Scheduling Associative Reductions with Homogeneous Costs when Overlapping Communications and Computations

Louis-Claude Canon, Gabriel Antoniu

Scheduling Associative Reductions with Homogeneous Costs when Overlapping Communications and Computations

Louis-Claude Canon, Gabriel Antoniu

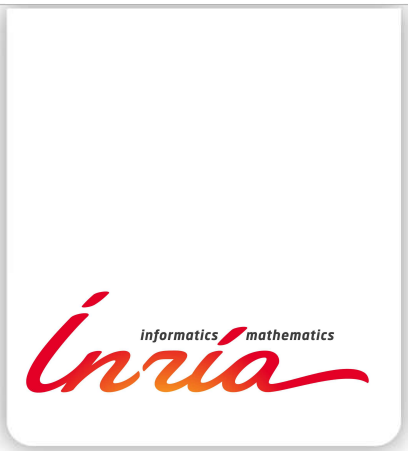
RESEARCH REPORT

N° 7898

March 2012

Project-Team KerData

ISSN 0249-6399 ISRN INRIA/RR--7898--FR+ENG



Scheduling Associative Reductions with Homogeneous Costs when Overlapping Communications and Computations

Louis-Claude Canon, Gabriel Antoniu

Project-Team KerData

Research Report n° 7898 — March 2012 — 17 pages

Abstract: Reduction is a core operation in parallel computing. Optimizing its cost has a high potential impact on the application execution time, particularly in MPI and MapReduce computations. In this paper, we propose an optimal algorithm for scheduling associative reductions. We focus on the case where communications and computations can be overlapped to fully exploit resources. Our algorithm greedily builds a spanning tree by starting from the sink and by adding a parent at each iteration. Bounds on the completion time of optimal schedules are then characterized. To show the algorithm extensibility, we adapt it to model variations in which either communication or computation resources are limited. Moreover, we study two specific spanning trees: while the binomial tree is optimal when there is either no transfer or no computation, the Fibonacci tree is optimal when the transfer cost is equal to the computation cost. Finally, approximation ratios of strategies that are derived from those trees are drawn.

Key-words: reduction; spanning tree; scheduling.

RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

Ordonnancement de réductions associatives avec des coûts homogènes lorsque les communications et les calculs se recouvrent

Résumé : L'opération de réduction est centrale au calcul parallèle. Optimiser son coût peut avoir un fort impact sur le temps d'exécution d'une application, en particulier dans le cas de MPI ou de MapReduce. Dans ce rapport, nous proposons une solution optimale pour ordonnancer des réductions associatives. Nous considérons que les communications et les calculs peuvent se recouvrir afin d'exploiter pleinement les ressources. Notre algorithme construit gloutonnement un arbre couvrant en commençant par le puits et en rajoutant un parent à chaque itération. Des bornes sur les temps d'exécution d'ordonnancements optimaux sont ensuite caractérisées. Pour montrer l'extensibilité de l'algorithme, nous l'adaptons à des variations du modèles dans lesquelles les communications ou les calculs sont limités. D'autre part, nous étudions deux arbres couvrants spécifiques: tandis que l'arbre binomial est optimal lorsqu'il n'y a soit aucun calcul, soit aucune communication, l'arbre de Fibonacci est optimal lorsque les temps de transfert et les temps de calcul sont égaux. Finalement, les facteurs d'approximation des stratégies dérivées de ces arbres sont déterminés.

Mots-clés : réduction; arbre couvrant; ordonnancement.

1 Introduction

Reduction is a very useful operation which frequently occurs in parallel computations: it consists in combining several elements to produce a single result, either as an intermediate or a final computational step. For instance, in the context of the MPI (Message Passing Interface) programming model, the `MPI_Reduce` collective function combines element-wise several arrays of the same size into a single one with the same size. More recently, the reduction problem has been brought forward again with the emergence of the MapReduce programmig model: a first (Map) step filters some initial data sets into intermediate data that are then aggregated in a second (Reduce) step.

In a distributed reduction operation, elements are typically retrieved through communications before being processed. When the reduction is associative, it consists in performing binary operations. In this case, its execution is determined by a spanning tree that schedules data transfers and computations. This allows subsets of elements to be reduced in parallel, which decreases the overall completion time of the reduction. In this paper we focus on this case, where the reduction operation can be expressed as an associative function. As this situation often occurs for two major parallel programming paradigms (MPI and MapReduce), optimizing associative reductions has a high potential impact on the overall application performance.

Existing work on this topic has mainly been focused on scheduling communications without accounting for computation costs. In this work, we assume that each reduction represents a large amount of computation. It is therefore natural to allow communications to overlap with computations. Most existing algorithms do not prefetch data to pipeline computations and are thus unable to fully exploit the resources. As the communication cost may differ from the computation cost, the degree of overlapping may vary, which makes this problem difficult.

We propose an algorithm that builds optimal spanning trees. This algorithm considers that time is reversed and starts thus by scheduling the sink that contains the final result. Other transfers are then scheduled greedily. Moreover, this algorithm can be adapted when limiting the number of concurrent transfers or the number of machines processing data. This shows the extensibility of the greedy principle that consists in building spanning trees starting from the sink. Two specific tree structures are also investigated. The binomial tree is a well-known spanning tree that is optimal when there is either no transfer or no computation. We introduce a similar tree, the Fibonacci tree, which is optimal when the transfer cost is equal to the computation cost. The binomial tree minimizes the number of steps, while the Fibonacci tree maximizes the pipelining of computations by prefetching data. For each tree, a corresponding strategy is derived from the main algorithm. Finally, the approximation ratios of those two strategies are analytically and empirically studied.

The paper is organized as follows. Section 2 discusses the related work. The model is then detailed in Section 3. The proposed algorithms use preliminary results that are available in Section 4. Section 5 presents the main algorithm, its optimality proof and bounds on the completion time of optimal schedules. Section 6 covers a similar analysis for two extensions of this algorithm. Finally, Section 7 describes the specific spanning trees.

2 Related Work

The literature has first been focused on the (global) combine problem [4, 5, 20], which is a variation of the reduction problem. Algorithmic contributions have then been proposed to improve MPI implementations and existing methods have been empirically studied in this context [2, 17, 19]. Recent works concerning MapReduce either exhibit the reduction problem or highlight the relations with MPI collective functions. We describe below the most significant contributions.

Bar-Noy et al. [3] propose a solution to the global combine problem, which is similar to

`MPI_Allreduce` when the array size is one: in addition to the reduction, all machines must be aware of the final result. They consider the postal model with a constraint on the number of concurrent transfers to the same node (multi-port model). However, computation costs are not considered.

Rabenseifner [15] introduces the butterfly algorithm for the same problem with arbitrary array sizes. Several vectors must be combined into a single one by applying a reduction operation element-wise. The algorithm solves the reduction problem in its first phase. Then, it broadcasts the result to all machines. The main principle lies in halving arrays successively. At each step, machines exchange data pairwise (each machine first contacts its closest neighbor, then its second closest neighbor, etc). The first half of the array is sent from one machine to the other while the second half is transfered in the opposite direction. Each machine performs reductions on the half for which they both possess data. Since this half only is considered for subsequent operations, transfer sizes are divided by two at each step. When all pairwise interactions have been done, the final result is scattered across all machines. To gather it, a similar algorithm is used. Another solution has also been proposed when the number of machines is not a power of two [16]. Those approaches are specifically adapted for element-wise reduction of arrays. Van de Geijn [6] also proposes a method with a similar cost.

Sanders, Speck and Träff [18] exploit in and out bandwidths. Although the reduction does not requires to be applied on arrays, the operation is split in at least two parts. This improves the approach based on a binary spanning tree by a factor of two.

Legrand, Marchal and Robert [11] study steady-state situations where a series of reductions are performed. As in our work, the reduction operation is assumed to be indivisible, transfers and computations can overlap and the full-duplex one-port model is considered. Costs are however heterogeneous. The solution is based on a linear program and is asymptotically optimal.

Liu, Kuo and Wand [12] solve optimally the problem with heterogeneous costs, but with non-overlapping transfers and computations.

In the MPI context, Kielmann et al. [9] design algorithms for collective communications, including `MPI_Reduce`, in hierarchical platforms. They propose three solutions: flat tree for short messages, binomial tree for long messages and a specific procedure for associative reductions in which data are first reduced locally on each cluster before the results are sent to the root process.

Pjesivac-Grbovic et al. [13] conduct an empirical and analytical comparison of existing algorithms for several collective communications. The analytical costs of those algorithms are first determined using different classical point-to-point communication models, such as Hockney, LogP/LogGP and PLogP. The compared `MPI_Reduce` algorithms are: flat tree, pipeline, binomial tree, binary tree and k-ary tree.

Finally, this problem has also been addressed for MapReduce applications. Agarwal et al. [1] present an implementation of AllReduce on top of Hadoop based on spanning trees. Moreover, some MapReduce infrastructures, such as MapReduce-MPI [14], are based on MPI implementations and benefits from the improvements done on `MPI_Reduce`. Hoefler et al. [7] further discuss how anticipated MPI-2.2 and MPI-3 features can optimize the Reduction phase.

3 Model

The model is divided into two parts. First, we characterize the operations that are performed and their costs. Then, we specify how those operations are planned and what constitutes the output of the problem.

3.1 Execution Model

The objective consists in processing efficiently n elements, each being available simultaneously on a separate machine. Combining those elements into a single result relies on an associative reduction operation

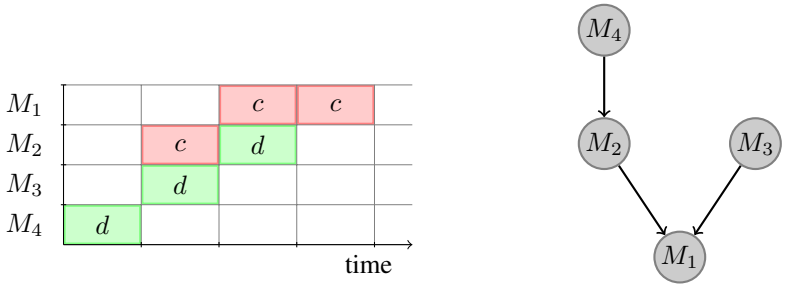


Figure 1: Schedule and corresponding spanning tree for reducing four elements ($n = 4$) when the transfer cost is equal to the computation cost ($d = c$). The transfer times are: $t_4 = 0$, $t_3 = d$ and $t_2 = 2d$.

that can be performed on any machine. Applying this operation on two elements costs c and produces a single element of the same nature.

Machines must then transfer their data to perform reduction. Let d be the time to transfer an element between any pair of machines. As we assume that transfers overlap with computations, a machine can fetch new data for future operations while reducing two local elements. However, each machine is involved in no more than one transfer at any given time (one-port model). Then, the time required to process k elements on a single machine is $d + (k-1)\max(d, c) + c$ (d for retrieving the first element with which the local element is reduced, followed by k reductions). If $d < c$, this simplifies as $d + kc$, otherwise, it is $kd + c$.

We finish with two general remarks. When transfers and computations do not overlap, the cost to sequentially reduce k elements is $k(d + c)$. This is a sub-case of the previous model where transfer costs are incorporated into computation costs. Note also that the operation is not assumed to be commutative. Although our proposed solutions rely on this property, a discussion on how to adapt them is provided in Section 4.1.

3.2 Scheduling Model

Each machine reduces every received elements with its own pairwise. Then, it sends the resulting data to another machine. This succession of operations is characterized by a reverse tree, called a *spanning tree*, where each vertex represents a machine and each edge represents a transfer between two machines.

The algorithms proposed in this paper build this tree by determining the child c_i of each machine i. Moreover, the transfer of the data computed by i starts at time t_i (if machine c_i is already transferring data, machine i waits before starting its own transfer). The objective is to minimize the schedule *length*, i.e., the time to reduce n elements. Finally, the sink (*root machine* using the MPI terminology), which contains the final result, is arbitrary. Figure 1 illustrates a spanning tree and its corresponding schedule with four elements.

3.3 Discussion

While each machine starts with an initial element with `MPI_Reduce`, it is not necessarily the case in MapReduce applications. This assumption holds when each mapper becomes a reducer because reducers have then access to the data resulting from the mappers. On the contrary, when reductions are performed on distinct machines, initial elements need to be retrieved locally on reducers. Once this step is finalized, the conditions defined by our model are met.

We also assume that all elements are available simultaneously. This is valid for the `MPI_Reduce` collective because it is blocking. For MapReduce applications, mappers can be executed by waves. In

this case, we consider that each time a wave is finished, the reduction occurs and terminates before the next wave.

Transfer and computation costs are furthermore supposed to be constant. While this is application-specific for the computation costs, transfer costs on non-dedicated platforms such as clouds may be impacted by high variability even with constant message lengths. The strength of this hypothesis depends thus on the application and on the platform.

As schedules are static, they are not robust to dynamic changes and faults. This, however, requires no synchronisation between machines since the reduction tree can be built identically on each machine (only the identifier of each machine needs to be globally known). Moreover, it allows to highlight the structure of the problem such as the two specific reduction trees that are characterized in Section 7.

4 Preliminary Results

Before proposing algorithms, we study the problem structure. First, we outline the general structure of any optimal greedy algorithm. Then, we determine the constraints on the transfer times t_i for any feasible schedule and we remark that the problem can be simplified when transfers are independent.

4.1 Optimal Greedy Construction

The scheduling algorithms proposed in this paper rely on a greedy strategy. Their optimality proofs is based on Lemma 1, which states that any schedule can be built greedily by iterating over the machines and by selecting the child of each machine among the set of visited machines.

We generalize the structure of this greedy approach as follow. Let $O \in \mathcal{O}$ denotes a specific order in which machine are visited. $G \in \mathcal{G}$ is a greedy strategy, i.e., a function whose input is the set of visited machines and whose output is the child of the current machine and its transfer date. The general algorithmic procedure can be modeled as a function A whose arguments are an order and a greedy strategy and whose image is a schedule. Lastly, two schedules $(S, S') \in \mathcal{S}^2$ are considered to be structurally equivalent, i.e., $S \equiv S'$, if and only if their respective spanning trees are isomorphic (similar machines through the isomorphism must additionally have the same transfer date).

Lemma 1. *For any schedule, there exists a greedy strategy that builds a structurally equivalent schedule using any arbitrary order (i.e.,* $\forall (S, O) \in \mathcal{S} \times \mathcal{O}, \exists G \in \mathcal{G}, S \equiv A(G, O)$*).*

Proof. Any spanning tree can be built by adding each machine in a reverse topological order by specifying the child of each inserted machine among the set of visited machines. Thus, any schedule can be built with a specific greedy strategy and a specific order (i.e., $\forall S \in \mathcal{S}, \exists (G, O) \in \mathcal{G} \times \mathcal{O}, S = A(G, O)$).

As machines are undistinguishable there is no distinction between machines, the vertices in any resulting spanning tree can be renumbered in any arbitrary order. Thus, any schedule built using a given order is structurally equivalent to all the schedules obtained with the same greedy strategy, but considering all the possible orders (i.e., $\forall (G, O, O') \in \mathcal{G} \times \mathcal{O}^2, A(G, O) \equiv A(G, O')$).

Therefore, for any schedule, there exists a greedy strategy allowing the construction of a structurally equivalent schedule using any arbitrary order. □

Section 3.1 states that the reduction is not assumed to be commutative. As presented in the previous proof, the machine order has no impact on the schedule structure. The machines can thus be permuted in any optimal schedule such that the non-commutative property of the reduction operation is respected.

4.2 Default Transfer Times

We first characterize a lower bound on the transfer times. Let m_i be the number of predecessors of machine i (i.e., $m_i = |\{j : c_j = i\}|$). Moreover, t_i^j is the jth largest transfer time among all the predecessors of machine i.

Proposition 2. *The transfer of the data computed by any machine i starts after that the reductions finish on i*

$$t_i \geq \max_{j\in[1,m_i]} \left(t_i^j + d + (m_i - j)\max(d,c) + c\right)$$

Proof. We prove by induction on k that the reduction of the first k elements on machine i finishes at time $f_i^k = \max_{j\in[1,k]} \left(t_i^j + d + (k-j)\max(d,c) + c\right)$.

Induction basis: the case for $k = 1$ is trivial since the reduction requires a single transfer and a single reduction and thus finishes at time $f_i^1 = t_i^1 + d + c$.

Induction step: there is two cases. If the transfer of the $(k+1)$th element starts before $f_i^k - \min(d,c)$, then the reduction can proceed without interruption. The computation finishes at time $f_i^{k+1} = f_i^k - \min(d,c) + d + c = f_i^k + \max(d,c)$. Using the induction hypothesis, $f_i^{k+1} = \max_{j\in[1,k]} \left(t_i^j + d + (k+1-j)\max(d,c) + c\right)$. As $t_i^{k+1} \leq f_i^k - \min(d,c)$, $f_i^{k+1} = \max_{j\in[1,k+1]} \left(t_i^j + d + (k+1-j)\max(d,c) + c\right)$ and the induction hypothesis holds for $k+1$.

Otherwise, the data transfer of the $(k+1)$th element starts after $f_i^k - \min(d,c)$ and the reduction finishing time depends only on t_i^{k+1}, i.e., $f_i^{k+1} = t_i^{k+1} + d + c$. As $t_i^{k+1} \geq f_i^k - \min(d,c)$, $f_i^{k+1} \geq f_i^k - \min(d,c) + d + c$. Using the induction hypothesis, $f_i^{k+1} \geq \max_{j\in[1,k]} \left(t_i^j + d + (k+1-j)\max(d,c) + c\right)$ and the induction hypothesis holds for $k+1$.

The proof is concluded by remarking that the transfer of machine i does not occur before its data is ready, i.e., $t_i \geq f_i^{m_i}$. □

Transfer times can optimally be set to their smallest possible values when there is no constraint on them. In the rest of this paper (with the exception of Section 6.1), presented algorithms rely on this implicit rule.

Proposition 3. *The transfer time of any machine i can be set to*

$$t_i = \max_{j\in[1,m_i]} \left(t_i^j + d + (m_i - j)\max(d,c) + c\right)$$

without increasing the schedule length.

Proof. Proposition 2 presents a lower bound for each transfer time. Its proof shows directly that this bound is tight, i.e., it corresponds exactly to the time at which the data is available. Therefore, the bound is the minimum feasible value for t_i.

As t_i is an optimistic transfer time (the transfer actually starts as soon as the currently queued transfers to c_i finish), postponing this time can lead to the inversion of two transfers targeting the same machine. Such inversions, however, do not reduce the schedule length. There is thus not better choice for t_i. □

Setting all transfer times to their smallest values requires to traverse the spanning tree in a breadth-first order and can be done in linear time.

Algorithm 1 Greedy scheduling algorithm

```
1:  inputs
2:    n {number of elements to reduce}
3:  do
4:    s_1 ← 0
5:    for i ← 2 to n do
6:      M ← argmin_{j∈[1,i−1]} s_j + c + d
7:      s_i ← s_M + c + d
8:      s_M ← s_M + max(d, c)
9:      c_i ← M
10:   end for
11:   return {c_i}_{1<i≤n}
```

5 Greedy Algorithm

The solution for the general model is given by Algorithm 1 and is explained below. The transfer times are set to the values specified by Proposition 3 because there is no constraint on the transfers. Its optimality is then proved and bounds on the length of optimal schedules are analyzed.

This algorithm builds a schedule by considering an *reversed* reduction operation. This operation first computes an element which is then transfered to another machine. This resembles a broadcast operation with intermediate computations and distinct elements. When time is reversed, s_i is the soonest time at which an element can be computed on machine i and then be transfered (machine i is thus available for a transfer at time $s_i + c$).

More specifically, the first part of the algorithm relies on the greedy principle outlined in Section 4.1: machines are successively inserted into a tree in an arbitrary order and the greedy strategy performed on Line 6 selects the machine M that would provide an element to machine i the soonest. Machine i is ready for a new computation as soon as the previous transfer completes (Line 7). The algorithm updates the end date of machine M on Line 8: machine M must have completed its computation and its transfers must be finished when the next computation occurs. The spanning tree is finally updated on Line 9. Child c_1 is left undefined as machine 1 stores the final reduced data.

Algorithm 1 requires $\Theta(n \log(n))$ steps (n iterations with a logarithmic search on Line 6).

Theorem 4. *Algorithm 1 is optimal.*

Proof. Lemma 1 states that all structurally distinct schedules can be built by iterating over the machines in any order and by inserting a parent at each iteration. Therefore, the optimal schedule can be obtained using any order as it is done by Algorithm 1.

By considering a given order, we show by contradiction that there is no other greedy strategy for selecting the child of any added node that leads to a shorter schedule length. Let $\{c'_i\}_{1<i\leq n}$ be the schedule obtained with another greedy strategy such that its length is lower than the length of the schedule $\{c_i\}_{1<i\leq n}$ built with Algorithm 1. Let k be the smallest index for which both strategies differ (i.e., $c'_k \neq c_k$). It is also assumed that $\{c'_i\}_{1<i\leq n}$ is such that there is no schedule with a better or equal length and such that the first k children are identical to $\{c_i\}_{1<i\leq k}$ (otherwise the divergence at index k is not significant and we consider this other schedule as being potentially better instead).

Let k' be the smallest index greater than k such that machine k' sends its data to c_k in schedule $\{c'_i\}_{1<i\leq n}$ ($c'_{k'} = c_k$). The indexes of machines k and k' can be permuted without increasing the length. If there is no such index, then machine k can send its data to c_k instead of c'_k. In both cases, it contradicts the assumption that there is no schedule with a better or equal length and such that the first k children are identical to $\{c_i\}_{1<i\leq k}$. □

The following two propositions characterize lower and upper bounds on the length of any optimal schedule.

Lemma 5. *The optimal length for reducing n elements is greater than or equal to $\lceil \log_2(n) \rceil \max(d, c)$.*

Proof. The proof is by induction on the number of elements. The induction hypothesis H_k is that the optimal length for reducing $n = 2^k + 1$ elements is greater than or equal to $\lceil \log_2(n) \rceil \max(d, c) = (k + 1) \max(d, c)$.

Induction basis: for $k = 0$, there is only one possible schedule, which is thus optimal. Its length is $d + c$, which is greater than or equal to $\max(d, c)$.

We show by contradiction that H_{k+1} cannot be false if H_k is true. Assume that the optimal length for reducing $2^{k+1} + 1$ is lower than $(k + 2) \max(d, c)$. From the corresponding schedule, we can build two schedules by ignoring the last reduction (the one that is related to the last data sent to the sink). Let S_1 be the schedule having the same sink as the initial one without the sub-tree whose sink sends the last data in the initial one (the schedule corresponding to this last sub-tree is denoted by S_2). The length of S_1 is lower than $(k + 2) \max(d, c) - \max(d, c)$ (the $d > c$ case requires to consider that each pipelined reduction waits for an element to be received) and the length of S_2 is lower than $(k + 2) \max(d, c) - (d + c)$. Both lengths are thus lower than $(k + 1) \max(d, c)$. As the number of elements reduced by S_1 and S_2 is $2^{k+2} + 1$, the optimal length for reducing $2^{k+1} + 1$ of them is lower than $(k + 1) \max(d, c)$, which contradicts H_k.

The proof is completed by remarking that the optimal length is monotonically non-decreasing when the number of elements to reduce increases. Let $n' = 2^{\lfloor \log_2(n-1) \rfloor} + 1$ be the largest value not greater than a given number of elements n and for which the previous induction provides a lower bound. Therefore, the optimal length for reducing n elements is greater than or equal to $\lceil \log_2(n') \rceil \max(d, c) = \lceil \log_2(2^{\lfloor \log_2(n-1) \rfloor} + 1) \rceil \max(d, c) = (\lfloor \log_2(n - 1) \rfloor + 1) \max(d, c) = \lceil \log_2(n) \rceil \max(d, c)$. □

Proposition 6. *The optimal length for reducing n elements is lower than or equal to $\lceil \log_2(n) \rceil (d + c)$.*

Proof. Consider a schedule that consists of several steps of length $d + c$. At each step, machines that have data compute and transfer new data to other machines (considering that time is reversed). Such a schedule takes $\lceil \log_2(n) \rceil$ steps and its length is greater than or equal to the optimal length. □

The more $\min(d, c)$ is close to zero, the tighter those bounds are (they are tight when $d = 0$ or $c = 0$).

Additional weaker lower bounds that depends on the structure of the spanning tree can be derived. They provide an intuition on the structure of optimal schedules. Let $\deg$ be the maximum in-degree in the tree (i.e., the maximum arity of any reduction) and depth be the depth of the tree. Then, the length of any schedule is greater than or equal to $d + (\deg -1) \max(d, c) + c$ and to $(\mathrm{depth} -1)(d + c)$. This suggests that optimal schedules have a maximum in-degree and a depth in $O(\log(n))$. This is actually the case for the trees presented in Section 7.

6 Extensions

This section shows that Algorithm 1 can be adjusted when constraints extend the model. Two examples are described: limited number of concurrent transfers; and, limited number of machines that perform computations.

6.1 Limited Concurrent Transfers

The following algorithm assumes that there is a limit K on the number of concurrent transfers (see Algorithm 2). This limits the contention in platforms where several machines are interconnected through

Algorithm 2 Greedy scheduling algorithm with a limited number of concurrent transfers

1: **inputs**
2: n {number of elements to reduce}
3: K {maximum number of concurrent transfers}
4: **do**
5: $s_1 \leftarrow 0$
6: **for** $i \leftarrow 2$ **to** n **do**
7: $M \leftarrow \text{argmin}_{j \in [1,i-1]} \, s_j + c + d$
8: $t_i \leftarrow \max(s_M + c, t_{i-K}) + d$
9: $s_i \leftarrow t_i$
10: $s_M \leftarrow \max(s_M + c, t_i - c)$
11: $c_i \leftarrow M$
12: **end for**
13: **for all** $i \leftarrow 2$ **to** n **do**
14: $t_i \leftarrow t_n - t_i + c$
15: **end for**
16: **return** $\{c_i, t_i\}_{1<i\leq n}$

a network equipment that has a limited aggregated bandwidth. This algorithm is explained below, its optimality is then proved and the length of each generated schedule is characterized.

This algorithm relies on the same principle as Algorithm 1, with which it shares the same complexity, $\Theta(n \log(n))$. Transfer times must however be defined. When time is reversed, t_i corresponds to the time at which the transfer from c_i to i is completed. The time taken to complete a data transfer is computed on Line 8. The maximum operation controls the contention by delaying the current transfer if the number of concurrent transfer reaches the limit K ($t_i = 0$ for $i \geq n$). The final schedule is obtained by reversing the transfer times from Line 13 to Line 15 (c_1 and t_1 are left undefined as machine 1 stores the final reduced data).

Theorem 7. *When no more than K concurrent transfers is allowed, Algorithm 2 is optimal.*

Proof. Proving this theorem follows the same structure as the proof of Theorem 4. Transfer times must, however, be considered. The arbitrary order in which machines are visited in Algorithm 2 is supported by Lemma 1.

We consider a given order and we show by contradiction that there is no other greedy strategy for selecting the child of any added node and for setting its transfer time that leads to a shorter schedule length. Let $\{c'_i, t'_i\}_{1<i\leq n}$ be the schedule obtained with another greedy strategy such that its length is lower than the length of the schedule $\{c_i, t_i\}_{1<i\leq n}$ built with Algorithm 2. Let k be the smallest index for which both strategies differ (i.e., $c'_k \neq c_k$ or $t'_k \neq t_k$). It is also assumed that $\{c'_i, t'_i\}_{1<i\leq n}$ is such that there is no schedule with a better or equal length and such that the first k children and transfer times are identical to $\{c_i, t_i\}_{1<i\leq k}$ (otherwise the divergence at index k is not significant and we consider this last schedule as $\{c'_i, t'_i\}_{1<i\leq n}$ instead). In the following, we consider that time is reversed (before Line 13). There is three cases.

The case where $t'_k < t_k$ can be eliminated because the transfer between machine k and machine c'_k cannot complete before t_k. This can be proved by remarking that transfer times are monotonically non-decreasing in any schedule built by Algorithm 2 (at each iteration, the minimum value s_M of the set $\{s_i\}_{1\leq i<k}$ is increased and t_k is greater than s_M). On Line 8, there is two initialization choices. If $t_k \leftarrow t_{k-K} + d$, then a value $t'_k < t_k$ would violate the contention limit. Otherwise, $t_k \leftarrow s_M + c + d$ and the schedule $\{c'_i, t'_i\}_{1<i\leq n}$ is also invalid by definition of s_M, which is the soonest time at which a computation can occur on machine k.

The second case is when $t'_k > t_k$ and $c'_k = c_k$. Let k' be the smallest index greater than k such that $t_k \leq t_{k'} < t_k + d$. If there is no such machine, the transfer can directly be advanced to t_k without increasing the length. Otherwise, two steps need to be performed before. First, the transfer times t_k and $t_{k'}$ are exchanged. Then, the parents of machines k are connected to machine k' and vice versa. This leads to a schedule with the same length as $\{c'_i, t'_i\}_{1<i\leq n}$ and such that the first k children and transfer times are identical to $\{c_i, t_i\}_{1<i\leq k}$, which contradicts the assumption that there is no such schedule.

When $t'_k \geq t_k$ and $c'_k \neq c_k$, the permutation presented in the proof of Theorem 4 can be performed, which leads to the same previous contradiction.

On the one hand, we have shown that it is not possible to built a better schedule while respecting the contention limit. On the other hand, having more than K concurrent transfers is impossible because transfer times are monotonically non-decreasing when time is reversed. Thus, generated schedules respect the contention limit. □

The length of any generated schedule depends on the limit K on the number of concurrent transfers. As each machine is assumed to be involved in at most one transfer at any time, there is no more than $\left\lfloor \frac{n}{2} \right\rfloor$ concurrent transfers. Thus, we consider that the limit K is lower than or equal to this hard limit.

Proposition 8. *The length of any schedule built by Algorithm 2 is lower than or equal to* $\left(\lfloor \log_2(K) + 1 \rfloor + \left\lceil \frac{n}{K} - 2 \right\rceil\right)(d + c)$ *with* $K \leq \left\lfloor \frac{n}{2} \right\rfloor$.

Proof. As in the proof of Proposition 6, we consider a sub-optimal schedule that consists of several steps of length $d + c$. At each step, some machines compute and transfer data to other machines (considering that time is reversed).

The behavior of this schedule has two modes. In the first mode, the number of concurrent transfers increases from 1 to the largest power of two that is not greater than K (i.e., $2^{\lfloor \log_2(K) \rfloor}$). This first mode takes $\lfloor \log_2(K) + 1 \rfloor$ steps and results in $2^{\lfloor \log_2(K)+1 \rfloor}$ machines having an element.

During the second mode, the same set of K machines compute and transfer new data to K other machines at each step. As there is $n - 2^{\lfloor \log_2(K)+1 \rfloor} < n - 2K$ remaining machines, this second mode takes less than $\left\lceil \frac{n}{K} - 2 \right\rceil$ steps, which concludes the proof. □

6.2 Limited Reducers

In MapReduce frameworks, there may be a predefined amount of *reducers*, i.e., machines that perform reductions. In this case, operations must be scheduled only on a subset of machines of size K. The other machines only transfer their data to the reducers, as the mappers do in the execution of MapReduce applications. Algorithm 3 is similar to Algorithm 2 and they both share several properties. However, the transfer times are set to their smallest values (Proposition 3) as with Algorithm 1.

In this algorithm, once a machine has been selected for a reduction on Line 7, it belongs to the subset of K reducers. In this case, this set comprises the first K visited machines. The rest of the algorithm is identical to Algorithm 1.

The cost of Algorithm 3 is again $\Theta(n \log(n))$.

Theorem 9. *When no more than K reducers are available, Algorithm 3 is optimal.*

Proof. Before proving the optimality of the algorithm, we first prove that the limit on the number of reducers is respected.

The child of any machine i is selected among the set of the first K machines (Line 7). Thus, the last $n - K$ machines have no parents. As only machines that receive data perform computations, no more than K reducers are used.

As in the proof of Theorem 4, we consider the smallest index k for which a strategy leading to a better schedule differs. Let $M = c_k$ be the machine selected by Algorithm 3 and $M' = c'_k$ the machine

Algorithm 3 Greedy scheduling algorithm with a limited number of reducers

```
1:  inputs
2:    n {number of elements to reduce}
3:    K {maximum number of computing machines}
4:  do
5:    s_1 ← 0
6:    for i ← 2 to n do
7:      M ← argmin_{j∈[1,min(i−1,K)]} s_j + c + d
8:      s_i ← s_M + c + d
9:      s_M ← s_M + max(d, c)
10:     c_i ← M
11:   end for
12:   return {c_i}_{1<i≤n}
```

in this hypothetical better strategy. There is three cases, the first two being straightforward. If $M' \leq K$, then machine M provides an element sooner or at the same time (when time is reversed) and should be selected instead. The same situation occurs when $M' > K$ and $s_{M'} \geq s_M$.

The final case is when $M' > K$ and $s_{M'} < s_M$. We show that the limit on the number of reducers is not respected because the first K machines are reducers. We first prove that when the machine with smallest $s_i + c + d$ does not belong to the first K machines, then each of the first K machines has a parent. By construction, the values that are assigned to s_i at each iteration are monotonically non-decreasing. Hence, when $s_{i'} < s_i$ with $i' > i$, then s_i has been incremented at Line 9 and machine i is a reducer (it performs a computation). Thus, the expression $\min(i-1, K)$ on Line 7 is equal to K only when the first K machines are all reducers. If $M' > K$ and $s_{M'} < s_M$, then there is $K+1$ reducers, which leads to a contradiction.

Therefore, the algorithm does not used more than K reducers and there is no other schedule with a lower length. □

Algorithm 3 also constitutes an optimal and simpler algorithm for the case covered in Section 6.1 when the transfer cost is greater than or equal to the computation cost.

Theorem 10. *When no more than K concurrent transfers is allowed and when $d \geq c$, Algorithm 3 is optimal.*

Proof. As any machine that is receiving data is a reducer, we prove that the number of concurrent transfers is no more than K because Algorithm 3 limits the number of reducers to K.

The proof is completed by following the same steps as the proof of Theorem 9. In the last case, which occurs when there is already K reducers, there is also K concurrent transfers. This is due to the fact that any reducer is continuously receiving data with the computations being completely overlapped when $d \geq c$. □

The length of the generated schedules is finally bounded by a corollary of Proposition 8, which can be proved using the same proof.

Corollary 11. *The length of any schedule built by Algorithm 3 is lower than or equal to $\left(\lfloor \log_2(K) + 1 \rfloor + \left\lceil \frac{n}{K} - 2 \right\rceil\right)(d+c)$ with $K \leq \left\lfloor \frac{n}{2} \right\rfloor$.*

7 Specific Spanning Trees

We introduce in this section two strategies that exhibit specific spanning trees: binomial and Fibonacci trees. Those trees are formally defined and the lengths of the corresponding schedules are characterized. Then, the situations in which they are optimal are identified. Finally, the two proposed strategies are described and their approximation ratios are studied analytically and empirically.

7.1 Binomial Tree

The binomial tree is a spanning tree that is already known to be optimal for broadcast operations [8].

Definition 12. *A* binomial tree *of order $k > 0$ is a binomial tree of order $k-1$ whose sink is the parent of the sink of another binomial tree of order $k-1$. A binomial tree of order* 0 *is a single node.*

Proposition 13. *The length of a schedule whose spanning tree is a binomial tree of order $k \geq 0$ is $k(d+c)$ and the number of reduced elements is 2^k.*

Proof. The proof is by induction on the order k of the binomial tree.

Induction basis: for $k = 0$, the cost to reduce a single element is zero.

Induction step: the length for order $k \geq 0$ is assumed to be $k(d + c)$. By definition, the last step with a binomial tree of order $k + 1$ consists in reducing two intermediate results of two binomial trees of order k. By induction hypothesis, both elements are available at time $k(d + c)$. The proof is completed by remarking that the time to transfer one element to the sink and to compute it takes $d + c$. □

The length of a binomial tree, characterized by the previous proposition, indicates that the reduction involves several steps during which data are reduced by half. Since the length of each step is $d + c$, data are first transfered before being processed. Thus, transfers do not overlap with computations in binomial trees. Moreover, since all transfers start at the same time at each step, machines send data as soon as possible. This eager strategy is, however, only optimal in the cases described by the following theorem.

Theorem 14. *For any $k \geq 0$ and when $\min(d, c) = 0$, no more than 2^k elements can be reduced in $k(c + d)$ time units and a binomial tree of order k is the unique solution.*

Proof. The proof is by induction on the order k of the binomial tree.

Induction basis: for $k = 0$, there is a single element. A binomial tree of order zero is thus the unique solution.

Induction step: the theorem is assumed to be true for a given order $k \geq 0$. We show by contradiction that it is also the case for $k+1$. Consider a schedule that reduces at least 2^{k+1} elements in $(k+1)(c+d)$ time units and that is not a binomial tree. As in the proof of Proposition 5, we can build two schedules from this one, both with lengths lower than or equal to $k(c + d)$ (because $d = 0$ or $c = 0$). By induction hypothesis, each of them is binomial and reduces at most 2^k elements, which leads to a contradiction. □

As a corollary, binomial trees are optimal when either transfers or computations have negligible costs and Algorithm 1 builds such trees when the number of elements is a power of two and when $\min(d, c) = 0$.

7.2 Fibonacci Tree

The Fibonacci tree is also a well-known tree structure [10, Section 6.2.1]. We propose, however, an alternate definition that mimics Definition 12. As a consequence, the following Fibonacci trees are not binary.

Definition 15. *A* Fibonacci tree *of order $k > 0$ is a Fibonacci tree of order $k-2$ whose sink is the parent of the sink of another Fibonacci tree of order $k-1$. Fibonacci trees of order -1 and 0 consist each of a single node.*

Proposition 16. *The length of a schedule whose spanning tree is a Fibonacci tree of order $k > 0$ is $d + (k-1)\max(d,c) + c$ and the number of reduced elements is F_{k+2}, the $(k+2)$th Fibonacci number. For order -1 and 0, no data is reduced.*

Proof. The proof is by induction on the order k of the Fibonacci tree.

Induction basis: for $k = 1$, the cost to reduce two elements is $d + c$. For $k = 2$, two elements are sent successively to the sink (the second transfer overlaps with the first computation). The cost is thus $d + \max(d,c) + c$.

Induction step: the proposition is assumed to be true for order k and $k+1$, with $k > 0$. By definition, the last step with a Fibonacci tree of order $k+2$ consists in reducing two intermediate results of two Fibonacci trees of orders k and $k+1$. By induction hypothesis, the first element is available at time $d + (k-1)\max(d,c) + c$ and the second at time $d + k\max(d,c) + c$. The first element can then be sent to the sink at time $d + k\max(d,c) + c - \min(d,c)$ in order to overlap the transfer with the penultimate computation of the sink. Transferring one element to the sink and computing it takes $d + c$, which concludes the proof. □

In contrast to binomial trees, the length of a Fibonacci tree implies that reductions are pipelined: any machine receiving data overlaps transfers and computations. Thus, machines receive data as soon as it is necessary to fill the pipeline.

Theorem 17. *For any $k > 0$ and when $d = c$, no more than F_{k+2} elements can be reduced in $d + (k-1)\max(d,c) + c$ time units and a Fibonacci tree of order k is the unique solution.*

Proof. The proof is by induction on the order k of the Fibonacci tree.

Induction basis: for $k = 1$, the only solution for reducing $F_3 = 2$ elements is a Fibonacci tree of order 1 with length $d + c$. Three elements can be reduced either with a chain or a Fibonacci tree of order 2. The length of the chain is $2(d+c)$ whereas the length of the Fibonacci tree is $d + \max(d,c) + c$. As there is no schedule for reducing four elements with a lower length, the theorem also holds for $k = 2$.

Induction step: the theorem is assumed to be true for order k and $k+1$, with $k > 0$. We show by contradiction that it is also the case for $k+2$. Consider a schedule that reduces at least F_{k+4} elements in $d + (k+1)\max(d,c) + c$ time units and that is not a Fibonacci tree. As in the proof of Proposition 5, we can build two schedules from this one, one with length at most $d + (k+1)\max(d,c) + c - (d+c) = d + (k-1)\max(d,c) + c$ and another with length at most $d + (k+1)\max(d,c) + c - \max(d,c) = d + k\max(d,c) + c$ (because $d = c$). By induction hypothesis, each of them is a Fibonacci tree and the first reduces at most F_{k+2} elements while the second reduces at most F_{k+3} elements, which leads to a contradiction. □

7.3 Approximation Ratios

A direct consequence of Theorems 14 and 17 is that Algorithm 1 builds binomial and Fibonacci trees for specific values of n, d and c. To determine the efficiency of those trees with arbitrary costs, we propose two simplifications of the greedy algorithm that are specific to each tree and we determine their approximation ratios.

The *binomial strategy* is an algorithm derived from Algorithm 1 where one of the cost (d or c) is systematically set to zero such that $\min(d,c) = 0$. It builds binomial trees when the number of elements is a power of two, independently of the actual costs.

Theorem 18. *The binomial strategy is a* $\left(1 + \frac{\min(d,c)}{\max(d,c)}\right)$*-approximation.*

Proof. As a corollary of Proposition 13, the binomial strategy builds schedules with lengths lower than or equal to $\lceil \log_2(n) \rceil (d + c)$.

The approximation ratio can be directly derived from Proposition 5. □

As a corollary, the binomial strategy is a 2-approximation in the worst-case, i.e., when $d = c$.

The *Fibonacci strategy* is an algorithm derived from Algorithm 1 where costs are supposed to be homogeneous ($d = c$). It builds Fibonacci trees when the number of elements is a Fibonacci number, independently of the actual costs.

Theorem 19. *The Fibonacci strategy is a 2-approximation.*

Proof. We first use Proposition 16 to characterize the length of any schedule built with the Fibonacci strategy. When the number of elements n is the $(k+2)$th Fibonacci number, the Fibonacci strategy builds a schedule with length $d + (k-1)\max(d,c) + c$. Let $F_k^{(-1)}$ denotes the inverse Fibonacci function that associates the number of elements to the order of a Fibonacci tree. We assume that this function is monotonically non-decreasing. Then, the previous length can be expressed as $d + (\lceil F_k^{(-1)}(n) \rceil - 3)\max(d,c) + c$, which is lower than or equals to $(\lceil F_k^{(-1)}(n) \rceil - 1)\max(d,c)$.

We then need to prove that this length is lower than or equal to twice the lower bound given by Proposition 5, i.e., to $2\lceil \log_2(n) \rceil \max(d,c)$. We show that it is the case when $F_k^{(-1)}(n) - 1 \leq 2\log_2(n)$:

$$
\begin{aligned}
F_k^{(-1)}(n) - 1 &\leq 2\log_2(n) \\
\lceil F_k^{(-1)}(n) \rceil - 1 &\leq \lceil 2\log_2(n) \rceil \leq 2\lceil \log_2(n) \rceil \\
(\lceil F_k^{(-1)}(n) \rceil - 1)\max(d,c) &\leq 2\lceil \log_2(n) \rceil \max(d,c)
\end{aligned}
$$

To determine the number of elements n for which the inequality $F_k^{(-1)}(n) - 1 \leq 2\log_2(n)$ holds, we establish a bound on the inverse Fibonacci function. By definition, $F_k \geq \frac{\phi^k - (1-\phi)^2}{\sqrt{5}}$. It follows that $F_k^{(-1)}(n) \leq \log_\phi(\sqrt{5}n + (1-\phi)^2)$. The inequality $\log_\phi(\sqrt{5}n + (1-\phi)^2) - 1 \leq 2\log_2(n)$ holds for any $n > 2$. The Fibonacci strategy is furthermore trivially optimal for a single element and for two elements. □

We further study the approximation ratios for specific values of n empirically. For each number of elements, the length obtained with the binomial (resp., Fibonacci) strategy is compared to the length obtained with Algorithm 1 when $d = c$ (resp., $\min(d,c) = 0$). This is conjectured to provide the worst-case ratios for both strategies. Figure 2 depicts those ratios for $2 \leq n \leq 10000$.

8 Conclusion

This paper covers the reduction problem when communications overlap with computations. This is a general problem related to at least two major paradigms in distributed systems, MPI and MapReduce. An optimal greedy algorithm is introduced and we show how to extend it to two model variations. Additionally, the structure of the problem is investigated by characterizing two specific spanning trees, binomial ans Fibonacci trees. A strategy was derived from each of those trees by fixing cost parameters in the main algorithm. This leads to simple solutions, which are within a factor of two from the optimum.

As perspectives, more complex settings in the model could be explored such as considering arrival dates for data, or heterogeneous costs. A second direction consists in studying dynamic scheduling strategies that would be robust to variability in the system.

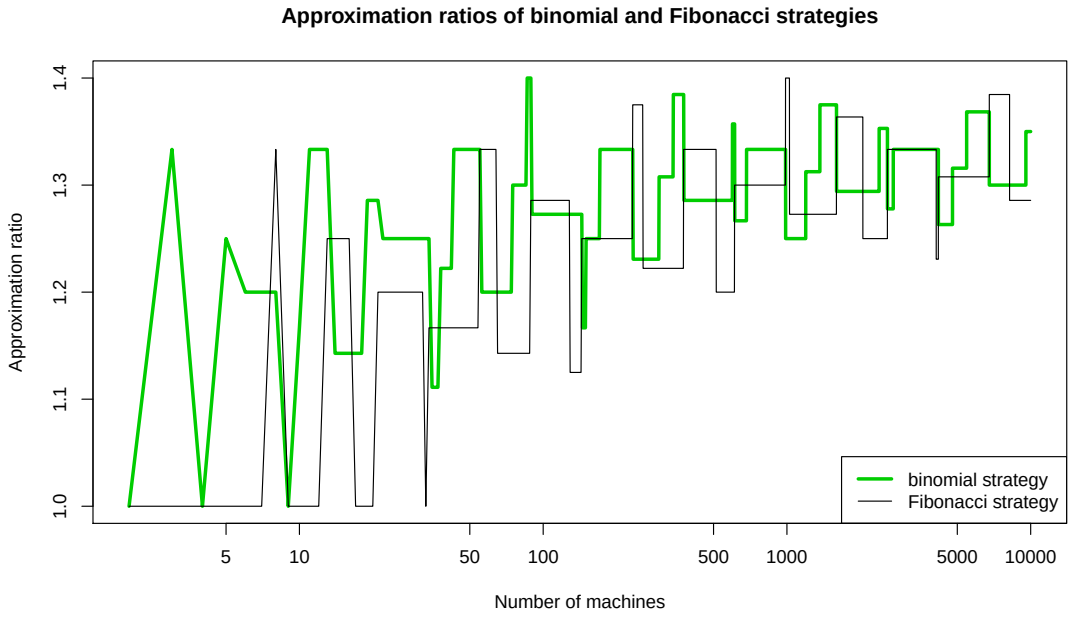


Figure 2: Ratios between binomial (resp., Fibonacci) schedule lengths and the lengths of the optimal schedules when $d = c$ (resp., $\min(d, c) = 0$).

References

[1] Alekh Agarwal, Olivier Chapelle, Miroslav Dudík, and John Langford. A Reliable Effective Terascale Linear Learning System. *CoRR*, abs/1110.4198, 2011.

[2] Qasim Ali, Vijay S. Pai, and Samuel P. Midkiff. Advanced collective communication in Aspen. In *Proceedings of the 22nd annual international conference on Supercomputing*, pages 83–93, New York, NY, USA, 2008. ACM.

[3] Amotz Bar-Noy, Jehoshua Bruck, Ching-Tien Ho, Shlomo Kipnis, and Baruch Schieber. Computing global combine operations in the multiport postal model. *IEEE Transactions on Parallel and Distributed Systems*, 6(8):896–900, August 1995.

[4] Amotz Bar-Noy, Shlomo Kipnis, and Baruch Schieber. An optimal algorithm for computing census functions in message-passing systems. *Parallel Processing Letters*, 3(1):19–23, 1993.

[5] Jehoshua Bruck and Ching-Tien Ho. Efficient global combine operations in multi-port message-passing systems. *Parallel Processing Letters*, 3(4):335–346, 1993.

[6] Ernie W. Chan, Marcel F. Heimlich, Avi Purkayastha, and Robert A. van de Geijn. Collective communication: theory, practice, and experience. *Concurrency and Computation: Practice and Experience*, 19(13):1749–1783, 2007.

[7] Torsten Hoefler, Andrew Lumsdaine, and Jack Dongarra. Towards Efficient MapReduce Using MPI. In Matti Ropo, Jan Westerholm, and Jack Dongarra, editors, *Recent Advances in Parallel Virtual*

Machine and Message Passing Interface, volume 5759 of *Lecture Notes in Computer Science*, pages 240–249. Springer Berlin / Heidelberg, 2009.

[8] S. Lennart Johnsson and Ching-Tien Ho. Optimum broadcasting and personalized communication in hypercubes. *IEEE Transactions on Computers*, 38(9):1249–1268, September 1989.

[9] Thilo Kielmann, Rutger F. H. Hofman, Henri E. Bal, Aske Plaat, and Raoul A. F. Bhoedjang. MPI's reduction operations in clustered wide area systems, 1999.

[10] Donald E. Knuth. *The art of computer programming, volume 3: (2nd ed.) sorting and searching*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998.

[11] Arnaud Legrand, Loris Marchal, and Yves Robert. Optimizing the steady-state throughput of scatter and reduce operations on heterogeneous platforms. *Journal of Parallel Distributed Computing*, 65(12):1497–1514, 2005.

[12] Pangfeng Liu, May-Chen Kuo, and Da-Wei Wang. An Approximation Algorithm and Dynamic Programming for Reduction in Heterogeneous Environments. *Algorithmica*, 53(3):425–453, February 2009.

[13] J. Pjesivac-Grbovic, T. Angskun, G. Bosilca, G.E. Fagg, E. Gabriel, and J.J. Dongarra. Performance analysis of MPI collective operations. In *19th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, April 2005.

[14] S. Plimpton and K. Devine. MapReduce-MPI Library. `http://www.sandia.gov/~sjplimp/mapreduce.html`.

[15] Rolf Rabenseifner. Optimization of Collective Reduction Operations. In Marian Bubak, Geert van Albada, Peter Sloot, and Jack Dongarra, editors, *Computational Science - ICCS 2004*, volume 3036 of *Lecture Notes in Computer Science*, pages 1–9. Springer Berlin / Heidelberg, 2004.

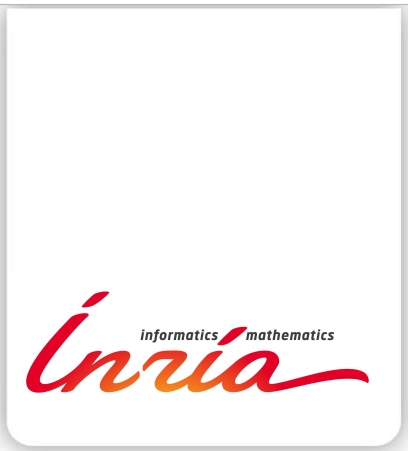
[16] Rolf Rabenseifner and Jesper Larsson Träff. More Efficient Reduction Algorithms for Non-Power-of-Two Number of Processors in Message-Passing Parallel Systems. In *Recent advances in parallel virtual machine and message passing interface*, Lecture Notes in Computer Science. Springer Berlin, 2004.

[17] H. Ritzdorf and Jesper Larsson Träff. Collective operations in NEC's high-performance MPI libraries. In *20th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, April 2006.

[18] Peter Sanders, Jochen Speck, and Jesper Larsson Träff. Two-tree algorithms for full bandwidth broadcast, reduction and scan. *Parallel Computing*, 35(12):581–594, 2009.

[19] Rajeev Thakur and Rolf Rabenseifner. Optimization of Collective communication operations in MPICH. *International Journal of High Performance Computing Applications*, 19:49–66, 2005.

[20] Robert A. van de Geijn. On global combine operations. *Journal of Parallel and Distributed Computing*, 22(2):324–328, 1994.



RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399