PRO: a Model for Parallel Resource-Optimal Computation [*]

Assefaw Hadish Gebremedhin[†] Isabelle Guérin Lassous[‡]
Jens Gustedt[§] Jan Arne Telle

Abstract

We present a new parallel computation model that enables the design of resource-optimal scalable parallel algorithms and simplifies their analysis. The model rests on the novel idea of incorporating relative optimality as an integral part and measuring the quality of a parallel algorithm in terms of granularity.

Key words: Parallel computers, Parallel models, Parallel algorithms, Complexity analysis

[*]Research supported by IS-AUR 02-34 of The Aurora Programme, a France-Norway Collaboration Research Project of The Research Council of Norway, The French Ministry of Foreign Affairs and The Ministry of Education, Research and Technology.

[†]Department of Informatics, University of Bergen, N-5020, Norway. {assefaw, telle}@ii.uib.no

[‡]LIP & INRIA Rhone-Alpes, France. Isabelle.Guerin-Lassous@inria.fr

[§]LORIA & INRIA Lorraine, France. gustedt@loria.fr

1 Introduction

One of the challenges in parallel processing is the development of a general purpose and effective model of parallel computation. Unlike the realm of sequential computation, where the Random Access Machine (RAM) has successfully served as a standard computational model, no such single unifying model exists in the field of parallel computation. From an algorithmic point of view, the performance of a sequential algorithm is adequately evaluated using its execution time making the RAM powerful enough for analysis and design. On the other hand, the performance evaluation of a parallel algorithm involves several metrics, the most important of which are *speedup*, *optimality* (or *efficiency*), and *scalability.* Speedup and optimality are *relative* in nature as they are expressed with respect to some sequential algorithm. The notion of relativity is also relevant from a practical point of view. A parallel algorithm is often not designed from scratch, but rather starting from a sequential algorithm.

We believe that a parallel computation model should incorporate the most important performance evaluation metrics of parallel algorithms as the RAM does for sequential algorithms. In light of this, the objective of the current work is to develop a model that simplifies the design and analysis of *resource-optimal scalable* parallel algorithms.

In an interesting survey paper [21], Maggs *et al.* suggest that an ideal parallel computation model be designed within "the philosophy of *simplicity* and *descriptivity* balanced with *prescriptivity*". The Parallel Resource-Optimal (PRO) computation model proposed here is developed within this spirit. The key features of the PRO model that distinguish it from existing parallel computation models are *relativity*, *resource-optimality*, and a new quality measure referred to as *granularity.*

Relativity pertains to the fact that the design and analysis of a parallel algorithm in PRO is done relative to the time and space complexity of a *specific* sequential algorithm. Consequently, the parameters involved in the analysis of a PRO-algorithm are the number of processors p, the input size n, and the time and space complexity of the reference sequential algorithm A_{seq}.

A PRO-algorithm is required to be both time- and space-optimal (hence resource-optimal). A parallel algorithm is said to be time- (or work-) optimal if the overall computation and communication cost involved in the algorithm is proportional to the time complexity of the sequential algorithm used as a reference. Similarly, it is said to be space-optimal if the overall memory space used by the algorithm is of the same order as the memory usage of the underlying sequential version. As a consequence of its time-optimality, a PRO-algorithm always yields *linear speedup* relative to the reference sequential algorithm; *i.e.*, the ratio between the sequential and parallel runtime is a linear function of p.

The *quality* of a PRO-algorithm is measured by the range of values p can assume while linear speedup is maintained. This range is captured by an attribute of the model called the granularity function $\mathrm{Grain}(n)$. In other words, a PRO-algorithm with granularity $\mathrm{Grain}(n)$ is required to be fully *scalable* for all values of p such that $p = O(\mathrm{Grain}(n))$. The granularity function $\mathrm{Grain}(n)$ determines the quality of one PRO-algorithm over another relative to the same sequential time and space complexity. The higher the function value $\mathrm{Grain}(n)$ the better the algorithm. Note that since optimality (consequently linear speedup) is 'hard-wired' into the model, the runtime cannot be a quality measure for a PRO algorithm. However, in a sense, the time and space complexity of the reference sequential algorithm A_{seq} can also be seen as a quality measure of the PRO-algorithm. This means that the selection of the reference sequential algorithm is of significant importance.

The rest of the paper is organized as follows. In Section 2 we give an overview of existing parallel computation models and highlight their limitations. In Section 3 the PRO model is presented in detail and in Section 4 it is compared with a selection of existing parallel models. In Section 5 we illustrate how the model is used in design and analysis using the matrix multiplication problem as an example. In Section 6 we give a PRO-algorithm for one-to-all broadcast, as an example of a primitive communication routine found in a potential PRO library. Finally, we conclude the paper in Section 7 with some remarks.

2 Existing models and their limitations

There exists a plethora of parallel computation models in the literature. On the theoretical end, we find the Parallel Random Access Machine (PRAM) model [8, 17] which in its simplest form posits a set of p processors, with global shared memory, executing the same program in lockstep. In this model, every processor can access any memory location at unit cost of time regardless of the memory location. This assumption is in obvious disagreement with the reality of practical parallel computers.

However, despite its serious limitation of being an 'idealized' model of parallel computation, the standard PRAM model still serves as a theoretical framework for investigating the maximum possible computational parallelism in a given task. Specifically, on this model, the NC versus P-complete dichotomy [14] is used to reflect the ease/hardness of finding a parallel algorithm for a problem. Recall that NC denotes the class of problems which have PRAM-algorithms with polylogarithmic runtime and polynomial number of processors in the input size. A problem is said to be P-complete if an NC-algorithm for it would imply that all polynomial time sequential problems have NC-algorithms. The problem of whether or not $P = NC$ has long been an open problem.

The NC versus P-complete dichotomy has its own practical limitations. First, P-completeness does not depict a full picture of non-parallelizability since the runtime requirement for an NC parallel algorithm is so stringent that the classification is confined to the case where up to polynomial number of processors in the input size is available (fine-grained setting). For example, there are P-complete problems for which less ambitious, but still satisfactory, runtime can be obtained by parallelization in PRAM [23]. In a fine-grained setting, since the number of processors p is a function of the input size n, it is customary to express speedup as a function of n. Thus the speedup obtained using an NC-algorithm is sometimes referred to as exponential. In a coarse-grained setting, *i.e.*, the case where n and p are orders of magnitude apart, speedup is expressed as a function of only p and some recent results [4, 7, 9, 15] show that this approach is practically relevant. Second, an NC-algorithm is not necessarily work-optimal, and thus not resource-optimal considering runtime and memory space as resources that one wants to use efficiently. Third, even if we restrict ourselves to work-optimal NC-algorithms and apply Brent's scheduling principle, which says an algorithm in theory can be simulated on a machine with fewer processors by only a constant factor more work, implementations of PRAM algorithms often do not reflect this optimality in practice [6]. This is mainly because the PRAM model does not account for non-local memory access (communication), and a Brent-type simulation relies heavily on cheap communication.

To overcome the defects of the PRAM related to its failure of capturing real machine characteristics, the advocates of shared memory models propose several modifications to the standard PRAM model. In particular, they enhance the standard PRAM model by taking practical machine features such as memory access, synchronization, latency and bandwidth issues into account. Pointers to the PRAM family of models can be found in [21].

Critics of shared memory models argue that the PRAM family of models fail to capture the nature of existing parallel computers with *distributed* memory architectures. Examples of distributed memory computational models suggested as alternatives include the Postal Model [2] and the Block

Distributed Memory (BDM) model [18]. Other categories of parallel models such as low-level, hierarchical memory, and network models are briefly reviewed in [21].

A more recent category of parallel models is that of 'bridging' models, a notion popularized by Valiant with his introduction of the Bulk Synchronous Parallel (BSP) model [22]. The BSP model is a distributed memory coarse-grained model in which parallel computation proceeds as a sequence of barrier synchronized supersteps where local computation and communication are distinct rather than intermingled phases. Culler *et al.* [5] extended the BSP model by allowing asynchronous execution and better accounting for communication overhead. Their model is coined LogP, an acronym for the four parameters involved. A common feature of the BSP, LogP, and other related models is their lack of simplicity: each model involves relatively many parameters making analysis and design of algorithms cumbersome.

The Coarse Grained Multicomputer (CGM) model [4, 7] was later proposed in an effort to retain the advantages of BSP while keeping the model simple (making the number of parameters fewer). The BSP and its special case CGM have been the primary inspirations for our model. Thus, we believe that many optimal CGM and BSP algorithms can easily be adapted to PRO.

The PRO model attempts to partially address the limitations of existing parallel models highlighted in the foregoing discussion and compromises between theoretical and practical considerations. One of its advantages from a theoretical point of view is that it is a step forward towards the identification of the class of problems for which 'good' parallel algorithms exist in a more realistic (practical) way than the existing NC versus P-complete classification.

Our main goal in suggesting the PRO model is to enable the development of scalable and resource-optimal parallel algorithms and to simplify their analysis. The model identifies the salient features of a parallel algorithm that make its practical scalability and optimality highly likely. In this regard, it can be considered as a set of 'guidelines' for the algorithm designer in the quest for developing scalable and efficient parallel algorithms. Hence, PRO can be seen as a mix of a parallel computation model and a parallel algorithm design scheme which makes it biased towards the software side in its role as a bridging model.

3 The PRO model

The PRO model is an algorithm *design* and *analysis tool* used to deliver a practical, optimal, and scalable parallel algorithm relative to a specific sequential algorithm whenever this is possible. Let Time(n) and Space(n) denote the time and space complexity of a specific sequential algorithm for a given problem with input size n. The PRO model is defined to have the following attributes.

Machine The underlying machine is assumed to consist of p processors with $M = O(\frac{\text{Space}(n)}{p})$ private memory each, interconnected by some communication network (or shared memory) that can deliver messages in a point-to-point fashion. A message can consist of several machine words.

Coarseness We assume that $p \leq M$, *i.e.*, the size of the local memory of each processor is big enough to store p words.

Execution For any value $p = O(\text{Grain}(n))$, a PRO algorithm,

- consists of $O(\frac{\text{Time}(n)}{p^2})$ *supersteps*. A superstep consists of a local computation phase and an interprocessor communication phase. In particular, in each superstep, each processor
  - sends at most one message to every other processor,
  - sends and receives at most M words in total, and pays a unit of time per word sent and received,
  - performs local computation, and pays a unit of time per operation,
- has parallel runtime $\text{Time}(n, p) = O(\frac{\text{Time}(n)}{p})$.

Note that the *granularity* function Grain(n) is a quality measure of a PRO-algorithm.

As discussed in the LogP paper [5], technological factors are forcing parallel systems to converge towards systems formed by a collection of essentially complete computers connected by a robust communication network. The *machine* model assumption of PRO is consistent with this convergence and maps well on several existing parallel computer architectures. The memory requirement $M = O(\frac{\text{Space}(n)}{p})$ ensures that the space utilized by the underlying sequential algorithm is uniformly distributed among the p processors. Since we may, without loss of generality, assume that $\text{Space}(n) = \Omega(n)$, the implication is that the private memory of each processor is large enough to store its 'share' of the input and any additional space the sequential algorithm might require. When $\text{Space}(n) = \Theta(n)$, note that the input data must be uniformly distributed on the p processors. In this case the machine model assumption of PRO is similar to the assumption in the CGM model [7].

The *coarseness* assumption $p \leq M$ is consistent with the structure of existing parallel machines and machines to be built in the foreseeable future. The assumption is required to simplify the implementation of collecting messages (from possibly all other processors) on a single processor.

The *execution* of a PRO-algorithm consists of a sequence of *supersteps* (or rounds). The *length* of (time spent in) a superstep on each processor is determined by the sum of the time used for communication and the time used for local computation. The length of a superstep s in the parallel algorithm seen as a whole, denoted by $\text{Time}_s(n, p)$, is the maximum over the lengths of the superstep on all processors. We can conceptually think as if the supersteps are synchronized by a barrier set at the end of the longest superstep across the processors. However, note that in PRO the processors are not in reality required to synchronize at the end of each superstep. The parallel runtime $\text{Time}(n, p)$ of the algorithm is the sum of the lengths of all the supersteps. Notice that the hypothetical barriers result in only a constant factor more time compared with an analysis that does not assume the barriers.

In PRO, since a processor sends at most one message to every other processor in each superstep, each processor is involved in at most $2(p-1)$ messages per superstep. Therefore, the requirement $\text{Steps} = O(\frac{\text{Time}(n)}{p^2})$ on the number of supersteps implies that the overall time paid per processor for *communication overhead* and *latency* is $O(\text{Time}(n)/p)$ and hence can be neglected from the analysis since our goal is to achieve an $O(\text{Time}(n)/p)$ parallel runtime. Notice that the bandwidth restriction of the underlying architecture which in turn contributes to the communication cost is accounted for since each processor pays a unit of time per word sent and received. This is not an unrealistic assumption noting that the network throughput (accounted in machine words) on modern architectures such as high performance clusters is relatively close to the CPU frequency and to the CPU/memory bandwidth.

The condition $\text{Time}(n, p) = O(\frac{\text{Time}(n)}{p})$ requires that a PRO-algorithm be optimal and yield linear speedup relative to the sequential algorithm used as a reference. This requirement ensures the potential practical use of the parallel algorithm.

Observation 1 *A PRO algorithm relative to a sequential algorithm with runtime $O(Time(n))$ and space requirement $O(Space(n))$ has maximum granularity $Grain(n) = O(min\{\sqrt{Space(n)}, \sqrt{(Time(n)}\}) = O(\sqrt{Space(n)})$. A PRO algorithm that achieves this is said to have optimal grain.*

Observation 1 is due to the limit on the memory size of each processor, the coarseness assumption, and the bound on the number of supersteps. The limit on the size of the private memory of each processor ($M = O(\frac{\text{Space}(n)}{p})$) together with the coarseness assumption $p \leq M$ imply $p = O(\sqrt{\text{Space}(n)})$. The fact that the number of supersteps of a PRO-algorithm should be Steps $= O(\text{Time}(n)/p^2)$, gives $p = O(\sqrt{(\text{Time}(n)/\text{Steps})})$ upon resolving and we clearly have Steps ≥ 1. Finally, note that Time$(n) \geq$ Space(n), since an algorithm has to at least read the input.

Since a PRO-algorithm yields linear speedup for any $p = O(\text{Grain}(n))$, a result like Brent's scheduling principle is implicit for these values of p. But Observation 1 shows that we cannot start with an arbitrary number of processors and efficiently simulate on a fewer number. So Brent's scheduling principle does not hold with full generality in the PRO model, which is in accordance with practical observations.

The design of a PRO-algorithm may sometimes involve subroutines for which there do not exist sequential counterparts. Examples of such tasks include communication primitives such as broadcasting, data (re)-distribution routines, and load balancing routines. Such routines are often required in various parallel algorithms. With a slight abuse of notation, we call such parallel routines PRO-algorithms if the overall computation and communication cost is linear in the input size to the routines.

4 Comparison with other models

In this section we compare the PRO model with PRAM, QSM, BSP, LogP, and CGM. Our tabular format for comparison is inspired by a similar presentation in [13], where the Queuing Shared Memory (QSM) model is proposed. The columns of Table 1 are labeled with the names of the selected models in our comparison and some relevant features of a model are listed along the rows.

The synchrony assumption of the model is indicated in the row labeled *synch.* Lock-step indicates that the processors are fully synchronized at each step (of a universal clock), without accounting for synchronization. Bulk-synchrony indicates that there can be asynchronous operations between synchronization barriers. The row labeled *memory* shows how the model views

	PRAM [8]	QSM [13]	BSP [22]	LogP [5]	CGM [4]	PRO
synch.	lock-step	bulk-synch.	bulk-synch.	asynch.	asynch.	asynch.
memory	sh.	sh.	dist.	dist.	priv.	priv.
commun.	SM	SM	MP	MP	MP/SM	MP/SM
parameters	n	p, g, n	p, g, L, n	p, g, l, o, n	p, n	p, n, A_{seq}
granularity	fine	fine	coarse	fine	coarse	Grain(n)
speedup	NA	NA	NA	NA	NA	$\Theta(p)$
optimal	NA	NA	NA	NA	NA	rel. A_{seq}
quality	time	time	time	time	rounds	Grain(n)

Table 1: Comparison of parallel computational models

the memory of the parallel computer: sh. indicates globally accessible shared memory, dist. stands for distributed memory and priv. is an abstraction for the case where the only assumption is that each processor has access to private (local) memory. In the last variant the whole memory could either be distributed or shared. The row labeled *commun.* shows the type of interprocessor communication assumed by the model. Shared memory (SM) indicates that communication is effected by reading from and writing to a globally accessible shared memory. Message-passing (MP) denotes the situation where processors communicate by explicitly exchanging messages in a point-to-point fashion. The MP abstraction hides the details of how the message is routed through the interprocessor communication network.

The parameters involved in the model are indicated in the row labeled *parameters*. The number of processors is denoted by p, n is the input size, A_{seq} is the reference sequential algorithm, l is the communication cost (latency), L is a single parameter that accounts for the sum of latency (l) and the cost for a barrier synchronization, g is the bandwidth gap, and o is the overhead associated with sending or receiving a message. Note that the machine characteristics l and o are are taken into account in PRO, even though they are not explicitly used as parameters. Latency is taken into consideration since the length of a superstep is determined by the sum of the computational and communication cost. Communication overhead is hidden by the PRO-requirement that states Steps $= O(\frac{\text{Time}(n)}{p^2})$.

The row labeled *granularity* indicates whether the model is fine-grained, coarse-grained or a more precise measure is used. We say that a model is coarse-grained if it applies to the case where $n \gg p$ and call it fine-grained if it relies on using up to a polynomial number of processors in the input size. In PRO granularity is exactly the quality measure Grain(n), and appears as one of the attributes of the model.

The rows labeled *speedup* and *optimal* indicate the speedup and resource optimality requirements imposed by the model. Whenever these issues are not directly addressed by the model or are not applicable, the word 'NA' is used. Note that these requirements are 'hard-wired' in the model in the case of PRO. The label 'rel. A_{seq}' means that the algorithm is optimal relative to the time and space complexity of A_{seq}. We point out that the goal in the design of algorithms using the CGM model [7, 4] is usually stated as that of achieving optimal algorithms, but the model *per se* does not impose an optimality requirement.

The last row indicates the *quality* measure of an algorithm designed using the different models. For all other models except CGM and PRO, the quality measure is running time. In CGM, the number of supersteps (rounds) is usually presented as a quality measure. In PRO the quality measure is granularity, one of the features that make PRO fundamentally different from all existing parallel computation models.

5 Algorithm example: matrix multiplication

In this section we illustrate how the PRO model is used, by starting from a given sequential algorithm and then designing and analyzing a parallel algorithm relative to it. We use the standard matrix multiplication algorithm with three nested for-loops as an example. This example is chosen for its simplicity and since our objective at this stage is to illustrate the use of a new model rather than solving a "difficult" problem.

Consider the problem of computing the product C of two $m \times m$ matrices A and B (input size $n = m^2$). We want to design a PRO-algorithm relative to the standard sequential matrix multiplication algorithm which has Time$(n) = O(n^{\frac{3}{2}})$ and Space$(n) = O(n)$.

We assume that the input matrices A and B are distributed among the p processors $P_0, ..., P_{p-1}$ so that processor P_i stores rows (respectively columns) $\frac{m}{p} \cdot i + 1$ to $\frac{m}{p} \cdot (i+1)$ of A (respectively B). The output matrix C will be row-partitioned among the p processors in a similar fashion. Notice that with this data distribution each processor can, without communication, compute a block of $\frac{m^2}{p^2}$ of the $\frac{m^2}{p}$ entries of C expected to reside on it. In order to compute the next block of $\frac{m^2}{p^2}$ entries, processor P_i needs the columns of matrix B that reside on processor P_{i+1}. In each superstep the processors in the PRO algorithm will therefore exchange columns in a round-robin fashion and then each will compute a new block of results. Note that each column exchanged in a superstep constitutes one single message. Note also that the initial distribution of the rows of matrix A remains unchanged. In Algorithm 1, we have organized this sequence of computation and communication steps in a manner that meets the requirements of the

```
Algorithm 1: Matrix multiplication
  Input: Two $m \times m$ matrices $A$ and $B$. The rows (columns) of $A$
         ($B$) are divided into $m/p$ contiguous blocks, and stored on
         processors $P_0, P_1, \ldots P_{p-1}$ respectively
  Output: The product matrix $C$ where the rows are stored in con-
          tiguous blocks across the $p$ processors
  for superstep $s = 1$ to $p$ do
      foreach processor $P_i$ do
          $P_i$ computes the local sub-matrix product of its rows and
          current columns;
          $P_{(i+1) mod\ p}$ sends its current block of columns to $P_i$;
          $P_i$ receives a new current block of columns from $P_{(i+1) mod\ p}$;
```

PRO model.

Algorithm 1 has p supersteps (Steps $= p$). In each superstep, the time spent in locally computing each of the m^2/p^2 entries is $\Theta(m)$ resulting in local computing time $\Theta(m^3/p^2) = \Theta(n^{\frac{3}{2}}/p^2)$ per superstep. Likewise, the total size of data (words) exchanged by each processor in a superstep is $\Theta(m^2/p) = \Theta(n/p)$. Thus, the length of a superstep s is $\text{Time}_s(n,p) = \Theta(n^{\frac{3}{2}}/p^2 + n/p)$. Note that for $p = O(\sqrt{n})$, $\text{Time}_s(n,p) = \Theta(n^{\frac{3}{2}}/p^2)$. Hence, for $p = O(\sqrt{n})$, the overall parallel runtime of the algorithm is

$$\text{Time}(n,p) = \sum_{\text{Steps}} \Theta(n^{\frac{3}{2}}/p^2) = \Theta(n^{\frac{3}{2}}/p) = \Theta(\text{Time}(n)/p). \qquad (1)$$

Noting that $\text{Space}(n) = \Theta(n)$, we see that the memory restriction of the PRO model is respected, *i.e.*, each processor has enough memory size to handle the transactions. In order to be able to neglect communication overhead, the condition on the number of supersteps, which in this case is just p, should be met. In other words, we need $p = O(\text{Time}(n)/p^2) = O(n^{\frac{3}{2}}/p^2)$, which is true for $p = O(\sqrt{n})$. Thus the granularity function of the PRO-algorithm is $\text{Grain}(n) = \sqrt{n}$.

In summary,

Lemma 1 *Multiplication of two m by m matrices has a PRO-algorithm with Grain$(n) = m$ relative to a sequential algorithm with Time$(n) = m^3$ and Space$(n) = m^2$ (input size $n = m^2$).*

From Observation 1, we note that Algorithm 1 achieves optimal granularity. Note that on a relaxed model, where the assumption that $p \leq M$ is not present, the strong regularity of matrix multiplication and the exact knowledge of the communication pattern allows for algorithms that have an even finer granularity than m. For example, a systolic matrix multiplication algorithm has a granularity of m^2. However, PRO is intended to be applicable for general problems and practically relevant parallel systems.

6 Communication primitive example: one-to-all broadcast

A good parallel computation model should have a selection of algorithms for primitive communication tasks available in its algorithm design toolbox. The PRO model is intended to meet this demand, but for lack of space we give only one example.

In this section we illustrate how the PRO model allows optimal one-to-all broadcasting among its processors. Since there is no sequential basis algorithm in this case, we want an algorithm whose overall communication and computation cost is linear in the input and output sizes. More precisely, we consider the situation where the input consists of a vector of size m on a single processor and the output should be a copy of this vector on each of the p processors, and we want an algorithm that achieves this in $O(m)$ time using $O(m)$ memory on each processor. See Algorithm 2.

Algorithm 2: One-to-All Broadcast

```
    Input: A vector $V$ of size $m$ on processor $P_0$
    Output: A copy of $V$ on each processor
S1  $P_0$ divides $V$ into $p$ equal sized parts;
    $P_0$ sends the $i^{th}$ part of $V$ to processor $P_i$, for each $0 < i \leq p$;
    foreach processor $P_i$, $i > 0$ do  processor $P_i$ receives the $i^{th}$ part
    from $P_0$;
S2  foreach processor $P_i$ do
        $P_i$ sends out the $i^{th}$ part to $P_j$, for each $j \neq i$ and $0 < j \leq p$.
    foreach processor $P_j$,$j \neq 0$ do
        $P_j$ receives the $i^{th}$ part from $P_i$, for each $i \neq j$ and $0 < i \leq p$
```

Lemma 2 *PRO Algorithm 2 implements a one-to-all broadcast of m memory words in two supersteps using $O(m)$ time and $O(m)$ space per processor, for any number of processors $p \leq m$.*

Proof: First, we note that the algorithm correctly broadcasts the desired vector V, while observing the space restriction, in two supersteps. We turn to the timing. In step **S1** processor P_0 in total sends out $(p-1)m/p$ words and each of the other processors receives a message of size m/p. In step **S2** processor P_i in total sends out $\frac{p-2}{p}m$ words. Processor P_j, $j \neq 0$, in total receives $\frac{p-1}{p}m$ words.

The total time is dominated by the communication which is

$$(p-1)m/p + m/p + \frac{p-2}{p}m + \frac{p-1}{p}m = \tag{2}$$

$$m/p(p+p-2+p-1) < 3m \tag{3}$$

for total time $O(m)$ as claimed. □

7 Conclusion

We have introduced a new parallel computation model (called PRO) that enables the development of efficient scalable parallel algorithms and simplifies the complexity analysis of such algorithms.

The distinguishing feature of the PRO model is the novel focus on relativity, resource-optimality, and a new quality measure (granularity). In particular, the model requires a parallel algorithm to be both time- and space-optimal relative to an underlying sequential algorithm. Having optimality as a built-in requirement, the quality of a PRO-algorithm is measured by the maximum number of processors that could be used while the optimality of the algorithm is maintained.

The focus on relativity has theoretical as well as practical justifications. From a theoretical point of view, the performance evaluation metrics of a parallel algorithm includes speedup and optimality, both of which are always expressed relative to some sequential algorithm. Moreover, there is an inherent asymmetry between sequential and parallel computation. A parallel algorithm would always imply a sequential algorithm, whereas the converse is usually not true. Thus, in a sense, it is natural to think of an underlying sequential algorithm whenever one speaks of a parallel algorithm. From a practical point of view, one notes that the development of a parallel algorithm is often built on some known sequential algorithm.

The fact that optimality is incorporated as a requirement in the PRO model enables one to concentrate only on parallel algorithms that are practically useful.

However, the PRO model is not just a collection of some 'ideal' features of parallel algorithms, it is also a means to achieve these features. In particular, the attributes of the model capture the salient characteristics of a parallel algorithm that make its practical optimality and scalability highly likely.

In this sense, it can also be seen as a parallel algorithm design scheme. Moreover, the simplicity of the model eases analysis.

We believe that the PRO model is a step forward towards the identification of problems for which 'practically good' parallel algorithms exist. Much work remains to be done, and we hope that other members of the research community will join in. As a first item on the agenda, the PRO model needs to be tested for compatibility with already existing practical parallel algorithms.

Acknowledgments We are grateful to the anonymous referees for their helpful comments.

References

[1] A. G. Alexandrakis, A. V. Gerbessiotis, D. S. Lecomber, and C. J. Siniolakis. Bandwidth, space and computation efficient PRAM programming: The BSP approach. In *Proceedings of the SUP'EUR '96 Conference, Krakow, Poland*, September 1996.

[2] A. Bar-Noy and S. Kipnis. Designing broadcasting algorithms in the Postal Model for message passing systems. In *The 4th annual ACM symposium on parallel algorithms and architectures*, pages 13–22, July 1992.

[3] R. P. Brent. The parallel evaluation of generic arithmetic expressions. *Journal of the ACM*, 21(2):201–206, 1974.

[4] E. Caceres, F. Dehne, A. Ferreira, P. Locchini, I. Rieping, A. Roncato, N. Santoro, and S. W. Song. Efficient parallel graph algorithms for coarse grained multicomputers and BSP. In *The 24th International Colloquium on Automata Languages and Programming*, volume 1256 of *LNCS*, pages 390–400. Springer Verlag, 1997.

[5] D. E. Culler, R. M. Karp, D. A. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a realistic model of parallel computation. In *4th ACM SIGPLAN Symposium on principles and practice of parallel programming, San Diego, CA*, May 1993.

[6] F. Dehne. Coarse grained parallel algorithms. *Algorithmica Special Issue on "Coarse grained parallel algorithms"*, 24(3/4):173–176, 1999.

[7] F. Dehne, A. Fabri, and A. Rau-Chaplin. Scalable parallel computational geometry for coarse grained multicomputers. *International Journal on Computational Geometry*, 6(3):379–400, 1996.

[8] S. Fortune and J. Wyllie. Parallelism in random access machines. In *10th ACM Symposium on Theory of Computing*, pages 114–118, May 1978.

[9] A. H. Gebremedhin, I. Guérin Lassous, J. Gustedt, and J. A. Telle. Graph coloring on a coarse grained multiprocessor. In Ulrik Brandes and Dorothea Wagner, editors, *WG 2000*, volume 1928 of *LNCS*, pages 184–195. Springer-Verlag, 2000.

[10] A. V. Gerbessiotis, D. S. Lecomber, C. J. Siniolakis, and K. R. Sujithan. PRAM programming: Theory vs. practice. In *Proceedings of 6th Euromicro Workshop on Parallel and Distributed Processing, Madrid, Spain.* IEEE Computer Society Press, January 1998.

[11] A. V. Gerbessiotis and C. J. Siniolakis. A new randomized sorting algorithm on the BSP model. Technical report, New Jersey Institute of Technology, 2001.

[12] A. V. Gerbessiotis and L. G. Valiant. Direct bulk-synchronous parallel algorithms. *Journal of Parallel and Distributed Computing*, 22:251–267, 1994.

[13] P. B. Gibbons, Y. Matias, and V. Ramachandran. Can a Shared-Memory Model Serve as a Bridging Model for Parallel Computation? *Theory of Computing Systems*, 32(3):327–359, 1999.

[14] R. Greenlaw, H.J. Hoover, and W. L. Ruzzo. *Limits to Parallel Computation: P-Completeness Theory.* Oxford University Press, New York, 1995.

[15] I. Guérin Lassous, J. Gustedt, and M. Morvan. Handling graphs according to a coarse grained approach: Experiments with MPI and PVM. In Jack Dongarra, Péter Kacsuk, and N. Podhorszki, editors, *7th European PVM/MPI Users' Group Meeting*, volume 1908 of *LNCS*, pages 72–79. Springer Verlag, 2000.

[16] K. Hawick et al. High performance computing and communications glossary. see `http://nhse.npac.syr.edu/hpccgloss/`.

[17] J. Jájá. *An Introduction to Parallel Algorithms.* Addison-Wesley, 1992.

[18] J. JáJá and K. W. Ryu. The Block Distributed Memory model. *IEEE Transactions on Parallel and Distributed Systems*, 8(7):830–840, 1996.

[19] R. M. Karp and V. Ramachandran. Parallel Algorithms for Shared-Memory Machines. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, Algorithms and Complexity, pages 869–941. Elsevier Science Publishers B.V., Amsterdam, 1990.

[20] C. P. Kruskal, L. Rudolph, and M. Snir. A complexity theory of efficient parallel algorithms. *Theoretical Computer Science*, 71(1):95–132, march 1990.

[21] B. M. Maggs, L. R. Matheson, and R. E. Tarjan. Models of parallel computation: A survey and synthesis. In *28th HICSS*, volume 2, pages 61–70, January 1995.

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

[23] J. S. Vitter and R. A. Simons. New classes for parallel complexity: A study of unification and other complete problems for P. *IEEE Transactions on Computers*, C-35(5):403–418, 1986.