is NP-Complete [1], but pseudo-polynomial:

- Instance : a set X of elements of integer length n_i, an integer B such that $\sum n_i = B$.
- Question : does a subset of total length $\frac{B}{2}$ exist ?

It is straightforward to transform this problem to the SIC problem on two identical processors. Given a partition problem, consider a problem of scheduling k independent chains of length $n_i, 1 \leq i \leq k$ and with $WL = \lceil \frac{B}{2} \rceil$. A schedule with makespan WL exists iff a set of total length $\frac{B}{2}$ exists. ❑

Moreover, a schedule without communication can be "far" from WL. For instance, consider again the example of section 2.2. with three chains of the same length. The best schedule without communication has length $\omega_o = \frac{4}{3} WL$.

3.1.2 Algorithm

The algorithm given below determines a schedule for SIC on two processors. It is based on the general asynchronous algorithm of section 2.3.1.

Require: $load(P_1)$ compute the number of tasks allocated to processor P_1
  $i \leftarrow 1$
  while $load(P_1) + n_i \leq WL$ and $i \leq k$ **do**
    allocate ch_i on P_1
    $i \leftarrow i + 1$
  end while
  if $load(P_1) \neq WLi \leq k$ **then**
    allocate first $n_i - (WL - load(P_1))$ tasks of ch_i on P_2
    allocate last $WL - load(P_1)$ tasks of ch_i on P_1
    synchronize the superstep at time $n_i - (WL - load(P_1))$
    $i \leftarrow i + 1$
  end if
  while $i \leq k$ **do**
    allocate ch_i on P_2
    $i \leftarrow i + 1$
  end while

There is no need to sort the chains before the algorithm.

3.1.3 Example

In this section, we give an example with 4 chains of lengths $10, 10, 5$ and 5 on two processors. The time of a communication-synchronization is $C = 2$.

We first compute $WL = 15$. Initially, the first processor is filled to at least time WL, which corresponds to two chains with a makespan of 20 (see fig. 4).

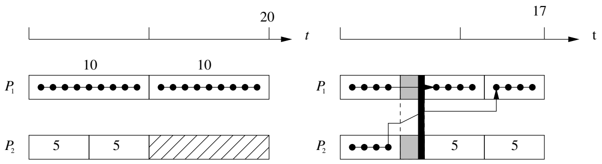

Figure 4. Bad and BSP algorithm allocations.

Due to the addition of one end of superstep, 5 tasks of the second chain are moved to the second processor. With one synchronization the makespan becomes $17 = WL + C$, and both processors are fully occupied (see fig. 4).

3.1.4 Analysis

Proposition 2 *The time of the schedule generated by the previous algorithm is at most $WL + C$.*

Proof: By construction, it is obvious that at most one chain is split by the previous algorithm. Then, there is at most one communication.

As the biggest chain has at most WL tasks, a communication and synchronization can be introduced between the two parts of the split chain, if it exists.

So the previous algorithm gives a valid schedule with processing time WL and at most one synchronization. ❑

Remark: The proposed algorithm does not always generate an optimal schedule. In some cases, a schedule with time at most WL without communication exists (figure 5). But, as proved before, knowing if such a schedule exists is an NP-complete problem.

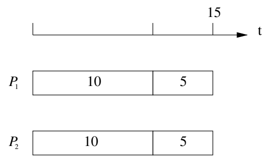

Figure 5. Schedule that reaches WL.

3.2 Unlimited number of processors

With identical processors, we have an oblivious algorithm. Given k chains, ch_i, $1 \leq i \leq k$, each one is allocated to a separate processor. The makespan is given by the length of the longest chain, n_1. It is unnecessary to use more than k processors because each chain has to be executed sequentially. There may be a solution with fewer processors, but it is not possible to improve the makespan.

Finding the optimal schedule without communication with the minimum number of processor is also NP-Hard. It is equivalent to the problem of solving a $1D$-Bin-Packing problem [1], with bins of length n_1.

4 Fixed number of processors

In this section, we study the extension to the case of a fixed number of processors. We proved in the last section that SIC is already difficult for two processors, but there exists an almost optimal algorithm which guarantees only an additive factor of C from the optimal.

We first discuss the complexity of the problem on $m > 2$ processors. (Of course, $k > m$, otherwise the problem is straightforward.) Then, we present a worst case analysis which shows that $\lceil \frac{m-1}{2} \rceil$ supersteps may be required for obtaining a schedule in which each processor has to execute at most WL tasks. Then, we propose a schedule with processing time WL and at most $\lceil \frac{m+1}{2} \rceil$ supersteps. In this algorithm, the end of supersteps can occur at any time.

4.1 Theoretical complexity

In this section, we are interested in SIC with the minimum theoretical time (WL). Given a set of chains, it is possible that there is no schedule without communication with time WL. But, even if such a schedule exists, it is difficult to find.

Proposition 3 *SIC, within the minimum theoretical time, on a fixed number of processors without communication, is NP-hard in the strong sense.*

Proof: The main idea is to use a reduction from the numerical 3-dimensional matching problem [1] which is recalled below: **N3DM problem:**

- Instance : Given three sets, $X = \{x_1, \ldots, x_N\}$, $Y = \{y_1, \ldots, y_N\}$ and $Z = \{z_1, \ldots, z_N\}$. Each element $x \in X$ (respectively $y \in Y$ and $z \in Z$) has a positive integer weight $s(x)$ (respectively $s(y)$ and $s(z)$). Let B be a positive integer.
- Question : Find N disjoint 3-partitions of $X \cup Y \cup Z$ such that in each partition $\{x_i, y_j, z_k\}$ the sum of the weights is equal to B.

This problem is known to be NP-hard in the strong sense [1]. In the presence of communications (more than one superstep), any schedule will have a makespan larger than WL ($C > 0$), so we look for a schedule without communication.

It is easy to show that the problem is in NP, since a non-deterministic algorithm needs only to guess an m-partition $(S_1, \ldots, S_m)$ of the set of chains, and to check in polynomial time that the sum of the chain lengths in each subset $S_j, 1 \leq j \leq m$ is at most WL.

To do the reduction, we will schedule $3N$ chains on N processors. The length of each chain is defined as follows:

$n_{x_i} = 8B + s(x_i), 1 \leq i \leq N,$

$n_{y_i} = 4B + s(y_i), 1 \leq i \leq N,$

$n_{z_i} = 2B + s(z_i), 1 \leq i \leq N.$

The total number of tasks is $N(8B + 4B + 2B) + \sum_{i=1}^{N} s(x_i) + s(y_i) + s(z_i) = 15NB$ because if $\sum_{i=1}^{N} s(x_i) + s(y_i) + s(z_i) \neq NB$, the given instance can not be partitioned into sets of size B.

The best possible makespan is $15B$. We will show that if there exists a schedule of length at most $15B$, then, we have a solution for the N3DM problem.

There is exactly one chain of each type (x, y and z) on each processor: each processor owns one chain of length n_{x_i}. (It is not possible to allocate more than one such chains per processor, as there are N processors and N such chains.) Taking into account the chains of length n_{x_i} fewer than $7B$ available time slots remains, thus there is at most one chain of length n_{y_j} per processor. The same holds for the chains of length n_{z_k}.

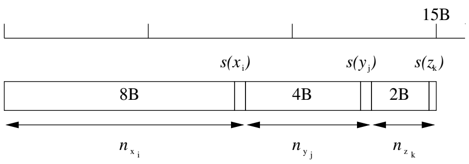

Figure 6. Allocation on one processor.

Each processor has exactly three chains, one of each type x_i, y_j, z_k, and $n_{x_i} + n_{y_j} + n_{z_k} = 15B$, that is $s(x_i) + s(y_j) + s(z_k) = B$. In any optimal schedule without communication, there is exactly one element from each set on each processor, and the sum of these elements is always equal to B. Thus, there is exactly one solution for the N3DM problem. ❑

Corollary 1 *Given the superstep synchronization time C, the problem of finding an optimal schedule, if such a schedule exists, with length less than $WL + C$ is NP-hard.*

Proof: The idea is to transform the N3DM problem into α-N3DM, by multiplying the weights from the original N3DM problem by any integer α such that $C < \alpha$.❑

4.2 Worst case for the number of supersteps

In this section, we present an instance for which any schedule needs $\lceil \frac{m-1}{2} \rceil$ supersteps for SIC on m processors in order to eliminate the idle time on all processors. We will prove this result by constructing valid schedules with, respectively, one, two, ..., $\lceil \frac{m-1}{2} \rceil$ supersteps for the case of $m + 1$ chains of length S.

In order to simplify the analysis, we suppose that $\frac{(m+1)S}{m}$ is an integer. Observe that for a large number of processors, WL tends to $S + 1$.

With only one superstep, using any allocation scheme, there will be a processor with at least two chains, so the best possible time is $\omega_o = 2S$. A best possible schedule is to allocate two chains on one processor, and one chain on each other processor. This allocation reaches the bound ω_o.

Ideally, to obtain a schedule with makespan WL, ignoring the communication time, all the processors must work all the time. All chains are strictly shorter than WL, so each processor needs to execute tasks from at least two chains in order to be fully occupied until WL. Otherwise at least one processor would stop strictly before WL. To fully occupy all the processors, some chains will start on one processor and finish on another one, thus communications are required.

Case of two supersteps

Claim 1 *Within one superstep longer than $\frac{S}{m}$, the best strategy is to execute tasks from two different chains on one processor and one chain on each other processor.*

Proof: If there is a chain that is not executed during this superstep, the best makespan will be the size of this superstep plus S ($> WL$). Otherwise if all chains are executed, a better makespan is possible. Since there are $m+1$ chains and m processors, there will be a processor with two chains. ❑

It is easy to verify that the chains that share a processor in one superstep have to be alone in the other superstep in order to maximize the number of chains shared. For instance in figure 7, chains ch_1, ch_2 and ch_3, ch_4 respectively are together in the first and second supersteps.

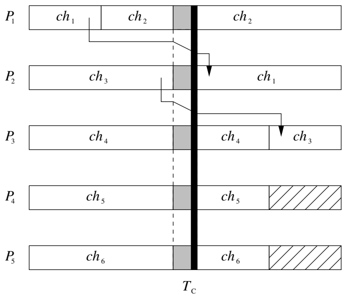

Figure 7. Worst case with two supersteps.

Claim 2 *In a superstep, when two chains are being executed on the same processor, half of the available time is used for each chain.*

Proof: All the chains have the same length. Given two chains that are sharing a processor, let x be the number of tasks from one of the chains that are executed during a superstep. Let T_C be the starting time of a superstep. Without taking into account the BSP-communications, we have to minimize $\max\{T_c + (S - x), T_c + (S - (T_c - x))\}$. This is achieve when $x = \frac{T_c}{2}$.❑

Now, we have to determine T_c. Without loss of generality, we suppose that the chains that share a processor in the first superstep are ch_1 and ch_2, and in the next superstep ch_3 and ch_4. The time when ch_1 and ch_2 will be completed is $T_c + (S - \frac{T_c}{2}) = S + \frac{T_c}{2}$. For ch_3 and ch_4, the finishing time is $2(S - T_c) + T_c = 2S - T_c$. For all other chains the time is S, which is less than the workload WL. So, to minimize the time, we have to minimize $\max\{S + \frac{T_c}{2}, 2S - T_c\}$. But, as one factor increases with T_c, and the other one decreases, the minimum is achieved for $S + \frac{T_c}{2} = 2S - T_c$. That is, $T_c = \frac{2S}{3}$, and the minimum time is $\frac{4S}{3}$. The time to solve the given case with two supersteps is $S + C + \frac{S}{3}$. Observe that for $m = 3$, $S + C + \frac{S}{3}$ is the workload plus one synchronization time.

More than two supersteps

In this section, we extend the previous scheduling algorithm to three or more supersteps. From claim 1, there are i regions in which two chains can share the same processor for i supersteps. By a straightforward extension of claim 2, these chains are split in the middle of the supersteps. The successive superstep lengths are easily computed by induction as a series of T_c^j (for $1 \leq j \leq i$). The minimum possible time is $S + iC + \frac{S}{2i+1}$ with the supersteps ending at times $2j\frac{S}{2i+1} + jC$, $1 \leq j \leq i$ for small communication times. This result can be proved by induction on i.

So, to eliminate the idle time on all processors, we have to communicate $z = \lceil \frac{m-1}{2} \rceil$ times, and the end of supersteps are evenly distributed at distance $2S/m + C$ from each other. The time for the given example is $S + zC + \frac{S}{2z+1}$, that is $WL + zC$. This bound is valid for odd m. For even m there will be some additional rounding terms which do not affect the main result.

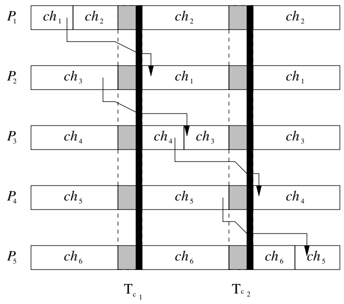

Figure 8. Three supersteps scheduling.

4.3 Limited number of supersteps algorithm

Given k chains and m processors, we present in this section an algorithm to find an approximate solution with at most $\lceil \frac{m+1}{2} \rceil$ supersteps. This may be required for some instances as we showed in the last section.

We consider the asynchronous algorithm of section 2.3.1. We show that it is possible to group at least two asynchronous communications into the same BSP superstep.

Apply the asynchronous algorithm
while some asynchronous communications remain **do**
  Find the first task receiving a communication
  Place an end of superstep just before the task
  Do all asynchronous communications starting before the end of this superstep in it
end while

4.3.1 Algorithm

4.3.2 Analysis

First, the chains are partitioned into three groups :
- Group A of chains of size WL,
- Group B of chains of size between WL and $\lceil \frac{WL}{2} \rceil + 1$,
- Group C of chains of size at most $\lceil \frac{WL}{2} \rceil$.

These groups will be scheduled one after the other.

Proposition 4 *The chains of group A can be scheduled within a same superstep. The chains in group C can be scheduled using only two supersteps where the first one finishes at time $\lfloor \frac{WL}{2} \rfloor$.*

Proof: Each chain of group A is allocated on one processor. Thus, these processors work for WL units of time.

Given any chain ch of group C, we know that ch is smaller than $\lceil \frac{WL}{2} \rceil$. A schedule using the asynchronous algorithm of section 2.3.1 can allocate ch to either one or two processors. If ch is allocated to only one processor there is no need of communication. Otherwise, ch begins on one processor and finishes on the previous one. By the allocation, the first part of ch can not finish after $\lfloor \frac{WL}{2} \rfloor$, and its end part can not start before $\lfloor \frac{WL}{2} \rfloor$. Thus all the chains of group C can be scheduled by the asynchronous algorithm and need only one BSP end of superstep at time $\lfloor \frac{WL}{2} \rfloor$. ❑

Problems only occur while scheduling chains of group B. The idea for reducing the number of supersteps is to group several supersteps into one BSP superstep. In order to prove that the number of supersteps is at most $\lceil \frac{m+1}{2} \rceil$, we state the following lemma.

Lemma 2 *Given two chains ch_{i_j} and $ch_{i_{j+1}}$, each with more than $\lceil \frac{WL}{2} \rceil + 1$ tasks that are executed in the same processor, either the communications can be done during the same BSP superstep, or the previous chain ($ch_{i_{j-1}}$) can use the same BSP superstep as chain ch_{i_j}. If the next chain $ch_{i_{j+2}}$ after chain $ch_{i_{j+1}}$ has more than $\lceil \frac{WL}{2} \rceil$ tasks, these two chains can share the same BSP communication.*

Proof: Consider two chains ch_{i_j} and $ch_{i_{j+1}}$ of respective lengths $n_{i_j} = WL - x$ and $n_{i_{j+1}} = WL - y$ with both $x < \lfloor \frac{WL}{2} \rfloor$ and $y < \lfloor \frac{WL}{2} \rfloor$, and let $j+1$ be the processor where they are allocated. Processor $j+1$ has at most one chain of group B allocated entirely to it (because all chains before $ch_{i_{j+1}}$ have more than $\lceil \frac{WL}{2} \rceil + 1$ tasks).

Let w_j be the number of tasks from ch_{i_j} that are executed on processor $j+1$, and w_{j+1} be the number of tasks from $ch_{i_{j+1}}$ executed on processor $j+2$. There are two possibilities:

1. Processor $j+1$ executes only tasks from chains ch_{i_j} and $ch_{i_{j+1}}$.

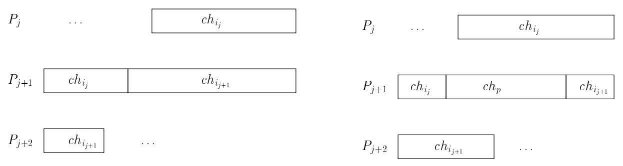

Figure 9. Possibility one and Two.

   The possible interval for the end of the superstep between the two parts of ch_{i_j} is $[w_j, WL-(n_{i_j}-w_j)] = [w_j, x+w_j]$. For $ch_{i_{j+1}}$ it is $[w_{j+1}, WL-(n_{i_{j+1}} - w_{j+1})] = [w_{j+1}, y+w_{j+1}]$. There is always a non-empty intersection because $w_j = y + w_{j+1}$.

2. Processor $j+1$ also executes another chain ch_p.

   Let n_p be the length of ch_p. The interval for communication between the two parts of ch_{i_j} is $[w_j, WL - (n_{i_j} - w_j)] = [w_j, x + w_j]$. For $ch_{i_{j+1}}$, it is $[w_{j+1}, WL - (n_{i_{j+1}} - w_{j+1})] = [w_{j+1}, y + w_{j+1}]$. To have a non-empty intersection we must have:

   $x + w_j \geq w_{j+1}$
   $x + w_j \geq (w_j + n_p + n_{i_{j+1}}) - WL$
   $x + y \geq n_p$.

   If $x + y \geq n_p$, we have an intersection. Otherwise, $x + y < n_p$, as $n_p \geq WL - y$, and we have $x + 2y < WL$. As $x < y$, $3x < WL$, and so, $n_{i_j} > \frac{2WL}{3}$. In this case, either ch_j or ch_{j+1} can have its communication at instant $\lfloor \frac{WL}{2} \rfloor$, or there is no chain entirely allocated to processor j. Thus, we can group the communications on processors $j-1$ and j within one common BSP superstep. If the next chain allocated to processor $j+1$ has more than $\lceil \frac{WL}{2} \rceil$ tasks, then no other chain can be entirely allocated to processor $j+2$. In this case, we can group communications on processors $j+1$ and $j+2$. (If this happens several times, there are chains allocated entirely on processors $j+1, j+3, \ldots$, as represented in figure 10.) ❑

We can now state the main result of this section:

Proposition 5 *It is always possible to group the communications of the asynchronous algorithm for k chains and m processors within at most $\lceil \frac{m+1}{2} \rceil$ BSP supersteps.*

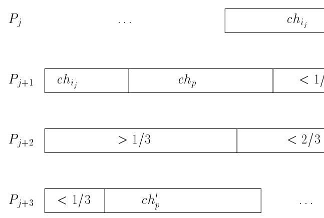

Figure 10. Chains larger than $\frac{2WL}{3}$.

Proof: Since the chains were allocated in decreasing order we will have three different groups of processors. Here we extend the notation for the groups of chains to groups of processors. In the first group (A) the chains of length WL are allocated. In the second group (B) the chains with more than $\lceil \frac{WL}{2} \rceil$ tasks are allocated. The left boundaries of the third group (C) is determined by the first allocation of a chain with at most $\lceil \frac{WL}{2} \rceil$ tasks to a processor.

It is clear that there is no need to do communications within group A. All the communications within group C can be grouped into a single superstep ending at $\lfloor \frac{WL}{2} \rfloor$.

The communications among the processors of group B, either they can either be grouped sequentially two by two (lemma 2), or one of the communications can be done at the instant $\lfloor \frac{WL}{2} \rfloor$. If there are x communications in group B it is possible to group them within $\lceil \frac{x}{2} \rceil$ supersteps, or $\lceil \frac{x}{2} \rceil + 1$ supersteps with an end of superstep at instant $\lfloor \frac{WL}{2} \rfloor$.

So there are at most $\lceil \frac{m-1}{2} \rceil + 1$ communications, therefore there are at most $\lceil \frac{m+1}{2} \rceil$ different BSP supersteps.❏

5 Experimental results

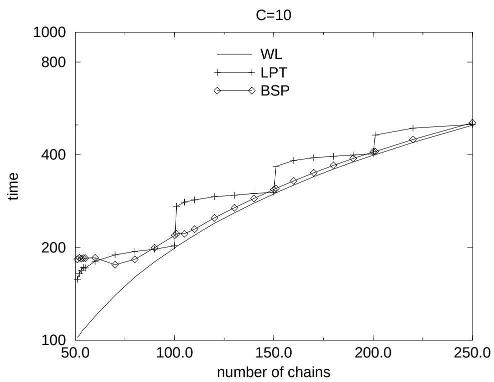

Figure 11. BSP vs LPT

Figure 11 presents the makespan achieved by our BSP algorithm compared to the well known Largest Processing Time first (LPT) algorithm (which has no communication). The simulations are run with 50 processors, and sets of chains of average length 100, standard deviation 10, with $C = 10$ (about 10 percents of a chain length).

We observe that even for large synchronization time, the proposed algorithm has a good performance. This is due to the small number of supersteps compared to the worst case, even if the chains have about the same length.

6 Concluding remarks

In this paper, we studied the influence of the computational model BSP on a simple scheduling problem, namely, SIC. We focused on the synchronization criterion. We proved that finding an optimal solution with any supersteps number is NP-hard, but near-optimal heuristics can be found. We showed that there exist some instances which may require $\lceil \frac{(m-1)}{2} \rceil$ supersteps, and proposed a general algorithm for this number with a performance guarantee.

Some simulation experiments show that in practice this algorithm needs only a few supersteps. We are currently working on an algorithm to solve SIC problem with a fixed number of supersteps with the idea to determine the best number which minimizes the total makespan. We are also interested by designing efficient algorithms in respect to the latency factor, which is the second important attribute of BSP model.

References

[1] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W.H. Freeman and Company, San Francisco, 1979.

[2] A. Goldman, J. Peters, and D. Trystram. Exchange of messages of different sizes. In *Irregular 98*, pages 194–205, August 1998. LNCS 1457.

[3] J. M. D. Hill, B. McColl, D. C. Stefanescu, M. W. Goudreau, K. Lang, S. B. Rao, T. Suel, T. Tsantilas, and R. Bisseling. Bsplib: The bsp programming library. Technical report PRG-TR-29-9, Oxford University Computing Laboratory, May 1997. url: http://www.BSP-Worldwide.org.

[4] J. M. D. Hill, W. McColl, and D. Skillircon. Questions and answers about bsp. Technical report PRG-TR-15-96, Oxford University Computing Laboratory, November 1996. to appear on Journal of Scientific Programming.

[5] I. Kort and D. Trystram. Some results on scheduling flat trees of under LogP. INFOR, to appear, 1998.

[6] V. Rayward-Smith. UET scheduling with unit interprocessor communication delays. *Discrete Applied Mathematics*, 1(18):55–71, January 1987.

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

[8] J. Verriet. Scheduling tree-structured programs in the LogP model. Technical report UU-CS-1997-18, Department of Computer Science, Utrecht University, Utrecht, the Netherlands, June 1997.