A novel MPI reduction algorithm resilient to imbalances in process arrival times

P. Marendic[a,c,*], J. Lemeire[a,1,d], D. Vucinic[b], P. Schelkens[a,c,1]

[a] *Vrije Universiteit Brussel, Dept. of Electronics and Informatics (ETRO), Pleinlaan 2, B-1050 Brussels, Belgium*
[b] *Vrije Universiteit Brussel, Department of Mechanical Engineering, Pleinlaan 2, B-1050 Brussels, Belgium*
[c] *iMinds, Dept. of Multimedia Technologies, Gaston Crommenlaan 8, B-9050 Ghent, Belgium*
[d] *Vrije Universiteit Brussel, Dept. of Industrial Sciences (INDI), Pleinlaan 2, B-1050 Brussels, Belgium*

Abstract

Reduction algorithms are optimized only under the assumption that all processes commence the reduction simultaneously. Research on process arrival times has shown that this is rarely the case. Thus, all benchmarking methodologies that take into account only balanced arrival times might not portray a true picture of real-world algorithm performance. In this paper, we select a subset of four reduction algorithms frequently used by library implementations and evaluate their performance for both balanced and imbalanced process arrival times. The main contribution of this paper is a novel imbalance robust algorithm that uses pre-knowledge of process arrival times to construct reduction schedules. The performance of selected algorithms was empirically evaluated on a 128 node subset of the PRACE CURIE supercomputer. The reported results show that the new imbalance robust algorithm universally outperforms all the selected algorithms, whenever the reduction schedule is precomputed. We find that when the cost of schedule construction is included in the total runtime, the new algorithm outperforms the selected algorithms for problem sizes greater than 1 MiB.

Keywords: Reduction, MPI, load imbalance, collective operations, system noise, process arrival time

1. Introduction

Reduction is a common collective operation in distributed memory applications whose performance often plays a critical role in parallel applications. Of the total communication volume of LeanMD (a molecular dynamics benchmark), 51.18% can

*Corresponding author
Email addresses: petar.marendic@etro.vub.ac.be (P. Marendic), jan.lemeire@etro.vub.ac.be (J. Lemeire), dean.vucinic@vub.ac.be (D. Vucinic), peter.schelkens@etro.vub.ac.be (P. Schelkens)
[1] Research director at iMinds.

be attributed to reduction [1]. Similar findings have been reported by [2] for several prominent HPC applications like CTH, SAGE and POP.

Therefore, a great deal of research effort has been conducted in the design of optimized reduction operation implementations. However, state-of-the-art reduction algorithms remain largely optimized only for the case where processes call (arrive at) the collective operation simultaneously. Such Process Arrival Times (PATs) are said to be balanced. Yet, balanced PATs are extremely rare and generally only occur immediately after synchronization routines [3]. That process arrival times can have an impact on the performance of collective operations has been largely overlooked by the research community. The reason for this is the perception that imbalanced PATs and optimization of collective operations are disjunct problems that can be addressed independently.

While there have been efforts to seamlessly integrate load balancing into Message Passing Interface (MPI), such as the Adaptive Message Passing Interface (AMPI) virtualization based approach [4], there has been some, but hardly comprehensive effort in designing collective operations that would be more robust to imbalanced PATs [5, 3, 6, 7]. It is reasonable to expect that the move towards exascale computing will only exacerbate the problem further.

In this paper, we present a PAT aware algorithm, that we thus term Clairvoyant, which constructs reduction schedules of minimum possible length for both atomic and non-atomic input data. Unlike other approaches that require different algorithms depending on problem size and process count, our algorithm is equally applicable to small and large problem sizes, with or without segmentation.

By reordering the reduction schedule, the algorithm mitigates the impact of PAT imbalance by performing as much of the reduction operation with those processes that are available. We compare its performance against a selection of four reduction algorithms frequently found in MPI library implementations. We ensure that all algorithm implementations adhere to the function interface and semantics defined by the MPI standard [8] for `MPI_Reduce`. We perform the experiments on a 128 node subset of the PRACE CURIE supercomputer, a Bull x86 system built on top of a fat-tree Infiniband network interconnect. In addition to the new algorithm, this paper introduces a new collective operation benchmarking methodology, designed to evaluate the operations' performance both for balanced and imbalanced PATs.

The paper is structured as follows. The next section surveys known reduction algorithms and reviews existing work on the problem of system noise and imbalanced process arrival times. Section 3 defines the network model, together with notions of algorithm runtime and process arrival time patterns. The subsequent section discusses imbalanced PATs and presents arguments in favour of clairvoyant collective operations. Section 5 introduces a model for expressing the computational complexity of reduction algorithms, followed by a detailed presentation and analysis of the new Clairvoyant reduction algorithm. The section concludes with a discussion on the four selected adversarial reduction algorithms, commonly found in MPI library implementations. The following section elaborates the experimental methodology. Section 7 summarizes the main findings and discusses the implications pertaining to implementations of reduction algorithms. Finally, Section 8 concludes the paper.

2. Related Work

One of the driving constraints in implementations of reduction operations is the atomicity of input data. An optimal reduction algorithm for atomic (non-segmentable) data was first presented by Karp in [9]. The authors assumed a fully connected homogeneous network and balanced process arrival times. The paper showed that the optimal algorithm for both operations is one that sends no redundant messages and has no unforced delays in sending or receiving messages. Such an algorithm thus sends messages as soon as it can and as often as it can. The authors in [10] present a greedy algorithm that preconstructs reduction schedules for homogeneous networks with communication-computation overlapping. Their algorithm constructs binomial tree reduction schedules if either the cost of communication or the cost of computation is zero. When the two costs are equal, the algorithm constructs a Fibonacci tree reduction schedule.

When data are non-atomic, implementations are based either on pipelined tree reductions or composite algorithms based on reduce-scatter and gather operations. A prominent example of a composite algorithm is the butterfly algorithm elaborated in [11]. Further improvements on this idea can be found in [12] with the additional focus on the non-power-of-two number of processes. Another composite algorithm well suited for large input data is the bucket or Parallel Ring algorithm [13, 14, 15]. In the domain of parallel volume rendering, where input data is typically in the order of 4 MiB-128 MiB an algorithm called Radix-k [16, 17] has been shown to outperform the commonly used butterfly algorithm. Radix-k is essentially a hybrid butterfly (a.k.a. binary-swap) and direct send algorithm that is configurable and adaptable to different topologies and network interconnects, able to take advantage of higher degrees of network concurrency if available.

Pipeline tree algorithms are simple to implement and typically come in the form of linear tree pipelines or binary trees. Linear pipeline algorithms are known to perform well for large input data and small to moderate number of processes, but do not scale well with large number of processes [18]. An improvement to the binary tree pipeline algorithm that exploits the full-duplex potential of modern networks was proposed in [19]. The authors report a near twofold speedup for their implementation compared with the pipelined binary tree reduction.

A rather comprehensive treatise on the general problem addressing the performance of MPI collectives and their implementations can be found in [15]. Here, the authors distinguish several different network topologies (linear, mesh, hypercubes, fully connected) and suggest an optimal solution for each collective operation over every considered topology. Another comprehensive work on the implementation of collectives in the MPICH library is that of Thakur et al. [20].
Pjesivac et al. [18] present an in-depth analysis of collective operations performance using several frequently used models of parallel communication such as Hockney, LogP/LogGP and PLog. Hoefler and Moor [21] survey a large body of collective operations implementations and present models of their performance, energy and memory costs.

2.1. System noise

System noise, a result of operating system level interrupts and various other architectural overheads, is a common source of performance degradations on large systems. Petrini et al. [22] succeeded in putting this issue into the spotlight by showing how interrupts from the system kernel and various daemons can lead to very big slowdowns of bulk synchronous (iterative compute-communicate phases) applications when run on a large number of processors. Their results have spurred much research on the topic: Agarwal et al. [23] performed a theoretical analysis of the potential impact of three distributions of system noise (exponential, heavy-tailed and Bernoulli) might have on the performance of MPI collectives. Their results show that most systems are expected to scale well under exponentially distributed noise, while the heavy-tailed and Bernoulli distributed noise are expected to incur significant performance penalties. Hoefler et al. [24] introduced an OS noise measurement and simulation framework and analyzed the impact such noise might have on large-scale applications. The authors performed simulated runs with up to 1 million processes and have shown that the scale at which system noise becomes a bottleneck is system specific and primarily dependent on its distribution. However, there is a clear trend of increased noise amplification with increasing system size. This research also showed the effect system noise can have on various collective operations and identified that allreduce is a particularly sensitive one, due to its tendency to amplify system noise. That allreduce is a sensitive collective operation was reported earlier by [2] where the authors have implemented a kernel injection noise generation system. They have shown a slowdown of 2000% for a loaded schedule noise signature (high duration, low frequency) for the Parallel Ocean Program (POP) due to noise amplification. This particular application spends the majority of its runtime in the allreduce MPI collective. The authors have also conjectured and confirmed that the more hardware imbalanced a given system is (higher computation to communication ratio), the less susceptible it might be to OS noise.

2.2. Non-blocking collective operations

Recently, The MPI Forum has released the MPI-3 standard, whose major novelties are non-blocking collectives. These present an alternative way of dealing with system noise and potentially also imbalanced PATs. The authors in [25] present a modification of the GMRES algorithm where they overlap the dot-product global reduction communication with SpMV. They reported significant speedups compared to standard GMRES for strong scaling experiments and were mentioned in the "Report on the Workshop on Extreme-Scale Solvers: Transition to Future Architectures" by the U.S. Department of Energy as a "...new class of algorithms presenting significant opportunities for fundamental research". The algorithm is now included with the widely used PETSc library. The ability of non-blocking collectives to combat system noise was confirmed by research in [26], where the authors implemented a non-blocking `MPI_Allreduce` and showed that with sufficient overlap between application-computation and collective communication, the effects of system noise can be almost entirely dampened. However, non-blocking collectives are not without limitations: computation needs to be independent of communication for this to work, or the core algorithms need to be rewritten accordingly. Consequently, most legacy code needs to be rewritten to take advantage of what non-blocking collectives have to offer.

2.3. Related work on imbalanced PATs

That imbalanced process arrival times can have adverse performance impacts on collective operations has been long known. While imbalance resilient algorithms for collective operations have been long proposed for shared memory architectures [27], perhaps the first to propose imbalance resilient algorithms in the domain of distributed memory machines were Mamidala et al. [5]. The authors implement imbalance resilient barrier and allreduce algorithms that use hardware multicasts to dynamically re-arrange the tree topologies inherent to the algorithms. A more comprehensive study of imbalanced PATs on application performance was reported by Faraj et al [3]. The authors have examined a set of NAS parallel benchmarks and identified large imbalances in PATs at collective operation call sites. A startling result of their study is that even with explicit load balancing, it is difficult to fully eliminate imbalanced PATs in cluster environments. A common observation in both of the papers is that algorithms that perform better in absence of imbalance tend to perform worse in the presence of imbalance. The study by Faraj et al. conclusively showed that the performance of collective operations is sensitive to process arrival time. Another interesting result of this study is that in most of the examined applications, the patterns of PAT imbalance at collective call sites remain highly correlated for sustained durations. The same authors in [6] present two algorithms for the collective broadcast operation with focus on large messages, where the overhead of control messages in their implementation is reduced. A more efficient RDMA-based solution for alltoall and allgather is proposed in [7] that can handle both small and large messages without overhead.

Compared to non-blocking collectives, imbalance resilient collectives have the advantage of offering immediate benefit to legacy code without any code changes. The work in this paper builds upon our prior work on imbalance robust reduction algorithms [28]. The principal idea behind that algorithm and the one presented in this paper is similar: re-arangement of the reduction schedule in such a way that would allow the early arriving processes to start exchanging data and solving the reduction problem as soon as possible. This principle is illustrated in Fig. 1.

However, the Local Redirect algorithm presented in [28] is designed for atomic input data, while the new imbalance robust algorithm presented in this paper can be equally well applied for both atomic and non-atomic input data that can be arbitrarily segmented. Another important difference is that the new algorithm (henceforward called Clairvoyant) requires pre-knowledge of PATs to produce an optimized reduction schedule, while the Local Redirect algorithm does not.

3. Problem definition and runtime of collective operations

We start this section by defining the network model for message passing. Following that, we define the research question. Finally, we detail our understanding of collective operation runtime and process arrival times.

Definition 3.1. Assume a fully connected, single-port homogeneous network of compute nodes (processes) with full duplex communication links. In such a network, any process can at the same time send and receive one message, possibly from/to different peers. The homogeneity of the network ensures that the communication cost between

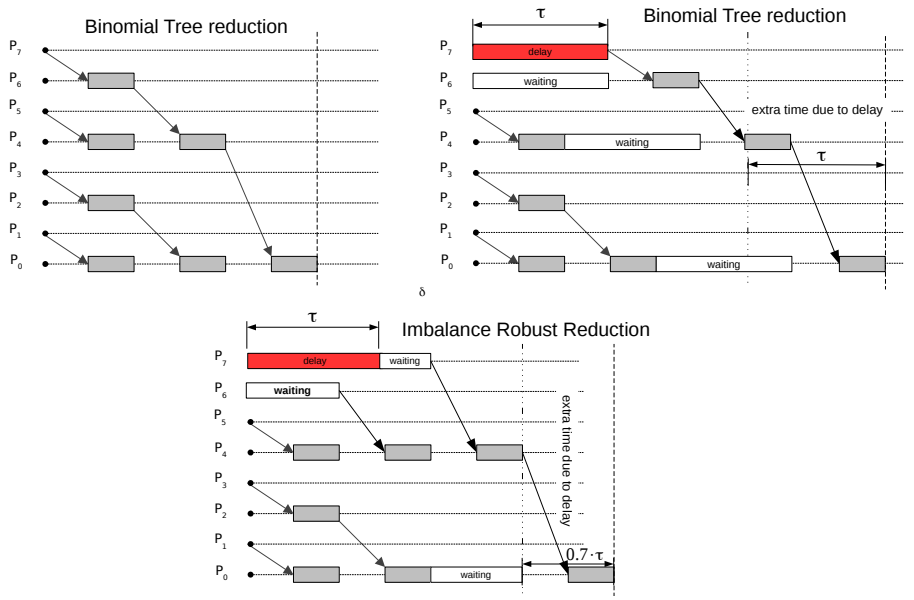

Figure 1: Impact of a single delayed process on the runtime of the binomial tree reduction algorithm. The imbalance robust algorithm uses a greedy strategy to re-order the reduction schedule. In this strategy, no process waits if there is another ready process. As a result, it manages to absorb nearly one third of the imbalance time τ

any two pairs of processes is identical. It is further assumed that while a process is combining messages, it cannot send or receive other messages. We call this the **homogeneous simultaneous send/receive no-overlap model**.

Because of the homogeneity, we assume that the system does not buffer small messages, and that a single communication protocol is used in message transmission. This network model is often adopted in the literature and considered appropriate for fully connected networks or modern fat-tree networks [29, 15, 30, 19]. The no-overlap assumption in Definition 3.1 was introduced due to the absence of computation-communication overlap capability on the PRACE CURIE machine (Section 6).

Definition 3.2. Consider a set of P processes numbered $0 \dots P-1$ distributed across a set of compute nodes, so that one and only one process is mapped onto each compute node. The compute nodes are spatially separated and communicate with one another by exchanging messages. Let each process i have a message m_i, where m_i is either a single value or an isotype vector of size m. We will refer to m as *problem size*. Consider now an associative commutative binary operator $\star$. We define the **P-way reduction problem** as the computation of the value $M = m_0 \star m_1 \star \cdots \star m_{P-1}$ that is made available at the root process 0 in the shortest time.

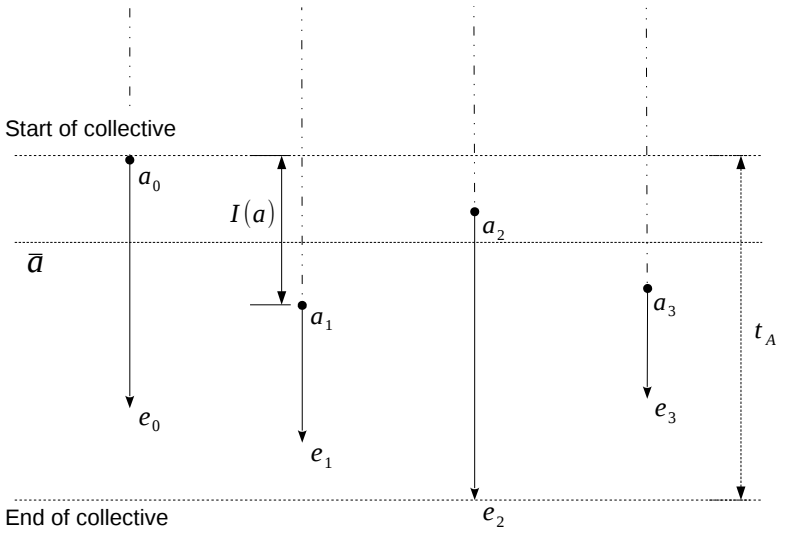

Figure 2: An illustration of process arrival time (represented with vector a), average arrival time $\bar{a}$, collective operation runtime $t_{\mathcal{A}}$ and absolute imbalance $I(a)$

To clarify what is meant by shortest time in Definition 3.2, we first have to define the starting conditions for the P-way reduction problem.

Definition 3.3. Let a_i denote the time when process i arrives at the collective call site, or in other words starts the collective operation. We define **process arrival times** (PAT) as the vector $a = (a_0, a_1, \ldots, a_{P-1})$.

The average arrival time is defined as $\bar{a} = \frac{a_0+a_1+\ldots a_{P-1}}{P}$. When all processes arrive at the same time, we say that the PATs are *balanced*. Otherwise, PATs are *imbalanced* (skewed).

Let e_i denote the time when process i exits the collective operation. We define the process exit time (PET) as the vector $e = (e_0, \ldots e_{P-1})$. Fig. 2 provides an illustration of these concepts.

There is no single definition of what constitutes collective operation runtime, nor how to measure it. Not all research papers define explicitly their understanding of collective operation runtime and often leave it implicit in their adopted runtime measure (estimation). This leads to measures and reported results that are sometimes incomparable. In all cases, estimation of collective operation runtime is performed by measuring the elapsed time between two distinguished events: the start and the end of the collective operation. These events either reside on the same process or on different processes. The former leads to local, while the latter to global measures. Definition 3.4 presents the understanding of collective operation runtime adopted in this paper.

Definition 3.4. Let $\mathcal{A}$ be an algorithm for *P-way reduction*. Let a be a PAT vector of size P. Without loss of generality, let $min(a) = 0$. Then the runtime of process i is denoted by its exit time e_i. We define the runtime of algorithm $\mathcal{A}$ for the PAT a and

communicator of size P to be:

$$t_{\mathcal{A}}(a) = \max(e) - \min(a) = \max(e)$$

In other words, we define the runtime of a collective operation as the time difference between the last process to exit and the first process to arrive at the collective operation. This is the same definition as adopted by the authors of MPIBlib [31].

Definition 3.5. Let a be a PAT vector $(a_0, a_1, \ldots, a_{P-1})$. We define $I(a)$, the **absolute imbalance** of a to be $\max(a) - \min(a)$, i.e. the time difference between the latest process to arrive and the earliest process to arrive.

The assumption of commutative and associative operator in Definition 3.2 is based on the fact that all 12 MPI combination operators, including sum, product, minimum, maximum, etc., are associative and commutative. However, The MPI standard encourages implementations to optimize for non-commutative operators as well. Problems can arise from using operators that are not "strictly" associative, such as most floating-point operations. The MPI standard [8] (Section 5.9.1., page 175) strongly encourages implementors to design algorithms such that the same result be obtained whenever the function is applied on the same arguments, appearing in the same order.

3.1. Absorption time

An algorithm that is not robust to an imbalance in the PAT will have its runtime prolonged by the magnitude of the absolute imbalance. On the other hand, an imbalance robust algorithm will mitigate, or absorb a part of the absolute imbalance, and thus suffer a smaller performance impact compared to an imbalance non-robust algorithm. We formalize this idea as Definition 3.6.

Definition 3.6. Let $\mathcal{A}$ be an algorithm for *P-way reduction* with problem size m. Let ψ be some PAT vector of size P. Let π be the balanced PAT $(0, 0, \ldots, 0)$. Then **absorption time** of algorithm $\mathcal{A}$ with respect to PAT ψ is defined as:

$$A(\psi, \mathcal{A}) = t_{\mathcal{A}}(\pi) - t_{\mathcal{A}}(\psi) + I(\psi)$$

If absorption time is equal to the absolute imbalance, then the algorithm will not exhibit any slowdown due to imbalance in process arrival times. Absorption time may also be negative, if a particular PAT has an adverse effect on the algorithm performance beyond that of the absolute imbalance. A particular case of this was observed in our performance experiment, as discussed in Section 7. It is interesting to observe that there is an upper bound on absorption time for any given algorithm.

Proposition 3.7.

$$A(\psi, \mathcal{A}) \leq t_{\mathcal{A}}(\pi) - t_{\mathcal{O}}(\pi, 2),$$

where $t_{\mathcal{O}}(\pi, 2)$ is the optimal time to solve the 2-way reduction problem.

Proof. The largest absorption time will be attained when by the time the last process (let that be process i) has become ready, only the input data m_i remain to be combined to derive the final result. Let us select the minimum absolute imbalance $I(\psi)$ for which

that can be the case, i.e. $I(\psi) = t_{\mathcal{A}}(\pi, P-1)$. Then, $t_{\mathcal{A}}(\psi) = I(\psi) + t_{\mathcal{O}}(\pi, 2)$. From this, it follows that

$$\begin{aligned} A(\psi, \mathcal{A}) &= t_{\mathcal{A}}(\pi) - t_{\mathcal{A}}(\psi) + I(\psi) \\ &= t_{\mathcal{A}}(\pi) - I(\psi) - t_{\mathcal{O}}(\pi, 2) + I(\psi) \\ &= t_{\mathcal{A}}(\pi) - t_{\mathcal{O}}(\pi, 2). \end{aligned}$$

□

Finally, it will be useful to normalize both the absolute imbalance and absorption time with respect to algorithm runtime for balanced PATs.

Definition 3.8. We define the normalized absolute imbalance to be

$$I_N(\psi) = \frac{I(\psi)}{t_{\mathcal{A}}(\pi)}$$

Definition 3.9. We define the normalized absorption time to be

$$A_N(\psi, \mathcal{A}) = \frac{A(\psi, \mathcal{A})}{t_{\mathcal{A}}(\pi)}$$

4. A case for a Clairvoyant Algorithm

In this section, we argue the feasibility of clairvoyant collective operations. Much of the argument will center on the assumption that it is feasible to provide information on PATs to collective operations at runtime. We discuss the difficulties present therein and the costs involved.

Faraj et al. [3] conducted a study of PATs at collective operation call sites for various MPI routines in a set of MPI benchmarks consisting of High Performance Computing (HPC) application kernels. They found that PATs for different invocations of the same collective operation exhibit a phased behaviour: PATs are strongly autocorrelated for a period of time before they change (Fig. 3). For some collective operation call sites, they found that PATs are autocorrelated for the entire program duration. These findings indicate that it might be feasible to construct a model in the form of a stochastic difference equation (such as ARMA) to predict PAT patterns from one invocation to the other.

Motivated by their findings, we performed a trace of the per process image rendering time across 100 iterations of the in-situ visualized Helsim[2] particle-in-cell space

[2]Helsim is an Electromagnetic Explicit 3D In-Situ-Visualized Resilient Particle-in-cell simulator, developed in the Leuven Intel ExaScale Lab, Belgium. It is a combined multidisciplinary effort integrating astrophysics, linear solvers, runtime environment, in-situ visualization and architectural optimization focused simulations. It was developed to be a proto-app, showing a realistic example of trade-offs between computation and communication on a small, manageable code-base with modern implementation techniques. It was implemented in C++11 utilizing the inlab Shark PGAS library for all distributed data structures and the Cobra library for load balancing and resiliency.

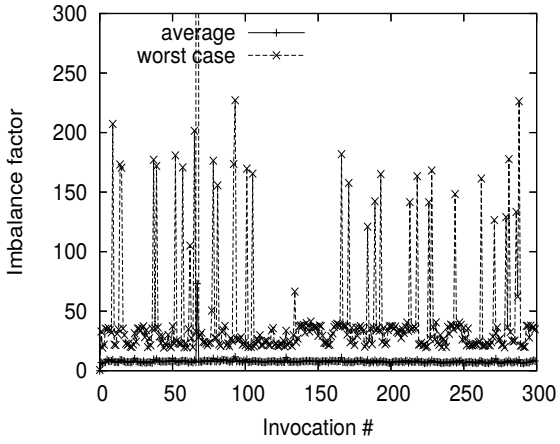

Figure 3: Imbalance factors for *MPI_Allgather* in NBODY on Lemieux cluster (P=128). Reprinted from "A study of Process Arrival Patterns for MPI Collective Operations", by Faraj A. Et al., 2008., International Journal of Parallel Programming. Volume 36, Issue 6, pp 543-570. Reprinted according to fair use

weather simulation on $P = 128$ processes with 8 processes per node, on the Lynx cluster machine. In sort-last distributed rendering, each process produces one full-sized image of the data that is subsequently composited into the final image with a global reduction operation. The variance in local image rendering time then manifests as an imbalance in PATs at the collective image compositing operation that immediately follows image rendering. A depiction of the variance evolution across simulation iterations is presented in Fig. 4.

The PATs of the first 24 processes, exhibited a recurring pattern (Fig. 5) with a period of 4 process ranks. The non-randomness and strong trends in the per-process PATs indicate that it might be feasible to construct a stochastic difference equation based model to predict PAT patterns in this setting. A simple moving average model (SMA) with a window size of 5 was shown to fit the data very well (Fig. 5). This conjecture is further reinforced by the autocorrelation plots of the data (Appendix B).

The clairvoyant schedule generation algorithm introduced in this paper requires that the entire PAT pattern be known at the time of schedule construction. This could ideally be accomplished in an iterative setting, by communicating the PAT pattern every k iterations to all the processes in the communicator with an *all_gather* operation, and relying on the model to predict the PAT patterns in-between the communications.

However, for this to be an efficient approach, the number of iterations k has to be sufficiently large so that the speedup brought about by the clairvoyant algorithm amortizes the PAT pattern dissemination cost. Because each process only communicates a single floating point value, the dissemination cost will become negligible for moderate to large problem sizes $m > 128\,\text{KiB}$, that are considered in this paper. Moreover, the clairvoyant schedule generation algorithm should be robust to the small inaccuracies produced by the model predictions. We provide preliminary results on the PAT misprediction sensitivity in Table 5.

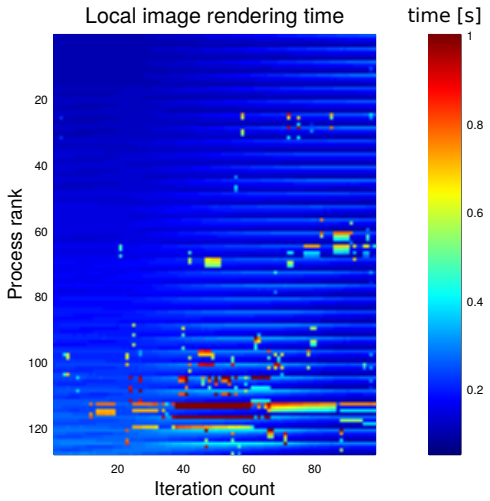

Figure 4: Distribution of image render time across the 100 iterations of the Helsim simulation, Lynx cluster $P = 128$ with 8 processes per node. In the image, the upper range values are clipped at 1 s. However only 0.16% of the values were clipped, with the maximum observed value being 1.6 s

From a design standpoint, it would be best to incorporate the schedule construction within the reduction operation. In this way, the whole process would be entirely transparent to the user and delegated to the library runtime system. An environment variable could be used to toggle the usage of the PAT imbalance features in the library implementation.

5. Reduction algorithms

In this section, we discuss the time complexity of reduction algorithms for balanced PATs and use a simple linear model to produce predictions of runtimes for the five surveyed algorithms. We then define the Clairvoyant algorithm that is the main contribution of the paper, followed by four other reduction algorithms that we selected for performance comparison. In the following text, we use n as shorthand for $\log_2 P$, where P is the number of processes participating in a collective operation.

5.1. Complexity model

To model the time complexity of reduction algorithms, we will use a simple linear communication cost model consisting of three parameters: α the latency in message transmission, β the per byte cost of message transmission, and γ the per-byte cost of message combination [32]. The time to send and combine a message of size m, from one process to the other, can be expressed in this model as: $\alpha + m\beta + m\gamma$. The three parameters in this model are assumed to be message size and process count independent.

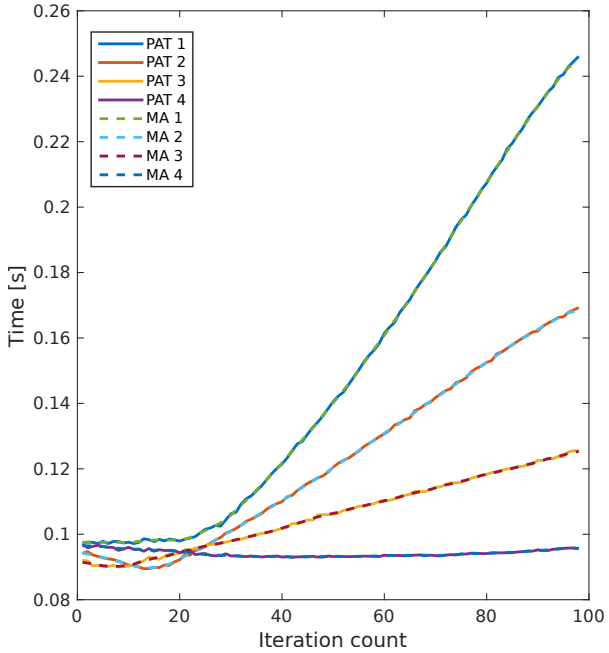

Figure 5: Examples of the 4 principal clusters of image render times across the 100 iterations of the Helsim simulation, valid for process ranks $p < 24$. The displayed PAT sequence plots are those for ranks 0-4. Superimposed on each pattern is the moving-average fit computed with a window of size 5. The data was originally gathered on the Lynx cluster with $P = 128$ and 8 processes per node (ppn=8). The x-axis denotes iteration count, while the y-axis the image render time in seconds

Table 1: Lower bounds for collective operations

Collective	Latency	Bandwidth	Computation
Reduce	$n\alpha$	βm	$\frac{P-1}{P}\gamma m$
Reduce-Scatter	$n\alpha$	$\frac{P-1}{P}\beta m$	$\frac{P-1}{P}\gamma m$
Gather	$n\alpha$	$\frac{P-1}{P}\beta m$	N/A

5.2. Lower bounds on time complexity

It is illustrative to establish the lower bounds on the cost of the reduction operation and the two collective operations used as building blocks for composite algorithms: reduce-scatter and gather.

Latency: In reduction, every process must contribute its data by sending at least one message. These messages have to be successively combined until a fully combined data vector resides at the root process. In a single port network model, at most two messages originating from different processes can be combined at the same time. This leads to a minimum of n steps, each of which costs time α. Similar reasoning can be applied for the reduce-scatter and gather operations.

Bandwidth: The lower bound for bandwidth is derived by observing that the root node must at the very least receive a quantity of data amounting to problem size m, wherein all the information from the other $P-1$ processes has been combined. For the reduce-scatter and gather operations, the root node must receive or send $P-1$ segments of size $\frac{1}{P}m$.

Computation: the lower bound on the computation can be derived by the observation that if all the computations were to be performed on a single node, they would take time $(P-1)m\gamma$. Assuming perfect load balancing, the computation time can be brought down by distribution to $\frac{P-1}{P}m\gamma$. The lower bounds are summarized in Table 1.

It is important to observe that it is not possible for any single algorithm to meet all three lower bounds. For example, to meet the lower bound on the computation requires that the computation be perfectly load balanced. This can be achieved through a reduce-scatter operation in the first phase where each process is responsible for $1/P$ of the data. In the second phase, the results of the reduce-scatter operation can be collected at the root process through a gather operation. In the linear cost model, the first phase would have the time complexity of $n\alpha + \frac{P-1}{P}m\beta + \frac{P-1}{P}m\gamma$. The second phase would have the time complexity of $n\alpha + \frac{P-1}{P}m\beta$. All together, this would roughly constitute a 2-approximation in latency and bandwidth to the established lower bounds in Table 1.

If we view the execution of an algorithm in terms of rounds, where in each round it can send, receive and combine one segment of size $B = \frac{m}{N}$, where N is the number of segments a message has been divided into, then the minimum number of rounds to complete the reduction (for balanced PATs) is $n+N-1$. In a single port network (Definition 3.1), n rounds are necessary for the first fully reduced segment to reside at the root, followed by additional $N-1$ rounds for the remaining segments.

Using the linear cost model, we can derive the time complexity of this algorithm as follows. Assume that the per-process input data of size m is split into N segments of size $B = \frac{m}{N}$.

Then the reduction time of an equi segmenting algorithm $\mathcal{A}$ for balanced PAT, that completes in R rounds is $T_{\mathcal{A}}(\pi) = R \cdot (\alpha + B(\beta + \gamma)$. Expanding R, gives us the following equation:

$$T_{\mathcal{O}}(\pi) = (n-1)\alpha + (n-1)B(\beta+\gamma) + N\alpha + m(\beta+\gamma) \tag{1}$$

Differentiating by $\frac{d}{dN}$ and selecting the positive real root, we can determine the optimal number of segments:

$$N_{\text{opt}} = \sqrt{\frac{(n-1)m(\beta+\gamma)}{\alpha}}$$

and the optimal segment size:

$$B_{\text{opt}} = \frac{m}{N_{\text{opt}}} = \sqrt{\frac{\alpha m}{(n-1)(\beta+\gamma)}}$$

By substituting B_{opt} for B and N_{opt} for N in Eq. 1, we derive the runtime complexity of the optimal equi segmenting reduction algorithm $\mathcal{O}$ for balanced PATs:

$$T_{\mathcal{O}}(\pi) = (n-1)\alpha + 2\sqrt{(n-1)\alpha}\sqrt{m(\beta+\gamma)} + m(\beta+\gamma) \tag{2}$$

5.3. Absorption potential of a clairvoyant reduction algorithm

As previously discussed, no reduction algorithm can meet all three lower bounds: latency, bandwidth and computation. Thus the algorithm $\mathcal{O}$ that solves the two-way reduction problem in optimal time will employ one of the following two strategies: either the workload is evenly divided between the two processes by first performing a reduce-scatter operation followed by a gather operation for the total time complexity $t_1 = 2\alpha + m\beta + 1/2m\gamma$ or the latency is minimized by having one process send all its data to the root in a single message, with the root performing all the computation, for time $t_2 = \alpha + m\beta + m\gamma$. The first strategy will be better whenever $\alpha < 1/2m\gamma$. Computing the maximum absorption according to Proposition 3.7 we can observe a switch from the second to first strategy near the point $m = 10\,\text{KiB}$ (Fig. 6). The exact point will depend on linear model parameters of each particular machine. This result, combined with the fact that we evaluated algorithm performance for $m \geq 128\,\text{KiB}$ has motivated our decision to opt for the first strategy of distributing the computational workload in the design of algorithm Clairvoyant.

In fact, there is another strategy for implementing algorithm $\mathcal{O}$: utilize the remaining $P-2$ processes to decrease the per-process workload. To do that, a scatter from each of the 2 processes to other $P-1$ processes would be required, followed by local computation on data of size $\frac{m}{P}$ concluded by a collective gather of computed $P-1$ segments of size $\frac{m}{P}$ to the root. The total time of this operation is:

$$2\alpha \log_2 P + 2\frac{P-1}{P}m\beta + \frac{1}{P}m\gamma$$

This strategy is hampered by extra latency of roughly $\log_2 P$ and extra data transmission time of $m\beta$. For very large messages and moderately large systems, we can ignore the latency term leading to the conclusion that network bandwidth would have to be double that of computation speed. This is, however, contrary to current trends in high performance systems where computation speed is upwards of three times that of effective bandwidth.

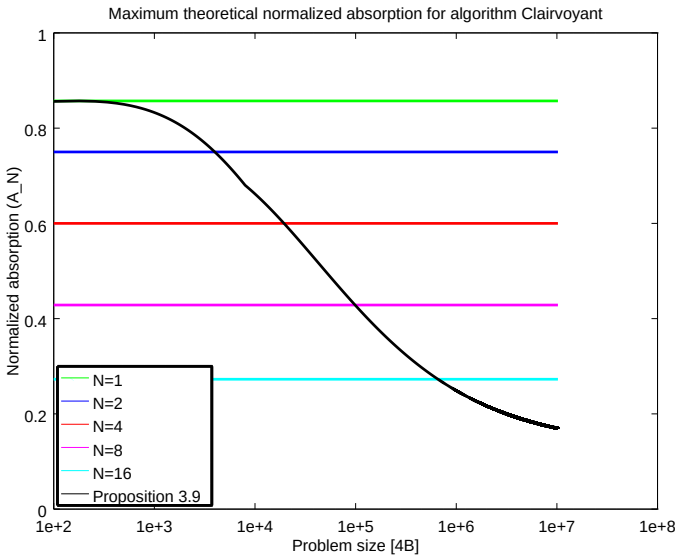

Figure 6: Theoretical prediction of maximum normalized absorption of algorithm Clairvoyant as function of problem size (multiples of 4 bytes) and number of segments (N). Normalized absorption is computed according to the equation: $A_N = \frac{t_{\mathfrak{C}}(\pi,128)-t_{\mathfrak{C}}(\pi,2)}{t_{\mathfrak{C}}(\pi,128)}$, where $t_{\mathfrak{C}}(\pi,P)$ is the runtime of algorithm Clairvoyant for balanced PATs as defined in Table 3. The black curve was computed according to Proposition 3.7. Parameters $\alpha = 2.66\,\mu\text{s}$, $\beta = 4.8179 \times 10^{-10}\,\text{sB}^{-1}$, $\gamma = 1.6654 \times 10^{-10}\,\text{sB}^{-1}$; experimentally determined on the PRACE CURIE supercomputer

5.4. Clairvoyant schedule generation

We define the Clairvoyant schedule generation algorithm as Algorithm 1. The algorithm generates a reduction schedule which is personalized for each of the P processes. The algorithm operates in rounds, within which a process can at most send one segment, receive one segment and combine one segment (Definition 3.1). As its input parameters, it receives the number of segments N into which the input data is to be split, the time $d = \alpha + B(\beta + \gamma)$ to complete one round of reduction (the time to receive and combine one segment) and the PAT vector a that represents the predicted PATs at the current reduction operation invocation point.

The algorithm proceeds in rounds, where in each round processes send/receive and combine at most one segment of the input data. At the beginning of each round, the algorithm establishes which processes are ready to participate in the reduction. This is done by first selecting the process Q_0 with the minimum arrival time, i.e. the top element of the priority queue Q, where priority is assigned to process ranks i with smaller (earlier) arrival time a_i (Line 3).

Initially, the set Q is comprised of all P processes. Then a group G of ready process is constructed from the queue Q, so that for $\forall i \in G, a_i \leq a_{Q_0} + d$ (Line 5). Thus, all members of G are formed by those processes whose arrival time is less than or equal to the arrival times of process Q_0 plus the time to complete one round of reduction (d). A state matrix M of size $P \cdot N$ is used to keep track of states for each segment on every process (Line 1). The possible states are A, for available, E for empty (a segment that has been sent) or P for partially combined segment (a segment that contains information from at least one other segment). The algorithm then proceeds by adhering to a greedy principle: each process attempts to receive and combine a segment that still resides in its local buffer (state A or P) by giving priority to segments with lower indices (Line 21 and 25). Care is taken to ensure that a process sends a segment only once within each round. This is done by keeping record in the vector S (Line 2). In each round, a sink process (r^*) is established: if process r, the root selected in the collective operation call, is not part of G, then process Q_0 with minimum arrival time is selected as the sink. Otherwise, process r is selected (Lines 14-18). Implicit to the algorithm is the principle, that once a process sends a segment of data, it will no longer receive segments of that index - unless that process is the sink process. This ensures that all segments eventually trickle down to the sink process. The sink process follows a slightly different greedy principle: it attempts to receive a segment regardless of whether a segment of matching index resides in its local buffer, with priority assigned to segments with lower indices (Line 23 and 25). This ensures that even if the root process r is the last to arrive, the reduction can proceed uninterrupted as long as there are segments to be received and reduced.

At the end of a round, if a process has sent all of its segments, then its schedule is complete and it is not put back into the set Q (Line 36). The algorithm repeats until all processes except the root process r have been removed from set Q.

The schedule constructed by the algorithm for $P = 4 \wedge N = 4$, when the PAT is balanced, is shown in Table 2. The execution of this schedule is illustrated in Fig. 7. We will briefly trace the schedule generation algorithm for the first two processes in Round 1. Because the PAT is balanced, $G = P$ and $Q_0 = r$. Beginning with process rank

$i = r = 0$ (the for loop on Line 12), the algorithm sets the index of interest $j = 0$ (Line 13). The linear search on Line 21 determines that the index $j = 0$ is indeed eligible ($(M(i, j) \in \{A, P, E\}$) - at algorithm start, all elements of the matrix are set to the value `A` (Line 1). Then, the algorithm searches for the first process z (Line 19) among the processes in group G whose segment of that index has not yet been sent (Line 25). If such a segment has been found on process z, the algorithm checks that process z has not already sent a segment in the current round (Line 25). In this case, $z = 1$ and the algorithm proceeds to Line 30. The inbound queue of process 0 (`I[0]`) is enqueued with the pair $(z, j) = (1, 0)$. At line 31 the outgoing queue of process 1 (`O[1]`) is enqueued with the pair $(i, j) = (0, 0)$ (Compare with Table 2). Finally, the state matrix M is updated accordingly (Lines 32-33).

The algorithm loops back, and the next process $i = 1$ in the group G is selected. The search on Line 21 determines that the first eligible segment is of index $j = 1$. The linear search on Line 25 determines that the first process that can send the segment of index $j = 1$ is process rank $z = 0$. The algorithm proceeds to line 30, and enqueues the pair $(0, 1)$ to the inbound queue of process rank 1 (`I[1]`) (Line 30) and the pair $(1, 1)$ to the outbound queue of process rank 0 (`I[0]`) (Table 2). This concludes the first round trace for the first 2 processes.

The execution of the generated schedule for the imbalanced PAT $\psi = (0, 0, 0, \delta), P = 4 \wedge N = 4$ is illustrated in Fig 8. Here, we can observe that the algorithm has generated such a schedule that allows for the entire *3-way reduction problem* to be solved among processes $\{0, 1, 2\}$ by the time process rank 3 arrives at the collective call site.

For $N = 1$, the schedule generated by this algorithm degenerates into a binomial tree. We experimentally evaluated the schedule generation algorithm for balanced PATs and all permutations of $P = \{4, 8, 16, 32, 64, 128, 256, 512\}$ and $N = \{4, 8, 16, 32, 64, 128, 256, 512\}$. All the generated schedule lengths were of length $R = n + N - 1$, thus matching the optimal equi segmentation schedule length. At this time, we do not have a proof that the schedule generation algorithm produces a schedule of length R for all input parameters. Due to out-of-order combination of data segments, this algorithm can only be used with commutative operations. This is however unavoidable for any algorithm that endeavours to take best advantage of available slack caused by imbalanced PATs.

5.4.1. Implementation details

We implemented the algorithm in C++ 11. In the implementation of the algorithm, the linear search in lines 21 and 23 is optimized by placing indices to non-empty segments in a flat set, built over `std::vector`. This was made in favour of asymptotically better ordered list where the cost of item removal is $O(1)$ vs $O(N)$ for flat set. Since the number of segments N is comparatively small, and each element is an integer, a contiguous data structure such as flat set is more cache friendly. Furthermore, the linear search on line 25 is optimized by keeping a priority queue for each segment. The queue is implemented with a pairing heap data structure [33]. The two operations, `heap.push()` and `heap.erase()`, performed by the algorithm both have amortized complexity of $O(2^{2\sqrt{\log \log N}})$ [34]. Profiling the algorithm execution indicates that 41% of the runtime is spent in `heap.erase()`. We suspect that much of this cost is due to expensive memory deallocation performed by the implementation of `boost::heap::pairing_heap`. Modifying the algorithm to use

Algorithm 1 Clairvoyant non-atomic schedule generation algorithm

Input:

P : `integer`, the communicator size
N : `integer`, the number of segments
a : `vector of double`, the PAT vector of size P
d : `double`, the time to complete one round of reduction
r : `integer`, the root rank

Output:

I : `queue of pair (rank,index)` //Inbound schedule queue
O : `queue of pair (rank,index)` //Outbound schedule queue

Let M be a state matrix of size $P \cdot N$. Set $M(:) = \mathtt{A}$, i.e. mark all segments as available.
Let S : `bool` be an array of size P
For $\forall i \in \{0...P-1\}$ insert process rank i into a priority queue Q, where priority is given to process ranks i with smaller arrival time a_i
while size$(Q) > 1$ **do**
  Pop processes $Q_0...Q_k$ from Q to form a sorted vector G, so that $\forall i \in G, a_i \leq a_{Q_0} + d$.
  for $\forall i \in G$ **do**
    let $S(i) = \perp$ // *No ready process in G has yet sent a segment*
  end for
  if $r \in G$ **then**
    insert r at first position in G
  end if
  for $\forall i \in G$ **do**
    let $j = 0$, let $z = \emptyset$
    if $r \in G$ **then**
      let $r^* = r$ // *sink is the root*
    else
      let $r^* = Q_0$ // *sink is the earliest to arrive process*
    end if
    while $z \in \emptyset$ **do**
      if $i \neq r^*$ **then**
        starting from j, find first segment x, such that $M(i,x) \in \{A,P\}$. Let $j = x$.
      else
        starting from j, find first segment x, such that $M(i,x) \in \{A,P,E\}$. Let $j = x$.
      end if
      perform linear search through group G to find the first process $z \neq i$ such that $M(z,j) \in \{A,P\} \wedge S(z) = \perp$
      if $z \in \emptyset$ **then**
        let $j = j + 1$
      end if
    end while
    enqueue to $I(i)$ the pair (z, j)
    enqueue to $O(z)$ the pair (i, j) and let $S(z) = \top$
    $M(z, j) = E$
    $M(j, z) = P$
  end for
  for $\forall i \in G$ **do**
    if $\exists j \in \{0...N-1\} : M(i, j) \neq E$ **then**
      $a_i = a_i + d$
      push process i into Q
    end if
  end for
end while

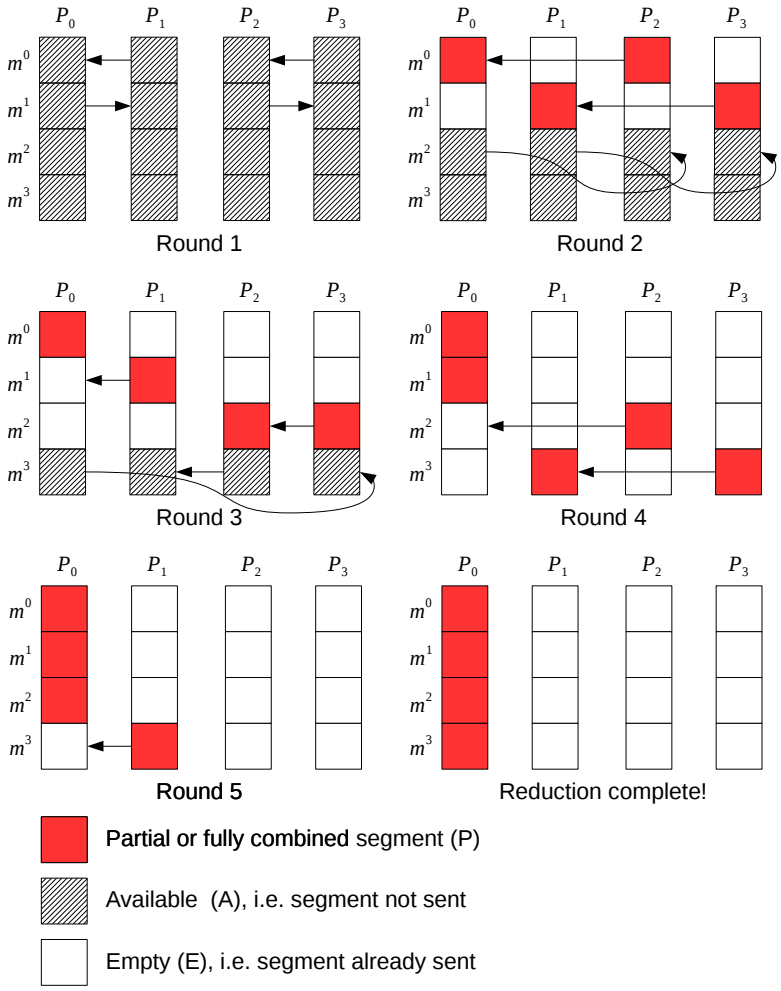

Figure 7: Execution of the schedule generated by Algorithm 1 for the balanced PAT $a = (0,0,0,0)$, communicator size $P = 4$ and the number of segments $N = 4$. The complete schedule is presented in Table 2

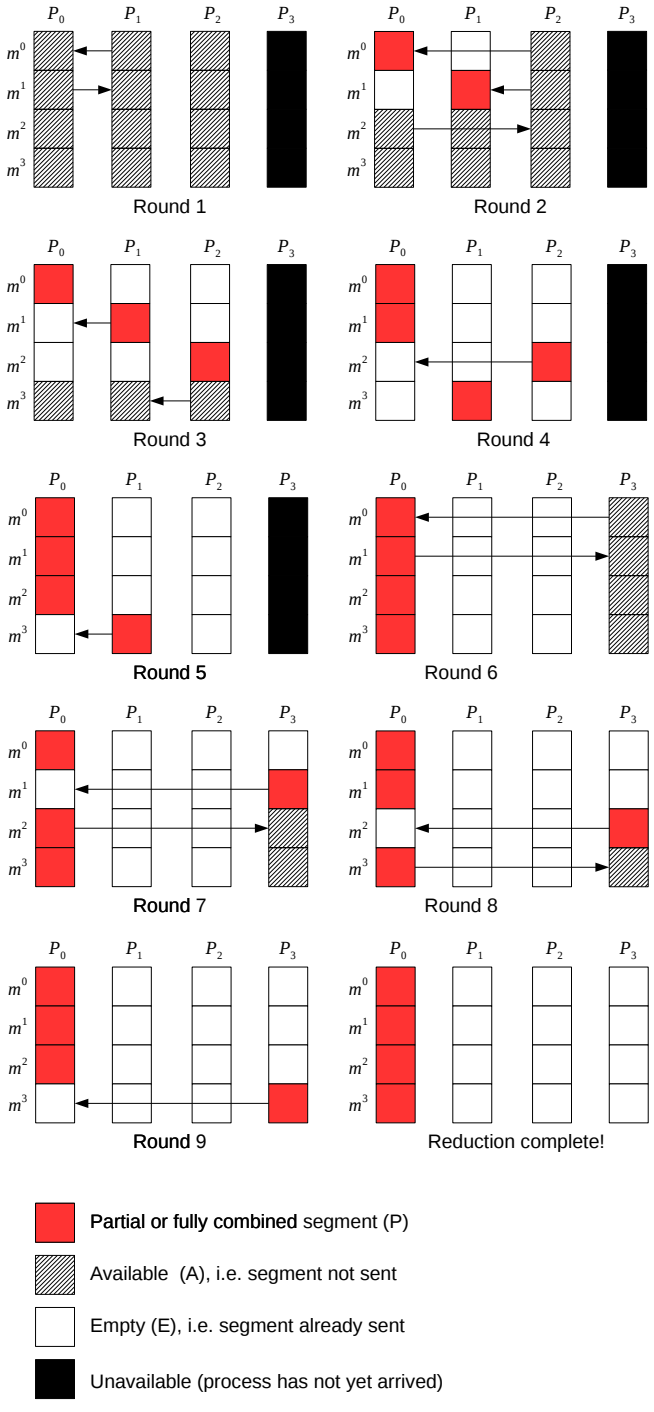

Figure 8: Execution of the schedule generated by Algorithm 1 for an imbalanced PAT. In this example, $P = N = 4$ and the PAT $a = (0,0,0,\delta)$, where the delay $\delta = t_{\mathcal{C}}(\pi,3)$, i.e. the time required for algorithm Clairvoyant to solve the 3-way reduction problem. The generated schedule length is $R = t^*_{\mathcal{C}}(\pi,3) + t^*_{\mathcal{C}}(\pi,2) = 5+4 = 9$ rounds, where $t^*_{\mathcal{C}}(\pi,P)$ is the schedule length algorithm Clairvoyant generates for the P-way reduction problem with balanced PATs

Table 2: Schedule generated by algorithm Clairvoyant (Algorithm 1) for the balanced PAT $a = (0,0,0,0)$, communicator size $P = 4$ and number of segments $N = 4$. The schedules consists of two queues `I` and `O` (inbound&outbound), where the queue elements are integer pairs `(rank,index)`, where `rank` denotes the communication peer rank and the `index` denotes the ordinal number of the segment to be communicated. For these input parameters, the generated schedule length is $R = n+N-1 = 5$ rounds.

Rank	Queue	R_1	R_2	R_3	R_4	R_5
0	I	(1,0)	(2,0)	(1,1)	(2,2)	(1,3)
	O	(1,1)	(2,2)	(3,3)	$\perp$	$\perp$
1	I	(0,1)	(3,1)	(2,3)	(3,3)	$\perp$
	O	(0,0)	(2,2)	(0,1)	$\perp$	(0,3)
2	I	(3,0)	(0,2)	(3,2)	$\perp$	$\perp$
	O	(3,1)	(0,0)	(1,3)	(0,2)	$\perp$
3	I	(2,1)	(1,2)	(0,3)	$\perp$	$\perp$
	O	(2,0)	(1,1)	(2,2)	(1,3)	$\perp$

`heap.decrease()` followed by `heap.increase()`, instead of `heap.erase()` might lead to further performance improvement.

5.4.2. Time and space complexity

To give a rough estimate of schedule generation time complexity, we will examine the case of balanced PATs. Then the algorithm will take R rounds where $R = n+N-1$. In each round, a loop of at most P iterations (Line 12) is executed. In each iteration, we can assume that on average a single lookup of the first element of both the flat set and pairing heap will be required ($O(1)$ time), followed by 2 pop (or one pop and one erase) and push operations on the heap, each of which is roughly O($\log P$) in complexity, plus a potentially $O(N)$ operation to remove an element from the flat set. This leads to a total of:

$$O(P(n+N-1)(N+3n)) \approx O(P \cdot N^2 + P\log^2 P)$$

However, for small N such that the flat set resides within cache memory, we can expect the element removal operation for flat set to be of near constant cost, as all the data can be rotated with a single contiguous memory operation. In that case, we can expect the algorithm to scale with the complexity:

$$O(P \cdot N + P\log^2 P)$$

Empirical data suggests (Fig. 9) that the runtime of the schedule generation algorithms grows roughly linear in both P and N. As we will see later, this has strong implications on the usage scenarios of this algorithm.

In the space domain, the algorithm requires a matrix of $P \cdot N$ integers denoting the status {`A` (available), `P` (partially combined) , `E` (empty)} of individual segments. In addition to this, for each segment, a priority queue of maximum size P is maintained, wherein each element consists of a single integer denoting process ranks. Greater priority is handed to smaller ranks. Moreover, a matrix of handles to priority queue elements of size $P \cdot N$ is maintained to perform `heap.erase()` if a match has been found in

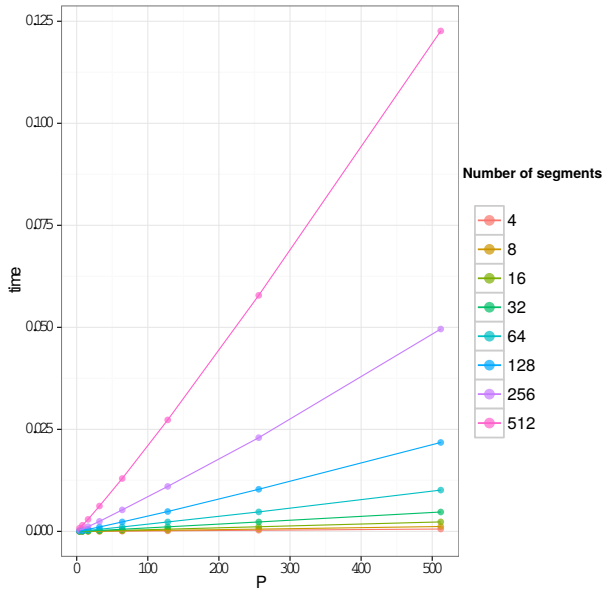

Figure 9: Schedule generation runtime as function of number of processes (P) and number of segments (N). Reported runtimes are the mean of 1000 observations per each pair of input parameters (P,N), denoted in seconds

the search on Line 25. As its return value, the algorithm generates two queues per process: queue `I` and queue `O`, the incoming and outgoing queue, respectively. The maximum length of the queues is determined by the total number of rounds R to complete the reduction, where $R = n + N - 1$. Each element of the queue is a pair of two integers: the rank of the communication peer and the ordinal number of the segment to communicate.

5.4.3. Clairvoyant schedule execution

Algorithm 1 generates a per process personalized schedule consisting of two queues: the inbound and outbound communication queue. Each element i of the queue defines the inbound and outbound communication peer in round i, as a pair of two integers (rank,index). The former represents the rank of the peer process and the latter the index of the segment that is to be communicated. The schedule execution algorithm then linearly iterates through the schedule, issuing up to one `MPI_IRecv` and `MPI_ISend` call per-process in each round and combining at most one segment of size m/N elements. Before a process proceeds to the next round, all its outstanding MPI calls are completed by a call to `MPI_Wait`. This means that there is no explicit synchronization within the subgroup G of ready process in round i. This decision was influenced by the dynamic nature of the subgroup and the non-trivial creation and maintenance cost.
For simplicity, we will hitherto refer to this algorithm as Clairvoyant.

5.5. Selected algorithms

We now proceed to define and discuss the four selected adversary algorithms. It is assumed throughout this section that the root resides at process rank 0, and that the communicator size is a power-of-two. In the presented pseudo code, `BLOCK` is a procedure of two parameters (iterator pointing to the beginning of a segment and segment size) that produces segments of input data.

5.5.1. Binomial Tree

This is the optimal reduction algorithm for atomic input data with balanced PATs and is used in implementations of many MPI libraries. The definition of the algorithm is provided in Appendix A.

5.5.2. Parallel Ring

This is an algorithm best suited for large messages as discussed in [15] where the authors name it the bucket or cyclic algorithm. Our implementation is based on the algorithm explicated in that paper. The definition of the algorithm is provided in Appendix A. The same algorithm forms the basis of the bandwidth optimal all-reduce algorithm discussed by [13]. The authors show, under the assumption that MPI processes with consecutive ranks are assigned to processors (cores) in each SMP node, the logical ring communication pattern of this algorithm is contention free (assuming full-duplex links on single ported nodes). This property makes the algorithm suitable for execution on fully subscribed SMP clusters.

5.5.3. Butterfly

Many MPI implementations employ some version of the butterfly algorithm for reduction of large messages. The version of MVAPICH2 used for our experiments implemented the Rabenseifner's version of the butterfly algorithm [11, 20]. In the domain of computer graphics, this algorithm is also known as binary swap used for sort-last image compositing [35, 36]. For the purposes of this paper, we have implemented the algorithm explicated in [15] as bidirectional exchange reduce-scatter followed by a call to library implemented `MPI_Gather`. Our implementation is written in iterative form, while that of [15] was written in recursive. The definition of the algorithm is provided in Appendix A. This implementation necessitates that P be a power-of-two. There exist optimizations of this algorithm for non-power-of-two process counts using a binary blocks [12, 20] scheme to reduce some of the load imbalance.

5.5.4. Radix-k

In the domain of image compositing, a new algorithm has recently emerged, by the name of Radix-k, first described in [16] and later improved in [17]. Reduction operations in this domain are characterized by very large problem sizes ($> 4\,\mathrm{MiB}$) and non-commutative combination operators, disqualifying algorithms such as Parallel Ring that operate only on commutative operators. The definition of the algorithm is provided in Appendix A. This algorithm operates by grouping P processes into r groups. These r groups form the radix vector $k = [k_1, \ldots, k_r]$ with the property that

$P = \prod_{i=1}^{r} k_i$. The algorithm then proceeds in r rounds, where in each round i it performs k_i exchanges and reductions among $\frac{P}{k_i}$ groups. In each round, the current slice is subdivided in k_i pieces, so that the size of the slice in round i is $\frac{l}{\prod_{j=1}^{i} \frac{1}{k_i}}$. Groups are formed in the following way: in round 1, the k_1 members of a group are nearest neighbours in rank order; in the next round, each member is now k_1 apart, in the third round $k_1 \cdot k_2$, etc. An illustration of the algorithm execution is given in Appendix C. Radix-k has the potential to fine-tune the amount of communication concurrency (multi-portness) to almost any given architecture by the appropriate selection of the k-values. When radix vector $k = [2, 2, ..., 2]$, then this algorithm becomes equivalent to Butterfly.

In the experimental evaluation of the algorithm's comparative performance, we have determined the radix vectors empirically, by selecting for each problem size m, the radix vector that resulted in best performance. The empirically determined vector was identical for all problem sizes and equalled $k = \{4, 4, 8\}$.

5.6. Time complexity of selected algorithms

Table 3 presents the computed time complexity of some well-known reduction algorithms, including those implemented in this study. The equations in Table 3 were derived using the linear cost model. It is illustrative to point out that among the listed algorithms, Butterfly will outperform Clairvoyant for some problem sizes m. In fact, for the data set presented in Fig. 10, Butterfly achieves a minimum of 0.96 relative runtime compared to Clairvoyant. However, the range of problem sizes m for which Butterfly is better or equal to Clairvoyant (Fig. 10) is $[1.7460 \times 10^4\,4\text{B}, 8.8820 \times 10^4\,4\text{B}]$. This represents only 0.697% of the set of problem sizes in Fig. 10. In general, the size of the range R_B where Butterfly outperforms Clairvoyant will depend on the ratio $r = \frac{\beta}{\gamma}$ and process counts P. The size of the range R_B is inversely proportional to both r and P.

5.7. Rationale of algorithm selection

This work does not attempt to provide a comprehensive survey of reduction algorithms known in the literature. In particular, two high-performance tree pipeline algorithms, Linear Pipeline and 2-Tree, were not included. While the experimental results of these two algorithms might be interesting, they are both equi-segmenting algorithms like Clairvoyant and in the absence of imbalance their expected performance is strictly worse than that of Clairvoyant (Table 3). On the other hand, algorithms Butterfly and Radix-k operate with heterogeneous segment sizes and can in theory outperform algorithm Clairvoyant. For that reason, both of the algorithms are included in this study.

We chose not to report the runtime of the native `MPI_Reduce` implementation, for two reasons. First, it was not known to us what algorithm was used to implement the operation. Second, the observed performance of the native implementation fell short of all non-atomic reduction algorithms. This would imply that either the implementation algorithm is far from optimal, or that the native implementation was poorly tuned for this particular problem size. Table 4 shows the results of our initial performance assessment study conducted in December 2014 upon which we have based our algorithm

Table 3: Time complexity of some reduction algorithms for homogeneous, fully connected full-duplex networks.

Algorithm	Communication cost (upper bound)	non-comm ops	Source
Binomial Tree	$n[\alpha+\beta m+\gamma m]$	yes	[10, 9]
Butterfly	$2\alpha n+2\frac{(P-1)}{P}m\beta+\frac{P-1}{P}m\gamma$	yes	[15]
Parallel Ring	$(P-1+n)\alpha+\frac{(P-1)}{P}m(2\beta+\gamma)$	no	[13]
Radix-k	$(\sum_{i=1}^{r}[(k_i-1)]+n)\alpha+\frac{(P-1)}{P}m(2\beta+\gamma)$	yes	[17]
Linear pipeline	$(P-2)\alpha+2\sqrt{(P-2)\alpha}\sqrt{m(\beta+\gamma)}+\beta m+\gamma m$	yes	[18]
Two-tree	$4(n-1)\alpha+4\sqrt{(n-1)\alpha}\sqrt{\beta m/2}+m\beta+2m\gamma$	yes	[19]
Clairvoyant	$(n-1)\alpha+2\sqrt{(n-1)\alpha}\sqrt{m(\beta+\gamma)}+m\beta+m\gamma$	no	This paper

Communication cost is calculated as the time required for the last process to complete execution in the worst case, for balanced PATs. Problem size is denoted by m, the number of process by P and $n=\lceil\log_2 P\rceil$. The formula for the Butterfly algorithm is valid when the communicator size P is a power-of-two.

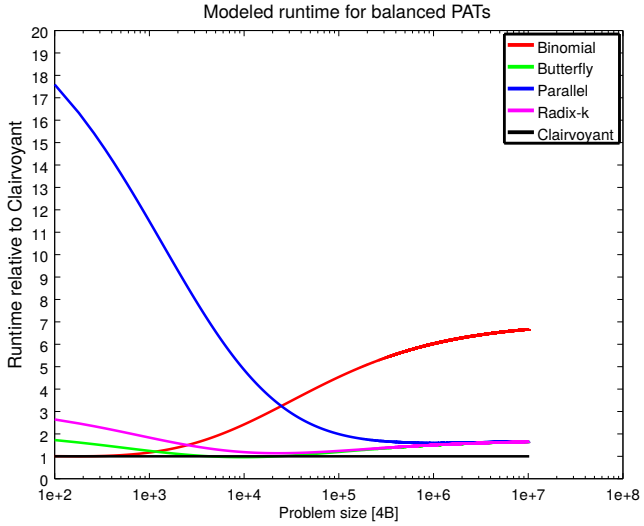

Figure 10: Performance prediction based on the time complexity equations presented in Table 3. The x-axis denotes the problem size in multiples of four bytes, while the y-axis denotes the algorithm runtime relative to algorithm Clairvoyant. Parameters $\alpha=2.66\,\mu s$, $\beta=4.8179\times10^{-10}\,sB^{-1}$, $\gamma=1.6654\times10^{-10}\,sB^{-1}$; experimentally determined on PRACE CURIE supercomputer. For Radix-k, the same radix vector $\{4,4,8\}$ was used for the entirety of the problem size range

Table 4: Results of the initial performance assessment study conducted in December 2014 on the PRACE CURIE supercomputer. Problem size $m = 4\,\text{MiB}$ and communicator size $P = 128$.

Algorithm	4 MiB
Binomial	0.034 s
Butterfly	0.0089 s
Native	0.011 s
Local Redirect	0.041 s
Parallel Ring	0.0083 s
Linear Pipeline	0.0061 s
Radix-k	0.0085 s

selection decision. Included among these is algorithm Local Redirect [28], the only other imbalance robust reduction algorithm. However, Local Redirect was designed for atomic input data only, and this is clearly reflected in its performance: almost an order of magnitude slower than the fastest algorithm in Table 4. Due to its lack of competitiveness, when pitted against non-atomic reduction algorithms, algorithm Local Redirect was excluded from this study.

6. Experimental methodology

In this section, we report the experimental results obtained from executing the selected algorithms on the Partnership for Advanced Computing in Europe (PRACE) CURIE supercomputer. This is a BULL x86 system with 5040 blades each equipped with 2 Intel Xeon E5-2680 8 core processors running at 2.7GHz and 64 GB of RAM. Machine nodes are interconnected using Infiniband QDR technology. All algorithms and experiments were implemented in C++11 using MPI p2p primitives. The code was compiled with gcc 4.6.3 and linked to a proprietary BullxMPI version 1.2.8.2 based on OpenMPI, provided with the PRACE CURIE machine. The number of nodes was maintained at $P = 128$ throughout all experiments and one process was assigned per node (ppn $= 1$).

The protoyping and testing of all the algorithms was performed on the Lynx cluster at the Intel ExaScale Lab in Leuven, housed at IMEC. The data displayed in Fig. 4 was produced on this cluster. This is a 32 node system, where each node is a DL170e G6 blade: dual socket, six core Intel Xeon X5660@2.8 GHz, 96 GB of memory and 500 GB of disk space. Each node comes with a Mellanox Technologies MT26428 Infiniband card (ConnectX VPI PCIe 2.0 5GT/s - IB QDR/10GigE). The nodes are interconnected using a single Voltaire 36P QDR switch. This is a crossbar switch, so this network should achieve full bisection bandwidth for all communication patterns. All nodes ran Ubuntu 12.04.3 LTS Precise.

6.1. Experimental Method

To evaluate the runtime performance of collective operations under load imbalance, we implemented a benchmarking methodology that is in design most similar to that of the Intel MPI benchmarks. To ensure the reproducibility of our results, we followed the guidelines laid out in [37] and benchmarked the collective operations for a range

of message sizes and a large number of iterations. We measured the runtime of each process i, i.e. $t_{\mathcal{A}}(i) = e_i - \min_{0 \le i < P} a_i$. For elapsed time estimation, we used the MPI timing routine `MPI_Wtime`.

6.2. Microbenchmark rationale

On the PRACE CURIE supercomputer, this clock (`MPI_Wtime`) was not global and the reported precision was 1 µs. Each time, before timing the performance of a collective operation, we synchronized all processes by explicit calls to two consecutive `MPI_Barrier` routines, as is done in Intel MPI benchmark version 4.0[38]. This ensures that runtimes of collectives are measured in isolation and that the arrival times are sufficiently balanced: assuming a binomial tree broadcast algorithm (as part of barrier implementation), for the evaluated number of processes $P = 128$, this approach leads to approximately 18 µs of spread in arrival times. We compute this spread from the measured latency of a control message L=2.66 µs, as reported by Netgauge [39]. The minimal injected absolute imbalance was 50 µs, and the minimal imbalance increments also 50 µs. For the smallest evaluated message size $m = 128$ KiB, where the minimum collective runtime was ≈ 200 µs, we considered this approach to be sufficiently accurate. If the broadcast operation is implemented with hardware multicast, a feature common to Infiniband systems, then even larger communicators or smaller message sizes could be evaluated. The entire runtime estimation procedure is described by Algorithm 2.

Instead of a double barrier synchronization mechanism, the benchmark also supports a window-based approach, such as the one in [40, 39]. However, due to very long tails in collective operation runtime distributions, when subjected to PAT imbalance, we found it difficult to fix the window size, such that we can guarantee that all processes will have completed the collective operation call before the new window onset. The solution was to either make time windows very long or to re-synchronize the processes after each iteration and re-establish the window. Both solutions were found to be relatively slow and due to limited computational resources at our disposal, we were forced to adopt the faster, if less precise, barrier-based benchmarking approach. Yet, for completeness, we allow our microbenchmark to be configured to either synchronize the processes with double barrier calls, or with the global clock window-based scheme, by setting a compile time flag in the configuration file.

In the execution of the microbenchmark, the PAT patterns are computed at the root and broadcast to all processes in the communicator, prior to collective operation runtime measurement. To produce imbalanced PATs, the microbenchmark first synchronizes all processes in the communicator, either through a double barrier scheme or a window-based approach. Then the imbalance is generated through calls to high-resolution *sleep* calls, provided by the C++11 standard library, or busy looping for time a_i. In our experiment, we used busy looping to keep process i suspended for time a_i. Finally, the process proceeds to invoke the collective operation.

In this manner, the PATs vectors passed as input are a perfect prediction, and all the reported results for algorithm Clairvoyant will represent a best case scenario. However, in Table 5 we present a preliminary comparison, of algorithm Clairvoyant's performance for PATs prediction using the SMA model (Fig. 5) against the perfectly

Algorithm 2 Runtime estimation procedure for a problem size m, constant arrival pattern $\psi = (a_0, ...a_{P-1})$, a list of algorithms $\mathcal{A}^v = \{\mathcal{A}_0, ..., \mathcal{A}_{n-1}\}$ and a list of operators $\star^v = \{\star_0, ...\star_{f-1}\}$.

Input data of size m at each process in the communicator
One output file for each combining operator, with $numIter - 1$ recorded observations per algorithm

L0 For each algorithm $\mathcal{A}_i \in \mathcal{A}^v$ do

L1 For each combining operator $\star_j \in \star^v$ do

S1 Perform a double barrier

S2 Busy loop for time a_i

S3 Start the timer (time t_s)

S3.1 Execute the collective operation

S3.2 Stop the timer (time t_f)

S4 Report the elapsed time $t_i = t_f - t_s$

S5 Collect the reported times at the root

S5.1 (Root) Store the time $\max(a_i + t_i) - min(a)$

S6 If the number of iterations $k <$ numIter goto L0

Table 5: SMA model predicted PATs vs. true PATs clairvoyancy. Preliminary data collected on the VSC muk cluster; $P = 64$, one process per node. Problem size $m = 16$ MiB. Reported values are the mean accumulated runtime for 100 consequtive reduction operations, in a sample of 5 observations.

Binomial Tree	Local Redirect	Clairvoyant (SMA)	Clairvoyant (true)
13.88 s	13.04 s	12.94 s	12.81 s

predicted PATs. Included in the comparison, are algorithm Binomial Tree and Local Redirect [28]. The utilized PAT data are from the 100 iteration Helsim trace file depicted in Fig. 4 where from the first 64 lines were sampled (due to the limited resources we had at our disposal on the VSC muk[3] cluster machine). In this experiment, the segment count was set to one ($N = 1$), to allow for a fair comparison against the Binomial Tree and Local Redirect algorithms, both of which are designed for atomic input data only.

The developed microbenchmark can be used to evaluate the performance of any MPI collective operation. The list of collective operations to benchmark is assembled at compile time, so adding a new operation requires recompilation. Ditto for combining operators. The input data type can be specified as any of the predefined MPI datatypes. All implementations of the reduction operation have to conform to the standard interface, as defined by MPI and are to be written as function templates with two type parameters: input data type and combining operator type. However, for those algorithms that require side information, using the C++11 `std::bind` function template to perform partial function application allows the user to expand the argument list of reduction operations, while still conforming to the standard interface.

One of the microbenchmarks's key features is the wide range of supported imbalance patterns. These range from singular imbalance (only one process in the communicator delayed), wherein any one process can be selected for suspension, an alternating distribution where even processes are delayed for time d_e and odd processes for time d_o, to various types of stochastically generated imbalance patterns (uniform distribution, normal distribution, gamma distribution and bernoulli distribution). Finally, the microbenchmark can read a trace file consisting of N lines of P values, where each line represents one PAT pattern, and subject all the selected algorithms to this PAT sequence. Some algorithms, like Radix-k, can be further customized through usage of custom input files specified as command line arguments.

To use the microbenchmark, the user at a minimum has to specify the problem size, imbalance pattern, delay magnitude and the number of repetitions as command line arguments. The program is then passed to `mpirun`.

As was done in [45] we summarize in Table 6 the common MPI benchmarks found in the literature, together with the principal statistics they report. A weak point of the majority of these benchmark suites is the lack of rigorous statistical analysis of

[3]VSC muk is a tier-1 cluster machine at Flemish Supercomputing Center. It has 528 compute nodes with two Xeon E5-2670 processors and 64 GiB of RAM memory. The nodes are interconnected with an FDR Infiniband interconnect in a fat tree topology (1:2 oversubscription).

Table 6: Overview of statistical methods applied in MPI benchmarks

Benchmark	mean	min	max	dispersion metric
mpptest [41]	min of means			✗
SKaMPI [42]	✓			std. error
OSU	✓	✓	✓	✗
Intel MPI [38]	✓	✓	✓	✗
MPIBlib [31]	✓			CI of the mean (95%)
MPIBench [43]	✓	✓		sub-sampled data
mpicroscope [44]	✓	✓	✓	✗
Phloem MPI	✓	✓	✓	✗

collected data. One important result of such analysis would be the production of confidence limits for the employed statistics. A common example are interval estimates for the mean. This can be done by applying a hypothesis test that the population mean has a specific value μ, against the alternative that it does not have the value μ. This is done by applying a one-sample Student's t-test or a z-test depending on sample size and assumptions about the population distributions. However, when the mean is not a desirable location estimator, confidence intervals tend to be mathematically difficult to derive. Then, bootstrap methods can be used to obtain confidence intervals [46].
Fig. 11 shows the distribution of algorithm runtime for problem size $m = 128KiB$ and balanced PATs. We can observe that most distributions exhibit positive skew. In light of the fact that the sample median is less susceptible to outliers in data and is a more efficient estimator of location than the sample mean for data with long tails, we decided to adopt it as the location estimator in our analysis. This decision was further supported by bootstrap uncertainty estimates for the mean and median on each of the collected samples, where the sampling distribution of the median was shown to have a smaller standard deviation than that of the mean.

In our report of algorithm runtime, we decided to communicate the ratio of the medians of sampled algorithm runtimes, normalized to algorithm Clairvoyant. As there is no simple traditional method to compute the significance test for such a statistic, a resampling permutation test method was used instead. For each sample point of interest (algorithm, problem size, imbalance), we performed a permutation test with 20000 iterations. Out of all the generated permutations, we counted for each sample point of interest how many have resulted with ratios higher than those observed. This gave us an estimated probability of the observed ratio occurring by chance. Then, for each sample point of interest we reported the highest such probability among the four algorithms (Clairvoyant is excluded as it is the denominator in the ratios). If the computed probability $p \geq 0.05$ then we marked the result as statistically not significant.

A prerequisite to this approach is that the observations are independent and come from a stationary random generation process. To determine stationarity of random generation processes, we used simple quantitative methods such as linear fits to quantify trends and Levene tests to quantify the stationarity of variance. Furthermore, we conducted runs test [46] on all samples to determine the presence of serial correlation in the gathered data. Table 7 shows the results of the runs test analysis. Autocorrelo-

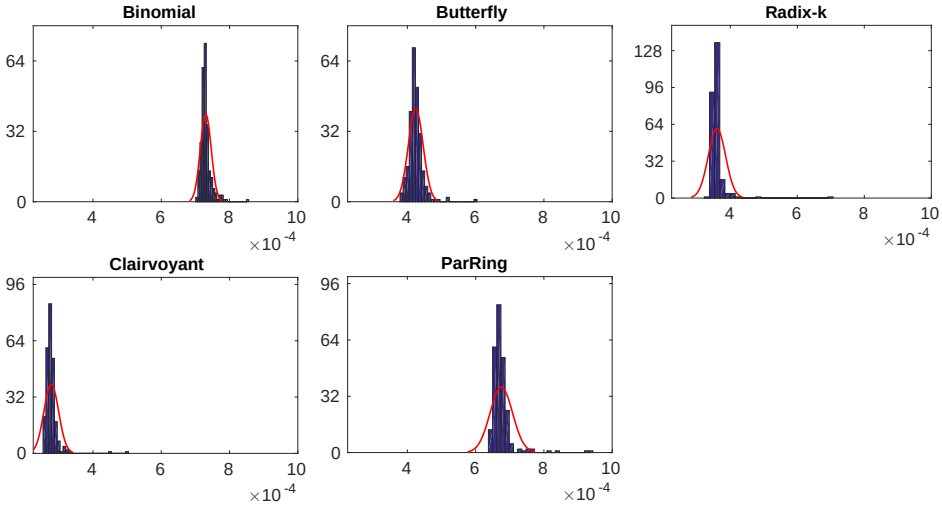

Figure 11: Distribution of algorithm runtime for balanced PATs and problem size $m = 128\,\text{KiB}$. Superimposed on top of each histogram is a fitted normal probability density function. In the plots, the x-axis denotes observed runtime in seconds, while the y-axis denotes the number of observations per bin

grams and spectral plots can be computed to further verify the presence and nature of the serial correlation in the data.

6.3. Experiment design

To evaluate algorithm robustness to imbalanced PATs, we selected the PAT pattern where a single process at rank $P-1$ was delayed. Proposition 3.7 indicates that this is where we should expect to observe the largest absorption time for any given absolute imbalance, and consequently the largest relative speedup for imbalance robust algorithms. In other words, for this PAT pattern, we would expect the ratio $\frac{A(\psi,\mathcal{A})}{t_{\mathcal{A}}}$ to be the largest.

Table 7: Serial correlation in time series data for each algorithm and each problem size. A runs test was performed on each sample with the confidence level set to 95%. Each value in the table reports the ratio of samples that passed the runs test, i.e. where no significant serial correlation was detected.

Algorithm	128 KiB	512 KiB	2 MiB	4 MiB	40 MiB
Binomial	0.77	0.52	0.63	0.86	0.76
Butterfly	0.72	1.00	0.77	0.86	0.84
Parallel Ring	0.61	0.52	0.63	0.91	0.76
Radix-k	0.88	0.90	0.95	0.82	0.76
Clairvoyant	0.77	0.76	0.77	0.78	0.84

The experiment was setup as follows: for each message size $m \in \{128\,\text{KiB}, 512\,\text{KiB}, 2\,\text{MiB}, 4\,\text{MiB}, 40\,\text{MiB}\}$ a different allocation of $P = 128$ nodes on the PRACE CURIE machine was obtained. For each problem size, a set of k absolute imbalances $I_i, i \in \{0 \ldots k\}$ was produced so that the magnitudes of these imbalances spanned from $0t_{\mathcal{C}}$ to anywhere between $2t_{\mathcal{C}}$ and $5t_{\mathcal{C}}$, where $t_{\mathcal{C}}$ denotes the runtime of algorithm Clairvoyant.

Then for each problem size and each imbalance I_i, a PAT ψ was generated, where process at rank $P-1 = 127$ was delayed for time I_i. The experiment then proceeded in r rounds, where in each round the five algorithms were executed in succession. The number of rounds r was 256 for the first two problem sizes, and 100 for the remaining. Input data was of type `MPI_INT` and the utilized combining operator was `MPI_SUM`. The whole procedure is formalized as Algorithm 2.

Table 8: Optimal number of segments N as determined by the linear model and empirical measurement.

Method	128 KiB	512 KiB	2 MiB	4 MiB	40 MiB
Linear model	13.2	27.3	54.7	77.39	244.75
Empirical	16	8	16	16	40

The number of segments N_{opt} for each problem size was determined empirically by measuring the runtime of algorithm Clairvoyant for all $N_{opt} \in \{2,4,8,16,32,64,128\}$ and selecting that N which produced the shortest runtime. For $m = 40\,\text{MiB}$ we added to the set of potential segment sizes the values $\{40,80\}$. The empirically determined value was significantly different from what the linear model predicted, as can be seen from Table 8. This is because the parameters β, γ are not message size invariant. In fact, the ratio $\frac{\beta}{\gamma}$ grows with decreasing message size due to larger relative weight of message overheads. This also explains why the empirically determined optimal segment sizes were smaller than that predicted by the linear model. We conjecture that a more detailed, functional linear model where parameters β, γ are expressed as functions of message size might produce better estimates.

7. Results and discussion

As the results show (Fig. 12), algorithm Clairvoyant dominates the surveyed algorithms in performance for all problem sizes and all absolute imbalances. The computed ratios have been shown by permutation tests to be of very high statistical significance (for most data points, $p \leq 0.001$). While this behaviour was predicted (Fig. 10), in many cases the observed speedup exceeded the predictions. This is due to shortcomings of the simple linear model where the cost parameters β, γ are modeled as message size independent, as discussed in Section 6.3.

Observing the transition from balanced PATs to imbalanced, we see one instance of order inversion: for $m = 4\,\text{MiB}$ algorithm Parallel Ring falls behind Radix-k for imbalanced PATs, while it was faster for balanced PATs. In general, however, we can state that the ordering observed for balanced PATs holds with increasing levels of

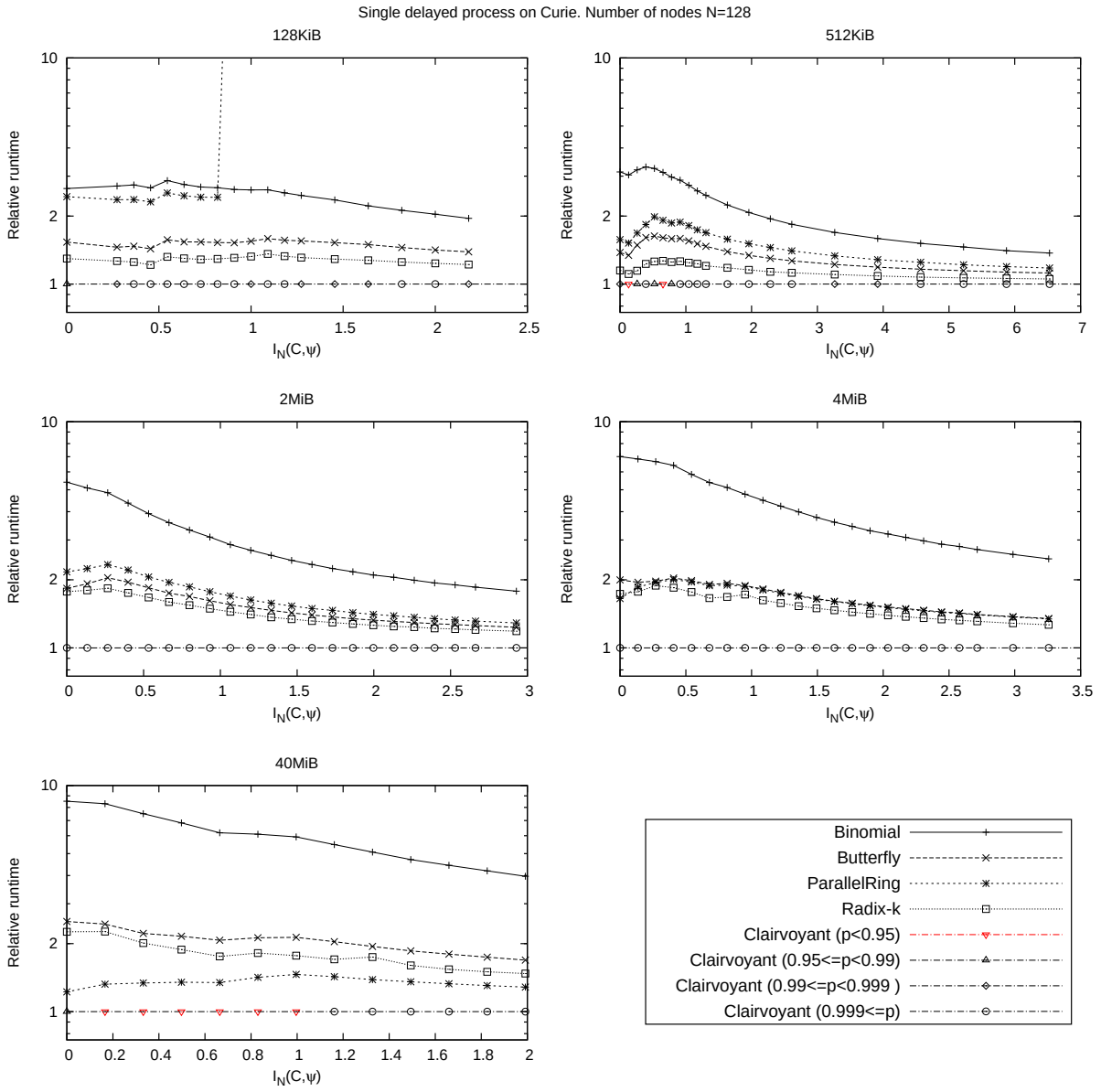

Figure 12: Relative algorithm performance with single delayed process (rank $P-1$). Statistical significance of results is reported by plotting computed p-values of permutation tests on the line $y = 1$. A downward red triangle indicates that the result does not meet the 95% significance threshold. Circles denote very high significance levels of $p \leq 0.001$, while upward triangles and rhombes denote levels in between as depicted in legend

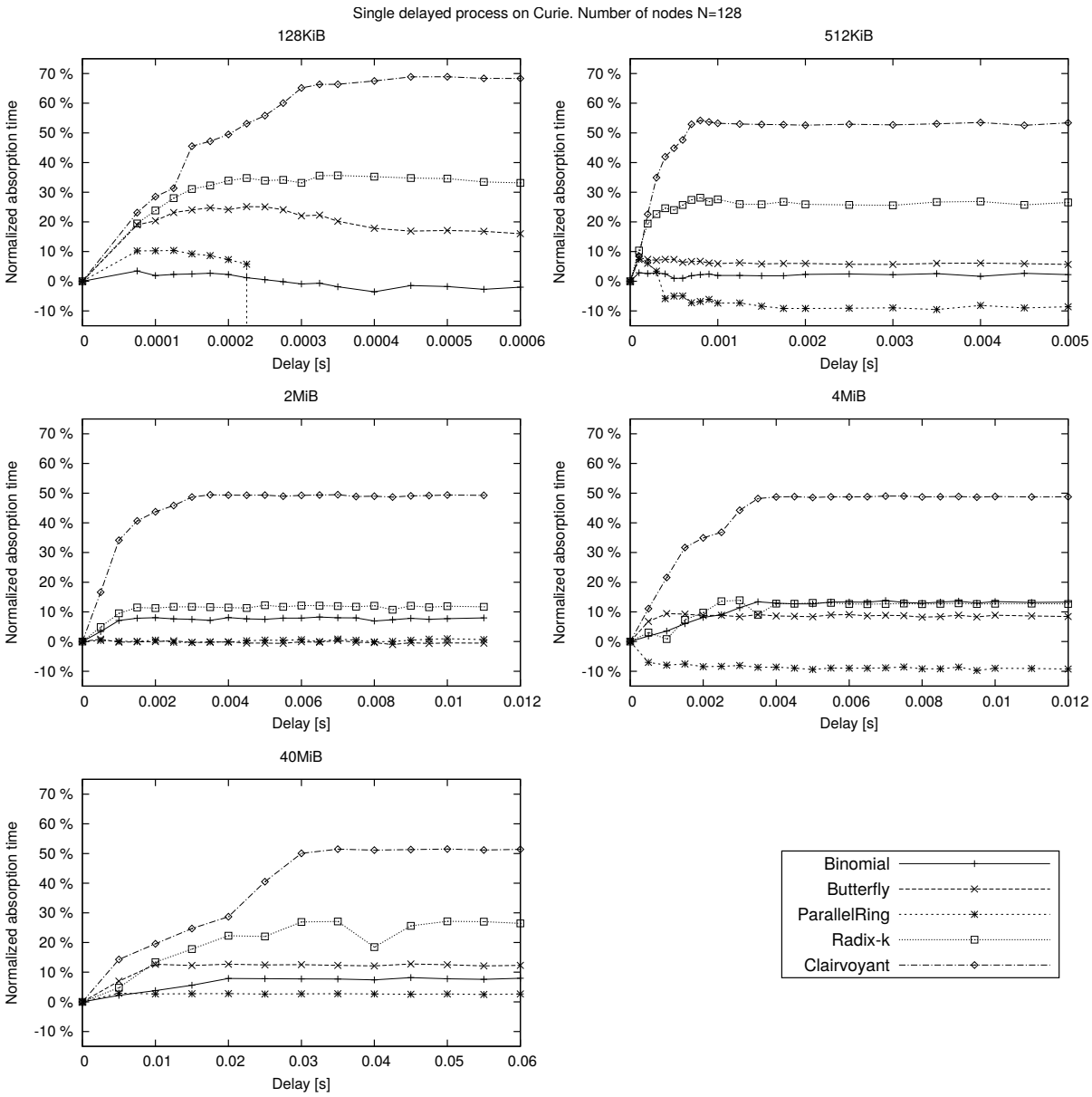

Figure 13: Normalized algorithm absorption time with single delayed process (rank $P-1$).

imbalance. This would imply that the existing collective operation benchmarks could be used to determine the best performing algorithm even for imbalanced PATs.

Algorithm Radix-k consistently outperforms Butterfly for all tested problem sizes, contrary to the linear model prediction. This leads us to believe that latency plays a smaller role than that used to model the time complexity of algorithm Radix-k. For $m = 40\,\text{MiB}$, the runtime of algorithm Parallel Ring is significantly lower than the prediction.

To understand better how each algorithm responded to PATimbalance, we have plotted the normalized absorption times in Fig. 13. A surprising result was that the observed normalized absorption for algorithm Clairvoyant was significantly higher than what we would expect (Fig. 6). For $m = 40\,\text{MiB}$ the maximum observed normalized imbalance I_N was 51%, compared to 14.9% the linear model would predict. For $I(\psi) = 60\,\text{ms}$, the generated schedule length $R = 133$. As one of its input parameters, algorithm Clairvoyant receives the time d required to complete one round of reduction (i.e. send/receive and combine one segment of size m/N). For $m = 40\,\text{MiB}$, we empirically determined that $d = 6.43 \times 10^{-4}\,\text{s}$. From proposition 3.7 and the fact that $P = 2, N = 40 \Rightarrow R = 40$, we can determine whether the schedule length $R = 133$ is consistent by verifying that $d(133-40) = I(\psi)$. Since $d(133-40) = 0.0598\,\text{s}$ we can conclude that the algorithm performed as expected, producing a schedule of minimum length.

However, the expected time $t_{\mathcal{C}}(128, \psi) = R \cdot d = 0.0855\,\text{s}$ is greater than the observed runtime of $0.0747\,\text{s}$. From the observed runtime and the fact that $I(\psi) = 60\,\text{ms}$ we can empirically estimate the time required to perform a 2-way reduction as $t_{\mathcal{C}}(2, \psi) = 0.0146\,\text{s}$. This is considerably shorter than what we would expect: $40 \cdot d = 0.025\,72\,\text{s}$. It would seem that with only two processes communicating, the network bandwidth is 176% higher than when 128 processes are communicating concurrently.

The schedule generation time for $N = 16, P = 128$ was on average $0.55\,\text{ms}$. Extrapolating the empirically determined schedule computation time (Fig. 9), leads us to conclude that in real time usage scenarios, the algorithm will be competitive whenever $m \geq 1\,\text{MiB}$. The PAT aware execution of algorithm Clairvoyant is only possible in iterative settings where the PAT patterns do not significantly change between iterations. In such settings, the reduction schedule can be computed once and reused multiple times and the schedule generation cost fully amortized.

7.1. Catastrophic slowdown

An interesting phenomenon was observed with algorithm Parallel Ring for $m = 128\,\text{KiB}$. For $I(\psi) \geq 0.2\,\text{ms}$ the algorithm experienced a catastrophic slowdown of two orders of magnitude. Its time series data oscillated between high and low values, with excursions into high value territory becoming more prominent with increasing absolute imbalance (Fig. Appendix D). We reproduced this behaviour by re-running the experiment for $m = 128\,\text{KiB}$ on a different day with a different allocation of compute nodes.

Autocorrelograms of the time series data (Appendix E) depict an alternating sequence of positive and negative spikes, slowly decaying to zero and well into statistically significant territory. This makes for a strong argument that the observed phe-

nomenon has a systematic cause, either because of some system interference or the underlying nature of the native MPI implementation.

As of the time of writing this paper, we have no conclusive explanation for the observed phenomenon. The fact that this behaviour was observed only for the problem size $m = 128\,\text{KiB}$, but not for larger problem sizes indicates that the reason might lie in the smaller segment size of $B = 1\,\text{KiB}$ and the possibility that the former were communicated with the eager protocol, while the latter with the rendezvous communication protocol. In the former case, for sufficiently large absolute imbalances, up to $P-1$ messages of size $B = m/P$ would be sent from process rank 0 to rank $P-1$. We conjecture that the manner in which these unexpected receives were handled by the implementation could explain the observed phenomenon.

8. Conclusion

This paper has provided a much needed insight into the performance of MPI reduction algorithms under the presence of imbalanced process arrival times and introduced a novel segmenting reduction algorithm designed with a high degree of imbalance resiliency. Experimental results show that this algorithm universally outperforms all selected reduction algorithms in this study. For some problem sizes, the algorithm was found to be nearly twice as fast as the next fastest algorithm. The algorithm is contingent on full knowledge of process arrival times and constructs the communication schedule prior to the execution of the reduction operation. If the schedule generation is performed at the time the collective operation is invoked, then the speedup obtained outweighs the construction costs for problem sizes larger or equal to 1 MiB. Otherwise, the algorithm can be used in iterative settings where the schedule can be precomputed once and reused multiple times.

Our findings indicate that, excluding the Clairvoyant algorithm, reduction algorithms have little to no resiliency to skewed PATs. An important result is that the ordering of algorithm runtime observed for balanced PATs appears to hold with increasing levels of imbalance. This is reassuring, as all known benchmarks used to optimize MPI collective operation performance ensure that PATs are balanced. However, one algorithm, Parallel Ring, exhibited a catastrophic slowdown of 2 orders of magnitude, for problem size $m = 128\,\text{KiB}$ and increasing magnitudes of imbalance. This result was reproduced by re-running the experiment on a different day and a different allocation of compute nodes.

This paper has important implications on HPC applications, as it has shown that an imbalance resilient reduction algorithm can be produced to consistently outperform reduction algorithms found in library implementations of the MPI programming interface. It would be interesting to investigate whether a dynamic imbalance robust algorithm for non-atomic input data could be devised, similar to that of Local Redirect [28]. The re-ordering of the communication graph could be performed similar to the principle of desynchronization algorithms used in wireless networks [47, 48].

Acknowledgments

This work is funded by Intel, the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT) and by the iMinds institute. Some of the data necessary for the experiments in this paper was produced at the ExaScience Life Lab, Leuven, Belgium. We acknowledge PRACE for awarding us access to resource CURIE based in France at CEA/TGCC-GENCI. Peter Schelkens has received funding from the European Research Council under the European Unions Seventh Framework Programme (FP7/2007-2013)/ERC Grant Agreement n.617779 (INTERFERE).

Appendix A. Algorithm definitions

Appendix B. Autocorrelograms of Helsim simulation image rendering time

Appendix C. Radix-k reduction schedule illustration

Appendix D. Time series data for algorithm Parallel Ring

Appendix E. Autocorrelation of time series data for Parallel Ring

References

[1] E. Meneses, L. V. Kal, Camel: collective-aware message logging, The Journal of Supercomputing 71 (7) (2015) 2516–2538. doi:10.1007/s11227-015-1402-3.

[2] K. B. Ferreira, P. Bridges, R. Brightwell, Characterizing application sensitivity to OS interference using kernel-level noise injection, in: Proceedings of the 2008 ACM/IEEE conference on Supercomputing, SC '08, IEEE Press, Piscataway, NJ, USA, 2008, pp. 19:1–19:12.

[3] X. Y. Ahmad Faraj, Pitch Patarasuk, A study of process arrival patterns for MPI collective operations, International Journal of Parallel Programming 36 (6) (2008) 571–591.

[4] C. Huang, O. Lawlor, L. V. Kale, Adaptive MPI, in: Languages and Compilers for Parallel Computing, Springer, 2004, pp. 306–322.

[5] A. Mamidala, J. Liu, D. K. Panda, Efficient barrier and allreduce on infiniband clusters using multicast and adaptive algorithms, in: Proceedings of the 2004 IEEE International Conference on Cluster Computing, CLUSTER '04, IEEE Computer Society, Washington, DC, USA, 2004, pp. 135–144.

[6] P. Patarasuk, X. Yuan, Efficient MPI bcast across different process arrival patterns, Parallel and Distributed Processing Symposium, International 0 (2008) 1–11. doi:http://doi.ieeecomputersociety.org/10.1109/IPDPS.2008.4536308.

[7] Y. Qian, Design and evaluation of efficient collective communications on modern interconnects and multi-core clusters, Ph.D. thesis, Queen's University (2010).

[8] Message Passing Interface Forum, MPI: A Message-Passing Interface Standard. Version 3.1, available at: http://www.mpi-forum.org/docs/mpi-3.1/ (Feb 2016) (June 4th 2015).

[9] R. M. Karp, A. Sahay, E. E. Santos, K. E. Schauser, Optimal broadcast and summation in the LogP model, in: Proceedings of the fifth annual ACM symposium on Parallel algorithms and architectures, ACM, 1993, pp. 142–153.

[10] G. A. Louis-Claude Canon, Scheduling associative reductions with homogenous costs when overlapping communications and computations, Tech. Rep. 7898, Inria (2012).

[11] R. Rabenseifner, Optimization of collective reduction operations, in: Procs. of Int. Conf. on Computational Science (ICCS), 2004, pp. 1–9.

[12] R. Rabenseifner, J. L. Trff, More efficient reduction algorithms for non-power-of-two number of processors in message-passing parallel systems., in: EuroPVM/MPI, 2004, pp. 36–46.

[13] P. Patarasuk, X. Yuan, Bandwidth optimal all-reduce algorithms for clusters of workstations, Journal of Parallel and Distributed Computing 69 (2) (2009) 117–124.

[14] N. Jain, Y. Sabharwal, Optimal bucket algorithms for large MPI collectives on torus interconnects, in: Proceedings of the 24th ACM International Conference on Supercomputing, ACM, 2010, pp. 27–36.

[15] E. Chan, M. Heimlich, A. Purkayastha, R. van de Geijn, Collective communication: theory, practice, and experience, Concurrency and Computation: Practice and Experience 19 (13) (2007) 1749–1783.

[16] T. Peterka, D. Goodell, R. Ross, H.-W. Shen, R. Thakur, A configurable algorithm for parallel image-compositing applications, in: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, SC '09, ACM, New York, NY, USA, 2009, pp. 4:1–4:10. doi:10.1145/1654059.1654064.

[17] W. Kendall, T. Peterka, J. Huang, H.-W. Shen, R. Ross, Accelerating and benchmarking radix-k image compositing at large scale, in: Proceedings of the 10th Eurographics conference on Parallel Graphics and Visualization, EG PGV'10, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 2010, pp. 101–110. doi:10.2312/EGPGV/EGPGV10/101-110.

[18] J. Pjesivac-Grbovic, T. Angskun, G. Bosilca, G. E. Fagg, G. J. J. Dongarra, Performance analysis of MPI collective operations, in: IEEE International Parallel and Distributed Processing Symposium, 2005.

[19] P. Sanders, J. Speck, J. L. Trff, Two-tree algorithms for full bandwidth broadcast, reduction and scan, Parallel Computing 35 (12) (2009) 581 – 594, selected papers from the 14th European PVM/MPI Users Group Meeting.

[20] R. Thakur, R. Rabenseifner, W. Gropp, Optimization of collective communication operations in MPICH, International Journal of High Performance Computing Applications 19 (1) (2005) 49–66.

[21] T. Hoefler, D. Moor, Energy, Memory, and Runtime Tradeoffs for Implementing Collective Communication Operations, Journal of Supercomputing Frontiers and Innovations 1 (2) (2014) 58–75.

[22] S. P. Fabrizio Petrini, Darren J. Kerbyson, The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q, in: Proceedings of the 2003 ACM/IEEE conference on Supercomputing, SC '03, 2003, pp. 55–.

[23] S. Agarwal, R. Garg, N. K. Vishnoi, The impact of noise on the scaling of collectives: a theoretical approach, in: Proceedings of the 12th international conference on High Performance Computing, HiPC'05, Springer-Verlag, Berlin, Heidelberg, 2005, pp. 280–289. doi:10.1007/11602569_31.

[24] T. Hoefler, T. Schneider, A. Lumsdaine, Characterizing the influence of system noise on large-scale applications by simulation, in: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, SC '10, IEEE Computer Society, Washington, DC, USA, 2010, pp. 1–11. doi:10.1109/SC.2010.12.

[25] P. Ghysels, T. J. Ashby, K. Meerbergen, W. Vanroose, Hiding global communication latency in the gmres algorithm on massively parallel machines, SIAM J. Scientific Computing 35 (1).

[26] K. B. Ferreira, P. G. Bridges, R. Brightwell, K. T. Pedretti, The impact of system design parameters on application noise sensitivity, Cluster computing 16 (1) (2013) 117–129.

[27] A. E. Eichenberger, S. G. Abraham, Impact of load imbalance on the design of software barriers, in: In Proceedings of the 1995 International Conference on Parallel Processing, 1995, pp. 63–72.

[28] P. Marendic, J. Lemeire, T. Haber, D. Vucinic, P. Schelkens, An investigation into the performance of reduction algorithms under load imbalance, in: C. Kaklamanis, T. Papatheodorou, P. Spirakis (Eds.), Euro-Par 2012 Parallel Processing, Vol. 7484 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2012, pp. 439–450. doi:10.1007/978-3-642-32820-6_44.

[29] E. W. Chan, M. F. Heimlich, A. Purkayastha, R. A. Van De Geijn, On optimizing collective communication, in: Cluster Computing, 2004 IEEE International Conference on, IEEE, 2004, pp. 145–155.

[30] J. L. Träff, A. Ripke, Optimal broadcast for fully connected processor-node networks, Journal of Parallel and Distributed Computing 68 (7) (2008) 887–901.

[31] A. Lastovetsky, V. Rychkov, M. OFlynn, Mpiblib: Benchmarking MPI communications for parallel computing on homogeneous and heterogeneous clusters, in: Recent Advances in Parallel Virtual Machine and Message Passing Interface, Springer, 2008, pp. 227–238.

[32] R. W. Hockney, The communication challenge for mpp: Intel paragon and meiko cs-2, Parallel Comput. 20 (3) (1994) 389–398. doi:10.1016/S0167-8191(06)80021-9.

[33] M. L. Fredman, R. Sedgewick, D. D. Sleator, R. E. Tarjan, The pairing heap: A new form of self-adjusting heap, Algorithmica 1 (1-4) (1986) 111–129.

[34] S. Pettie, Towards a final analysis of pairing heaps, in: Foundations of Computer Science, 2005. FOCS 2005. 46th Annual IEEE Symposium on, IEEE, 2005, pp. 174–183.

[35] K.-L. Ma, J. S. Painter, C. D. Hansen, M. F. Krogh, Parallel volume rendering using binary-swap compositing, Computer Graphics and Applications, IEEE 14 (4) (1994) 59–68.

[36] D.-L. Yang, J.-C. Yu, Y.-C. Chung, Efficient Compositing Methods for the Sort-Last-Sparse Parallel Volume Rendering System on Distributed Memory Multicomputers, The Journal of Supercomputing 18 (2) (2001) 201–220. doi:10.1023/A:1008165001515.

[37] W. Gropp, E. Lusk, Reproducible measurements of MPI performance characteristics, Springer-Verlag, 1999, pp. 11–18.

[38] I. corporation, Intel MPI benchmarks (2013).
URL https://software.intel.com/en-us/articles/intel-mpi-benchmarks

[39] T. Hoefler, T. Schneider, A. Lumsdaine, Accurately measuring overhead, communication time and progression of blocking and nonblocking collective operations at massive scale, Int. J. Parallel Emerg. Distrib. Syst. 25 (4) (2010) 241–258. doi:10.1080/17445760902894688.

[40] T. S. T. Hoefler, A. Lumsdaine, Accurately measuring overhead, communication time and progression of blocking and nonblocking collective operations at massive scale, International Journal of Parallel, Emergent and Distributed Systems 25 (4) (2010) 241–258.

[41] W. Gropp, E. Lusk, Reproducible measurements of mpi performance characteristics, in: Recent Advances in Parallel Virtual Machine and Message Passing Interface, Springer, 1999, pp. 11–18.

[42] R. Reussner, P. Sanders, J. L. Träff, Skampi: A comprehensive benchmark for public benchmarking of MPI, Scientific Programming 10 (1) (2002) 55–65.

[43] D. A. Grove, P. D. Coddington, Communication benchmarking and performance modelling of mpi programs on cluster computers, The Journal of Supercomputing 34 (2) (2005) 201–217.

[44] J. L. Träff, mpicroscope: Towards an MPI benchmark tool for performance guideline verification, in: Recent Advances in the Message Passing Interface - 19th European MPI Users' Group Meeting, EuroMPI 2012, Vienna, Austria, September 23-26, 2012. Proceedings, 2012, pp. 100–109. doi:10.1007/978-3-642-33518-1_15.

[45] S. Hunold, A. Carpen-Amarie, J. L. Träff, Reproducible MPI micro-benchmarking isn't as easy as you think, in: Proceedings of the 21st European MPI Users' Group Meeting, ACM, 2014, p. 69.

[46] NIST/SEMATECH, e-handbook of statistical methods, available at: http://www.itl.nist.gov/div898/handbook/ (Jun 2015) (2012).

[47] D. Buranapanichkit, N. Deligiannis, Y. Andreopoulos, Convergence of desynchronization primitives in wireless sensor networks: A stochastic modeling approach, CoRR abs/1411.2862.

[48] N. Deligiannis, J. F. Mota, G. Smart, Y. Andreopoulos, Fast desynchronization for decentralized multichannel medium access control, Communications, IEEE Transactions on 63 (9) (2015) 3336–3349.