Discrete Event Execution with One-Sided and Two-Sided GVT Algorithms on 216,000 Processor Cores

KALYAN S. PERUMALLA, Oak Ridge National Laboratory
ALFRED J. PARK, Microsoft Corporation
VINOD TIPPARAJU, Advanced Micro Devices, Inc.

Global virtual time (GVT) computation is a key determinant of the efficiency and runtime dynamics of parallel discrete event simulations (PDES), especially on large-scale parallel platforms. Here, three execution modes of a generalized GVT computation algorithm are studied on high-performance parallel computing systems: (1) a synchronous GVT algorithm that affords ease of implementation, (2) an asynchronous GVT algorithm that is more complex to implement but can relieve blocking latencies, and (3) a variant of the asynchronous GVT algorithm to exploit one-sided communication in extant supercomputing platforms. Performance results are presented of implementations of these algorithms on up to 216,000 cores of a Cray XT5 system, exercised on a range of parameters: optimistic and conservative synchronization, fine- to medium-grained event computation, synthetic and non-synthetic applications, and different lookahead values. Performance to the tune of tens of billions of events executed per second is registered, and asynchronous GVT algorithms are observed to generally outperform state-of-the-art synchronous GVT algorithms. Detailed PDES-specific runtime metrics are presented to further the understanding of tightly-coupled discrete event dynamics on massively parallel platforms.

Categories and Subject Descriptors: C.2.2 [**Computer Systems Organization**]: Computer-Communication Networks—*Network Protocols*; C.5.1 [**Computer Systems Organization**]: Computer System Implementation—*Large and Medium Computers*; I.6.1 [**Computing Methodologies**]: Simulation and Modeling—*Discrete*; I.6.8 [**Computing Methodologies**]: Simulation and Modeling—*Types of Simulation (Discrete Event, Parallel)*

General Terms: Algorithms, Design, Experimentation, Performance

Additional Key Words and Phrases: Parallel Discrete Event Simulation, Time Warp, Global Virtual Time, One-sided Communication, Asynchrony

ACM Reference Format:
Perumalla, K. S., Park, A. J., Tipparaju, V., 2012. Discrete Event Execution with One-Sided and Two-Sided GVT Algorithms on 216,000 Processor Cores ACM Trans. Model. Comput. Simul. 0, 0, Article 0 (2012), 42 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. INTRODUCTION

Parallel discrete event simulation (PDES) is used for simulating large scenario configurations in several important areas such as epidemiological outbreak phenomena, Internet modeling, vehicular transportation, emergency/event planning, and social behavioral simulations, to name a few. Discrete event execution evolves the states of the underlying entities in an asynchronous fashion, in contrast to time-stepped execu-

Author's addresses: K. S. Perumalla, Computational Sciences and Engineering Division, Oak Ridge National Laboratory, Oak Ridge, Tennessee, USA. A. J. Park, Microsoft Corporation, Redmond, Washington, USA. V. Tipparaju, Advanced Micro Devices, Austin, Texas, USA.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

tion in traditional scientific computing applications in which the entire system state is (logically) updated over fixed time steps. In general, PDES represents a class of codes that are challenging to scale to large number of processors, due to tight global timestamp-ordering and fine-grained event execution. Parallel runtime engines for discrete event simulations need to deliver fast and accurate global timestamp-ordered execution across a large number of processors. The parallel runtime engine provides correctness and speed of execution by guaranteeing preservation of the dependencies, absence of livelocks and deadlocks, and facilitating rapid progress. A major challenge in scaling PDES runtime engines is the design and development of appropriate algorithms for virtual time synchronization. Equally critical is also the verification of the synchronization efficiency at large parallel computing scales on a variety of PDES models.

However, the detailed performance effects of actual virtual time synchronization algorithm implementations are relatively unknown in understanding scalability to massively parallel platforms, Few synchronization algorithms have thus far been gainfully employed on supercomputers with many thousands of processor cores, Much remains to be explored about the dynamics of discrete execution on a range of representative applications and benchmarks. Also, advanced network mechanisms such as one-sided communication of massively parallel platforms have not been exploited for virtual time synchronization and discrete event execution before (related work is discussed in greater detail in Section 5).

The focus of this article is in advancing virtual time synchronization to massively parallel platforms, exploiting specific hardware mechanisms that such large installations offer, and studying the dynamics of discrete event execution. Mechanisms explored here include synchronous as well as asynchronous execution and the notion of sidedness in terms of conventional "two-sided" communication, and the native "one-sided" communication natively supported on advanced parallel systems. The work presented here is the first in the literature in the use of one-sided communication for GVT implementation on massively parallel systems.

1.1. Global Virtual Time

In PDES, independent logical processes (LPs) hold encapsulated states and evolve their states along a virtual time axis, and exchange timestamped events to incorporate inter-LP data dependencies. In *conservative* PDES, an LP does not execute an event until it can guarantee that no event with a smaller timestamp will later be received by that LP. In *optimistic* PDES, events are potentially executed before such a guarantee can be obtained, but, suitable corrective action (called rollback) is performed on the incorrectly processed events if any timestamp order violation is later discovered. PDES runtime engines may support conservative, optimistic, or both (mixed) approaches.

At the core of execution of PDES engines of all types is the parallel/distributed synchronization of virtual time to correctly process the events in conservative or optimistic fashion. Fast virtual time synchronization algorithms rapidly compute a quantity called the *global virtual time* (GVT), to directly speed up the distributed wave of progress of all processors executing events staggered along the global virtual timeline. The fine-grained nature of event execution imposes tight constraints on GVT algorithms with respect to scalability. Thus, a performance-critical aspect of any PDES engine is the specific GVT computation algorithm it employs.

Multiple, largely equivalent, definitions of GVT are possible; see [Gomes et al. 1998; Fujimoto 1999] for surveys. Here, we shall employ one such view in which GVT is a virtual time value T_{min} such that no processor shall receive any event E with a timestamp T_E such that $T_E < T_{min}$. Thus, each processor, after receiving a value of T_{min}, can commit local event processing until T_{min} without fear of data dependency

violations. Clearly, the rapidity with which T_{min} can be advanced globally has a direct bearing on the speed with which processors can concurrently execute their event work loads.

1.2. Two-sided *vs.* One-sided Communication

In relation to high performance execution of GVT computation on massively parallel systems, GVT algorithms must take into account a communication concept called *sidedness* which raises important systems-level effects at scale. Conventional message passing communication on distributed memory platforms fall in the category of "two-sided" communication, while a more direct, memory-to-memory interface between processors is supported in "one-sided" communication. A functional view of the two paradigms is shown in Figure 1.

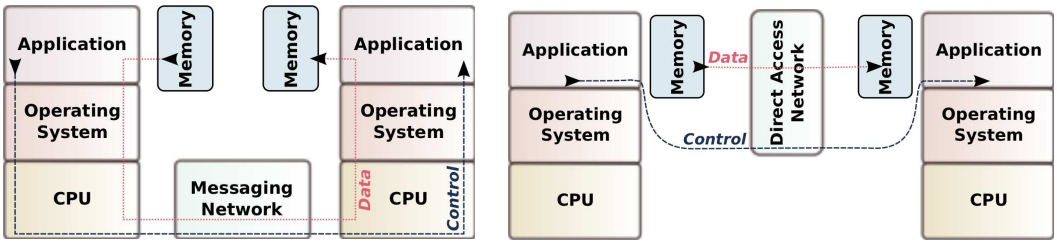


Fig. 1: Functional view of two-sided vs. one-sided communication systems

In two-sided communication, both the receiver and sender sides of the application participate in every data exchange. Data and control information are transmitted via the operating system, the central processor unit (CPU), and the messaging network from the sender to the receiver. Two-sided communication essentially requires both sides of the exchange to coordinate for any data transmission. On the other hand, one-sided communication typically provides a more direct-transfer interface in which a copy of the data from a memory location of the sender is sent to another memory location of the receiver on another processor. Importantly, such a transfer can be performed by the sender *without the participation of the receiver* to complete (and make record of) the transfer. Control information regarding the inter-processor mappings among memory locations and the signaling of events such as start and end of transfer is also transmitted asynchronously via the direct access network. When one-sided communication *interfaces* are supported over two-sided *implementations*, they are not truly one-sided in actual execution. For example, the Message Passing Interface (MPI) standard provides interface routines such as `MPI_Get()` and `MPI_Put()` that are one-sided in semantics, but MPI does not guarantee actual one-sided implementation of those routines. By contrast, our focus here is on using actual, natively supported implementations of one-sided communication on massively parallel platforms, such as the `Portals` implementation on a Cray XT5 machine (explained later in detail).

In almost all parallel systems that support one-sided communication, two-sided communication is also provided as an additional interface that is implemented over either a dedicated fraction of the one-sided communication network or an entirely separate network. Due this facility, with one-sided communication, GVT information can be exchanged over the one-sided network while event data is exchanged over the two-sided network, thereby separating the two distinct use cases. The potential advantages of one-sided messaging are: (1) GVT messaging is separated from event communication,

thereby eliminating competition, and its resultant latency increase, for GVT messages, (2) overheads of dynamic memory remapping are avoided due to static inter-processor messaging structure for GVT messages.

1.3. Organization

Due to the complex interactions among model behaviors, hardware features, and software characteristics, the actual scalability and efficiency of any GVT algorithm can only be properly evaluated with actual implementation and benchmarking of PDES engines and applications at scale. To evaluate our GVT algorithms, we study their performance along four different dimensions in PDES application characteristics:

(1) event dependency structure, determined by the application's event computation characteristics such as event granularity, and the distribution of timestamps dynamically generated by events,
(2) conservative or optimistic synchronization, which determines whether some local events can be processed beyond GVT,
(3) lookahead, which is a measure of static concurrency available in the application scenario, and
(4) inter-processor messaging types, categorized here as two-sided and one-sided.

The rest of the paper is organized as follows. The GVT algorithms are described in Section 2, and their implementation details are presented in Section 3. A detailed performance study on a variety of PDES benchmarks is described in Section 4. Prior, related work on virtual time synchronization algorithms is covered in Section 5. The paper is concluded and potential future work is identified in Section 6.

2. GVT ALGORITHMS

In a typical PDES execution, the execution engine operates in a loop to process local events (main loop), and also participates in inter-processor synchronization for GVT. The GVT computation, in general, is performed in a separate module (GVT loop) which may be inlined within the main loop or executed in its own thread. Based on the specific needs of the synchronization scheme employed by the engine, a GVT computation is initiated inside the main loop. For example, a conservative engine initiates a new GVT computation when it runs out of local events to process safely. An optimistic execution initiates either at a predefined frequency or on demand when memory used for rollback support needs to be reclaimed. Fast GVT advancement can improve caching behavior, since it can reduce the size of the working set by quickly committing, reclaiming, and reusing a small number of memory buffers for events.

2.1. Execution Modes

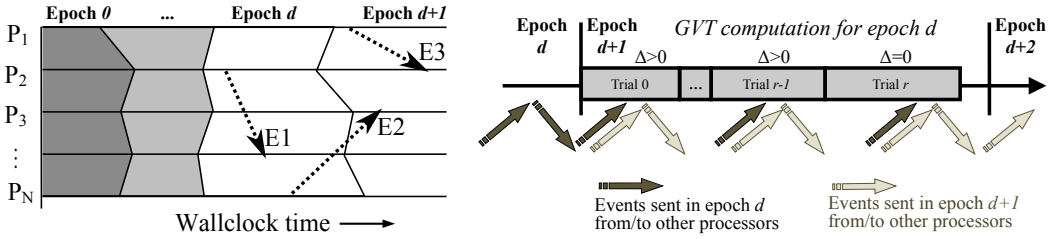
Here, we focus on three execution modes of a generalized GVT algorithm, covering the space of synchronous *vs*. asynchronous execution and two-sided *vs*. one-sided communication:

1 **Two-sided Synchronous**: Whenever the engine initiates a GVT computation, it blocks until the computation terminates and the new GVT value is obtained from it.
2a **Two-sided Asynchronous**: In this mode, the GVT computation and the engine's main loop are concurrently active. Two-sided inter-processor communication is used for event exchanges as well as GVT messages.
2b **One-sided Asynchronous**: Just as in 2a, GVT and event loops are concurrent, but they are independent with respect to communication. The GVT is computed via one-sided communication.

All modes support both conservative and optimistic execution. With two-sided communication, both synchronous and asynchronous execution are possible. With one-sided communication, only asynchronous execution is sensible due to its primary benefit being the overlap of receiver's computation with communication.

2.2. Distributed Snapshots-based GVT Computation

Our unified GVT algorithmic template is based on the general approach of computing distributed snapshots [Mattern 1993; Choe and Tropper 1998]. In this approach, PDES execution is divided by the GVT algorithm into epochs, as illustrated in Figure 2. Each epoch is a distributed snapshot such that no event "goes backward" across epochs. In other words, for every event sent by a processor in epoch numbered d, the event is only received at the destination processor in the same epoch d or later epochs $d' > d$, but never in a previous epoch $d' < d$. For example, in Figure 2, event **E1** is entirely contained in epoch d, while **E2** crosses epoch d into $d+1$, and **E3** is contained within epoch $d+1$. In our algorithm, it is guaranteed that all events sent in epoch d are fully received at their destinations in epochs d or $d+1$, but never beyond $d+1$. This is achieved by ensuring that the global number of events Δ sent in d yet to be received by their destinations is equal to zero. Events still in flight (sent but not received) are indicated by $\Delta > 0$. The GVT computation is designed to carefully demarcate epoch boundaries at each processor such that they constitute a distributed snapshot while also computing the least event timestamp across all processors.



(a) Epoch numbering and events

(b) Trial sequence and termination

E1=Sent and received in epoch d
E2=Sent in epoch d, received in $d+1$
E3=Sent and received in epoch $d+1$

Δ=Count of events sent in epoch d not yet received by their destinations
Trial=Parallel computation of Δ

Fig. 2: Distributed snapshots-based GVT computation

2.3. Unified GVT Algorithm Template

Algorithm 1 shows the pseudocode of our GVT algorithmic template. It is unified to suit all the three execution modes of operation, parametrized by boolean flags `synchronous` and `conservative` to denote synchronous *vs*. asynchronous execution and conservative *vs*. optimistic execution, respectively. The template includes four procedures: (1) the main simulation loop $\mathcal{ML}$, (2) the GVT computation loop $\mathcal{GL}$, (3) the procedure $\mathcal{IE}$ that accounts for every incoming inter-processor event, and (4) the procedure $\mathcal{OE}$ that tags every outgoing inter-processor event.

A variable d is used as a counter of the number of GVT computations performed so far, which is the epoch number. Each GVT computation proceeds as sequence of *trials*, which are successive reductions to determine the number Δ of transient events "in flight." The trials are counted by the variable r ($\mathcal{GL}$ line 8), starting at 0 for each epoch d. The transient event count is computed as a global reduction with the addition

Algorithm 1 GVT algorithmic template executed at every processor p

Variables

Name		Initial Value	Description
$synchronous$	$\leftarrow$	user-defined	Is synchronous execution desired?
$conservative$	$\leftarrow$	system-defined	Is the main loop conservative?
$active$	$\leftarrow$	**false**	Is a new GVT value being computed?
d	$\leftarrow$	0	Current GVT epoch number
$\delta_p[d]$	$\leftarrow$	0 for every $d \geq 0$	p's contribution to the total count of transient messages sent in epoch d across all processors
$\tau_p[d]$	$\leftarrow$	∞ for every $d \geq 0$	Lowerbound on any event timestamp in epoch d sendable from p and receivable by any processor
LVT_p	$\leftarrow$	$\min(\{T_E\})$	Least of all event timestamps $\{T_E\}$ at p
LA	$\leftarrow$	user-defined ≥ 0	Lookahead on inter-processor event times

$\mathcal{ML}$: Main Loop

```
1:  while GVT < end time do
2:    while GVT < LVT_p and (not active) do
3:      active ← true
4:      if synchronous or conservative then
5:        Wait until not active
6:      end if
7:      Update LVT_p
8:    end while
9:    if conservative then
10:     Execute all E(T_E), T_E ≤ GVT
11:   else
12:     Commit all E(T_E), T_E ≤ GVT
13:     and perform rollbacks, if any
14:     or execute some E(T_E), T_E > GVT
15:   end if
16: end while
```

$\mathcal{GL}$: GVT Loop

```
1:  start: Wait until active
2:  d' ← d
3:  d ← d + 1
4:  r ← 0
5:  τ_p[d'] ← min(τ_p[d'], LVT_p + LA)
6:  repeat
7:      Δ ← Σ_{q=1}^{N} δ_q[d']      } Composite
        T ← min_{q=1}^{N} τ_q[d']    } reduction
8:    r ← r + 1
9:  until Δ = 0
10: GVT ← T
11: active ← false
12: goto start
```

$\mathcal{IE}$: Handle Incoming Event $E(T_E, d_E)$

```
1: δ_p[d_E] ← δ_p[d_E] − 1
2: if active then
3:   τ_p[d_E] ← min(τ_p[d_E], T_E + LA)
4: else
5:   LVT_p ← min(LVT_p, T_E)
6: end if
```

$\mathcal{OE}$: Tag Outgoing Event $E(T_E)$

```
1: δ_p[d] ← δ_p[d] + 1
2: Tag E as E(T_E, d)
```

operator on the difference between the number of events sent in previous epoch and the number received in previous or current epoch. Together with the summation, a global minimum reduction operator is also applied on the minimum local timestamps at each processor. This combined reduction of transient message count and the minimum timestamp is indicated as composite reduction in line 7 of $\mathcal{GL}$. When Δ becomes zero, the globally reduced minimum time is usable as a (non-decreasing) GVT value (line 10 of GVT loop $\mathcal{GL}$). If Δ is non-zero, then, another reduction is started to determine if there has been progress in event delivery.

In optimistic discrete event executions, retractions (anti-messages) are treated as regular events by using their timestamps just as those for regular messages.

The data structures $\delta_p[d]$ and $\tau_p[d]$ are shown to be arrays of length equal to the number of GVT computations. The lengths of the arrays are shown this way for simplicity and clarity of understanding. In practice, the arrays need only have two elements each, and all references by index d can be safely replaced by $d \mod 2$. Thus, every reference to $\delta_p[d]$ is replaced by $\delta_p[d \mod 2]$ and every $\tau_p[d]$ by $\tau_p[d \mod 2]$ in the implementation of $\mathcal{GL}$, $\mathcal{IE}$, and $\mathcal{OE}$ portions of Algorithm 1. This works correctly because no event spans more than two epochs: every event $E(T_E, d_E)$ tagged by the source processor in epoch d_E is received by the destination processor only in epochs d_E or d_{E+1}.

2.4. Correctness of GVT Computation

Here we present the correctness conditions for the GVT computation and outline a proof sketch for the correctness of the algorithm. Consider a globally frozen snapshot of the PDES execution. The GVT value can be easily computed as the minimum among the timestamps of the events at all processor and the events in flight within the network. However, this "ideal" value GVT cannot be efficiently computed in practice because execution cannot be frozen precisely at the same time on a large number of processors.. Instead, the GVT algorithms compute an estimate $\widetilde{GVT} \leq GVT$ that is always bounded by the ideal GVT value. The challenge is to advance $\widetilde{GVT}$ as fast and as close to GVT as possible without violating the properties of GVT. In particular, the properties of the ideal GVT value must be reflected: $\widetilde{GVT}$ should never regress, and no processor should ever receive an event with timestamp less than $\widetilde{GVT}$. These requirements translate to the following important correctness conditions in the implementation of GVT computation [Fujimoto and Hybinette 1997; Holder and Carothers 2008].

— **Transient messages**: For any epoch, a transient message is an event in flight (sent but not yet received) that needs to be accounted in its epoch. In other words, the timestamp T_E of every such event $E(d_E, T_E)$ in the network must be included in the GVT computed in epoch d_E. Otherwise, the GVT value can regress because failure to account for E can take $\widetilde{GVT}$ farther than safety ($T_E < \widetilde{GVT}$) when E eventually arrives at its destination. This violation of correctness is avoided by rejecting all candidates for $\widetilde{GVT}$ when even a single transient event exists in the network. Finally, when no transient event exists, all events are present in processor memories, and hence their timestamps are all included in the global minimum. The algorithm uses this approach in the GVT loop (line 6 to line 9) by rejecting all candidate $\widetilde{GVT}$ until no transient events exist as indicated by $\Delta = 0$.

— **Simultaneous Reporting**: The GVT computation must also ensure that every event always has at least one processor that takes ownership of the timestamp of the event insofar as accounting for its timestamp. This is achieved by a combination of two conditions: the atomicity of the establishment of the cut point line 3 and the fact that no transient messages exist when $\widetilde{GVT}$ is evaluated at line 7.

A more rigorous proof of correctness can be derived using proof by induction for the transient message problem and proof by contradiction for the simultaneous reporting problem, along the lines of the proofs in [Holder and Carothers 2008].

2.5. Synchronous Two-sided GVT

The synchronous two-sided execution mode is achieved in Algorithm 1 by setting the *synchronous* variable to **true**, and using a blocking reduction operation (e.g.,

`MPI_Allreduce()` of MPI) for the global reduction performed in the GVT loop $\mathcal{GL}$ at line 7. This mode is a generalization of previous algorithms in the literature [Perumalla and Fujimoto 2001; Holder and Carothers 2008; Bauer Jr. et al. 2009], enhanced to support conservative as well as optimistic execution *with lookahead*.

While this execution mode is relatively easy to implement (e.g., using the blocking collectives of MPI), the blocking nature requires every processor to stop processing its events while the GVT is being computed. It also is prevented from sending any additional events (as part of executing local events) to other processors. In conservative execution, in order to prevent other processors from blocking for too long, every processor must join the GVT computation periodically, even if that processor itself locally has events to safely process. In optimistic execution, processors must quit optimistic event processing while being blocked. Thus, blocking in both modes is detrimental, especially since the blocked time increases with the number of processors. On massively parallel platforms, the blocked time can grow substantially. Note that hardware-accelerated collectives (e.g., Blue Gene collective networks [Almási et al. 2005; Faraj et al. 2009]) help only a little in decreasing the time taken for the collectives because, the blocked time is dominated by the time difference between the first and last joining processors, which cannot be accelerated by hardware. Nevertheless, this synchronous algorithm can work well for well-balanced work loads in which the event time stamp distribution is relatively uniformly spread across all processor timelines.

2.6. Asynchronous Two-sided GVT

The asynchronous two-sided execution mode is achieved in Algorithm 1 by setting the *synchronous* variable to **false**, and using a *non-blocking* implementation of global reduction operation performed in the GVT loop $\mathcal{GL}$ at line 7.

We implemented an optimized non-blocking reduction operator over the two-sided MPI point-to-point messaging primitives. This asynchronous, application-level reduction is performed using a tree topology that uses a butterfly communication pattern among all nodes at the node-level, in which only one core (core 0) per node participates. Each core 0 communicates internally with other cores on its node using a centralized communication topology, thus minimizing network traffic and making effective use of shared memory communication within the node.

Note that the processor starts and leaves an active GVT computation in the main loop $\mathcal{ML}$ at line 3, thereby avoiding blocking. While GVT is being computed, it continues to execute any additional local events that are processable (safe events in conservative execution, or future events in optimistic execution). Additionally, it is also free to send and receive events without any restrictions, unlike in the previous synchronous two-sided algorithm.

2.7. Asynchronous One-sided GVT

The one-sided GVT operates as in asynchronous two-sided GVT, with one major difference: the reduction operations are performed asynchronously using direct-memory operations on remote processors, with data transfer carried out asynchronously by the network. This achieves non-blocking operation for GVT messages in a way that is completely decoupled from event messaging. Normally (as in the previous two-sided asynchronous algorithm), non-blocking GVT must perform its synchronization via messaging that is multiplexed along with incoming and outgoing event communication. Since GVT messages compete with event messages, the mixed communication can impose latency for GVT messages, thereby delaying GVT completion. On the other hand, assuming *efficient* implementations of one-sided communication on the parallel machine, GVT messages can be exchanged with minimal delay by decoupling them from the event communication.

Thus, the one-sided GVT algorithm must arrange memory buffers across processors in such a way that asynchronous reductions at line 7 are all performed using one-sided communication. To enable such operation, the memory organization maintained on each processor is shown in Figure 3, with arrows showing the potential one-sided transfer of data from the send buffers of processor P_i to the receive buffers of processor P_j. Since GVT computation proceeds asynchronously with the main event processing loop, some processors complete a given GVT epoch d earlier than others and may proceed to initiate the next epoch $d + 1$. Analogously, a trial r within an epoch d may complete on one processor which proceeds to its next trial $r + 1$, thereby sending information belonging to epoch d and trial $r + 1$ while the receiving processor may still be in the process of completing the earlier epoch d, trial r. Hence, at any given moment, every processor must maintain four different blocks of receivable data: $\{(d, r), (d, r + 1), (d + 1, r), (d + 1, r + 1)\}$, to keep the asynchronous computations independent of each other.

The asynchronous reductions are performed using the same inter-processor structure as for the asynchronous two-sided GVT mode. Wtih the same tree topology optimized for hierarchical reductions on multi-core architectures, the inter-processor structure is fixed for GVT messaging, determined and initialized before beginning the main simulation loop.

The unit of memory layout for the GVT data structures is a fixed message size (a `C struct`) defined to hold a GVT message type, which contains the tuple $< P_{source}, d, r, LVT_{source}, \delta >$. Additionally, room for *jumpstart* messages is also allocated such that processors may jumpstart other processors (within or outside its hierarchy) to begin participating in a GVT computation. Some processors may need to be informed so, because, during their own asynchronous event processing, they may not themselves need any additional GVT advances until they run out of local event execution work. The jumpstart messages thus help inform processors when they need to participate in GVT computations started by other processors.

3. IMPLEMENTATION

We now present implementation details of the algorithms incorporated into the μsik discrete event execution engine [Perumalla 2005; Perumalla et al. 2011]. In μsik, conservative, optimistic, and mixed synchronization are supported. A new GVT computation is always initiated as soon as a previous GVT completes, to minimize blocking for conservative LPs, and to minimize uncommitted activity for optimistic LPs. All the GVT algorithms have been implemented into μsik, any one of which can be chosen by the user at runtime initialization via an environment variable specification. Both non-blocking and one-sided GVT algorithms are carefully implemented such that no barriers are invoked from the main loop. All the benchmarks used here to evaluate the GVT algorithm performance are written as applications over μsik.

The implementation and experimentation were performed on a Cray XT5 system with 18,688 nodes, in which each node consists of 2 hex-core AMD Opteron 2435 (Istanbul) 2.6GHz processors and 16GB of memory. The nodes are connected through Cray's SeaStar 2+ 3D torus interconnect. All of the software used in this performance study was compiled with the Portland Group (pgi) compiler version 2.2.73 with `-O3 -fast` compilation flags. All inter-processor event communication is performed using traditional two-sided communication via the MPI. The GVT message exchange for two-sided GVT algorithms is also performed using MPI. Asynchrony with respect to event messaging is realized via `MPI_Iprobe()` for two-sided communication. The synchronous two-sided GVT algorithm uses `MPI_Allreduce()` for the blocking reduction of the transient message counts and local virtual time values. It is also easy to implement because complexities of multiple concurrent epochs are absent due to the fact

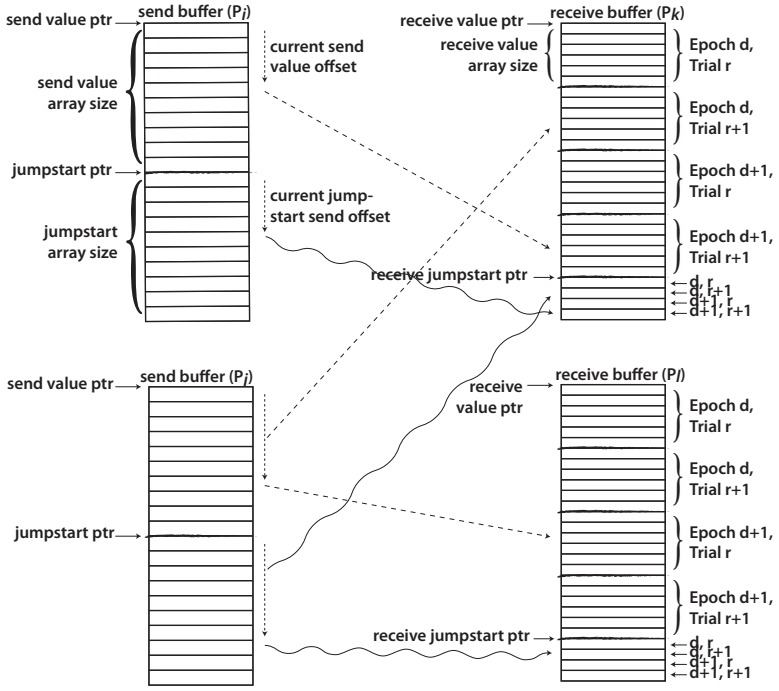


Fig. 3: Data structures for one-sided communication-based GVT. Each in the send or receive buffers includes the $(LVT + LA, nsent - nrecd)$ values of the corresponding sender. Dashed lines represent `put` operations of reduced values, while the squiggly lines represent `put` operations of *jump start* messages to initiate reduction on the receiver side.

that all processors are always at the same epoch number and the same trial number. The asynchronous two-sided GVT algorithm is implemented with user-level reductions performed via the previously described (optimized butterfly) topology, using MPI messaging for exchanging the reduction control messages.

For one-sided GVT algorithms, the Portals interface [Brightwell et al. 2005] is used for one-sided communication. The Portals API on the Cray XT5 is implemented using the Portals Network Access Layer (NAL). The Portals NAL provides a bridge between the Portals API and the SeaStar Network Interface Card (NIC) and utilizes Basic End-to-End Reliability (BEER) protocol for ensuring reliability and performing credit-based flow control. A Direct Memory Access (DMA) program to send/receive data from/to each Portals memory descriptor (MD) is generated in the host and transferred to the SeaStar DMA queue. On SeaStar, message transmission and reception machinery each utilize a single FIFO and a single Direct Memory Access (DMA) queue. Small messages ($<$ 16 bytes) are handled directly from the FIFO while larger messages utilize the DMA Queue. The message transmission machinery on SeaStar has a 32 entry transmit (TX) DMA queue. Elements in the receive engine machinery in SeaStar NIC are: (a) 256 entry Receiver (RX) DMA queue that includes a DMA program in each entry for DMA into a specific memory descriptor, (b) a 256 entry Content Addressable Memory (CAM) table that maps incoming messages to one of the RX DMA queue entry, and (c) an interrupt mechanism that interrupts the host when such a match cannot be found in the CAM table. If a particular host receives messages from over 256 different sources

and continues to do so randomly throughout the life of the program, the number of host interruptions increase significantly. This is detrimental to both the application running at the receiver (as its computations are interrupted) and the sender (due to higher latency incurred by the interrupt).

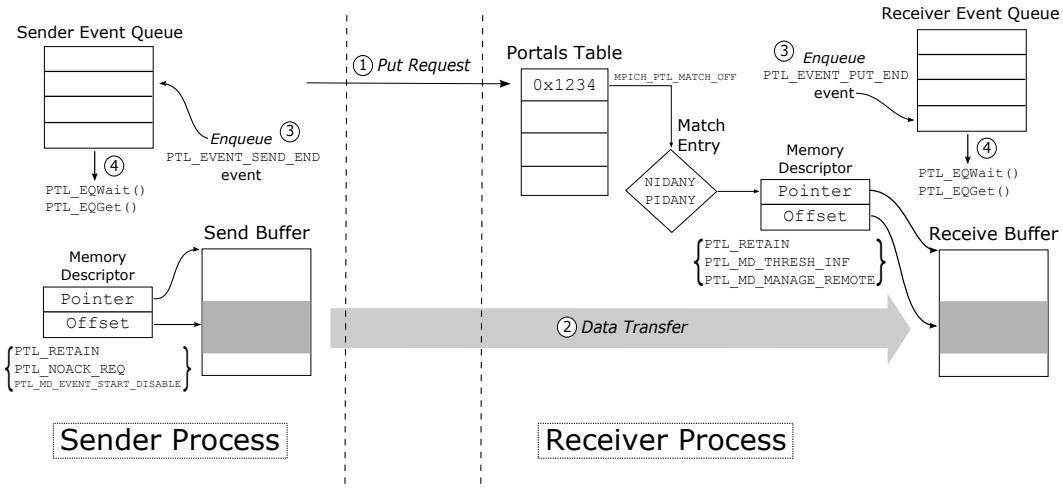


Fig. 4: Portals data structures and data movement operations for one-sided GVT

The functionality of the one-sided GVT algorithm implementation is illustrated in Figure 4. Portals is initialized with memory descriptors (MDs) used for `put` operations configured with infinite threshold, and bound using `PtlMDBind()` with the `PTL_RETAIN` setting to make MDs reusable for later sends. To send, `PtlPutRegion()` is used with `PTL_NOACK_REQ` since acknowledgments for send completions are not needed by our GVT algorithm. For notification of completion of one-sided put operations for GVT messages, we subscribe to the `PTL_EVENT_SEND_END` notification. Similarly, `PTL_EVENT_PUT_END` notification is subscribed to for notification of incoming GVT messages. All destination memory locations of all one-sided puts are managed on the sender-side, and hence, `PTL_MD_MANAGE_REMOTE` is used on all sender-side MDs. The `PTL_EQGet()` and `PTL_EQWait()` calls are used to process all Portals notifications asynchronously. Since the maximum number of outstanding puts are bounded per GVT (epoch), it is possible to select a Portal event queue size such that no notifications would be dropped, and hence `PTL_EQ_DROPPED` would be flagged as an error condition. The MPI option `MPICH_PTL_MATCH_OFF` was used to make MPI perform message matching for the underlying Portals device. In synchronous two-sided operation, we have found that this provides a noticeable performance improvement due to the latency-sensitive nature of PDES applications.

The order in which one-sided communication proceeds in the GVT algorithm is indicated by the circled numbers in Figure 4. The send buffer in this figure corresponds to a send buffer shown on the left side of Figure 3; similarly, the receive buffer in this figure corresponds to a receive buffer shown on the right side of Figure 3. Operations tagged with the same circled number in Figure 4 indicate concurrent operation across the sending and receiving processors.

4. PERFORMANCE ANALYSIS

We examine the dynamics of discrete event execution exercised with the GVT algorithms presented in this work. In order to evaluate the efficacy of each GVT algorithm, we selected four PDES benchmarks which represent a wide cross-section of PDES application characteristics, from varied event densities to mixed messaging and event computation intensities.

4.1. Execution Benchmarks

We use the following four PDES applications that run over μsik, thus automatically inheriting the runtime choice of functionality of all the three GVT algorithm implementations and their optimizations incorporated into μsik.

4.1.1. RCPHOLD. The PHOLD application is a *de facto* standard PDES benchmark commonly used to exercise the underlying simulator's efficiency in event processing, message transmission and reception to destination LPs and, if applicable, rollback efficiency. PHOLD is a synthetic benchmark with little event computation other than random number generation to determine the virtual time increments and destination LPs. PHOLD can be executed conservative mode as well as optimistic mode.

PHOLD can be configured to send to random or a subset of destinations. We define a value, *neighbor reach*, such that a processor only sends to remote processors whose identifiers are within a $\pm$ neighborhood of its own. Events can also be sent to self. Outgoing events are timestamped with a exponentially distributed timestamp with a mean of 1.0 plus lookahead.

The PHOLD benchmark can be configured into specific structures affecting event density and messaging behavior, two of which are used for evaluation. For the present purposes, we denote *structure* as a tuple of (σ, γ), where σ is the number of LPs per core, and γ is a specific parameter for the simulation. For PHOLD, γ is the multiplier for the message population of the simulation. Thus, $\sigma \times \gamma \times \omega$ gives the total message population of the entire simulation across ω cores.

The "RC" moniker of RCPHOLD stands for reverse computation. Instead of storing the state of the simulation prior to each event processed to facilitate rollback in optimistic simulations. When a rollback occurs, the simulator performs a sequence of undo operations that restore the state of the simulation to the proper good state before incorrect events were executed. This is a classic space-time tradeoff where, to roll back the simulation, significant memory savings may be obtained in exchange for some computational overhead.

4.1.2. RCREDIF. Another PDES benchmark used is called RCREDIF [Perumalla and Seal 2011], which is a large-scale epidemiological outbreak simulation based on a reaction-diffusion model. It uses a novel discrete event formulation of the phenomenon, and a new reverse computation-based model as rollback support in its scalable optimistic simulation. Organized in terms of a number of individuals per location (γ), a number of locations per region (σ), and a region per processor, RCREDIF simulates probabilistic transition state machines at the level of each individual within populations Similar to RCPHOLD, RCREDIF also can be executed both in conservative mode as well as optimistic mode using reverse computation, and also employs the *neighbor reach* specification (similar to RCPHOLD) in determining the remote processors selected as potential destinations.

Due to the amount of computation involved in reversing an event, the rollback cost per event is relatively high in RCREDIF. Thus, even if the rollback *length* is small in an RCREDIF simulation run, the total rollback runtime *overhead* can be relatively high.

4.1.3. $\mu\pi$. $\mu\pi$ [Perumalla 2010; Perumalla and Park 2011] is a software-based experimentation platform for testing synthetic and real unmodified MPI programs. $\mu\pi$ multiplexes virtual MPI ranks per real rank (the ratio to be referred to as *LPX*) for execution over simulated virtual platforms through μsik's process-oriented PDES framework.

Barrier Test. The barrier test benchmark aims to stress-test multiple items of interest: (a) ability to instantiate and advance millions of virtual ranks on the simulation time axis, (b) performance under very tight coupling among ranks, especially with regard to stringent characteristics of their virtual interconnection network, and (c) ability for a high level of multiplexing for maximum efficiency. In the benchmark, every rank repeatedly joins a barrier by invoking `MPI_Barrier()`, and querying the time taken by each barrier via the times returned by `MPI_Wtime()`. Also between each pair of barriers, each rank advances simulation time by one millisecond to model a relatively coarse-grained computation.

Ping Test. The ping test benchmark is used to measure bandwidth and latency between pairs of communicating MPI ranks. This ping test has virtual ranks arranged in a naturally-ordered ring topology. The sender sends data to the next higher virtual rank while receiving data from the lower virtual rank. If the virtual rank number is even, it performs a blocking send followed by a blocking receive. The order of operations is reversed for odd-numbered virtual ranks. These operations are timed via calls to `MPI_Wtime()` for bandwidth and latency measurement.

These operations are iterated successively from 8 bytes to the maximum specified test message size, where the length of each message is doubled for each trial until the maximum limit is reached.

4.2. Experiment Setup

The GVT algorithms were tested with all the aforementioned applications. For labels in all of the following charts, we use a 3-tuple `(X Y Z)` to identify the scenario tested, as follows:

- *X* is the synchronization strategy employed: `C` for conservative or `O` for optimistic.
- *Y* denotes usage of a one-sided GVT algorithm (utilizing the Portals interface within μsik time management): `1` for one-sided communication and `2` for two-sided communication.
- *Z* signifies whether or not the synchronous execution is used: `S` for synchronous and `A` for asynchronous execution.

Thus, for example, `(. 2 S)` refers to the conservative or optimistic executions of the two-sided synchronous GVT algorithm, `(O 2 A)` to the optimistic execution of two-sided asynchronous GVT algorithm and `(C 1 A)` to the conservative one-sided asynchronous GVT algorithm.

Combinations of lookahead, structure, synchronization strategy and GVT algorithms were varied for each benchmark. Lookaheads were varied across the RCPHOLD and RCREDIF benchmarks, ranging from low to high values of lookahead. Additionally, the structure (σ, γ) of each application was varied between `(10,1000)` and `(100,100)`. Thus the message population remained constant between structures per core, but the number of LPs per core varied resulting in high and low event densities per LP for the respective scenarios. The number of remote messages, although in total remains the same per core, is an order of magnitude more per LP in the `(10,1000)` case.

For $\mu\pi$ benchmarks, lookahead was fixed based on the network properties. Here we selected a prototypical fast (i.e., latency of 10μs and bandwidth of 1Gb/s) and very fast (i.e., latency of 1μs and bandwidth of 10Gb/s) network specification to determine the

Table I: Notation Used in Charts

	Description
ε	Aggregate committed event rate (millions events/sec)
λ	Number of GVT epochs
ρ	Maximum number of rollbacks observed on a single core
α	Average number of rollbacks per core
F_1	Factor of improvement of asynchronous one-sided GVT algorithm over synchronous two-sided GVT algorithm
F_2	Factor of improvement of asynchronous two-sided GVT algorithm over synchronous two-sided GVT algorithm
LPX	Virtual MPI ranks per real rank

	Description
(σ, γ)	Structure of simulation
`(C . .)`	Conservative simulation
`(O . .)`	Optimistic simulation
`(. 2 S)`	Two-sided synchronous
`(. 2 A)`	Two-sided asynchronous
`(. 1 A)`	One-sided asynchronous

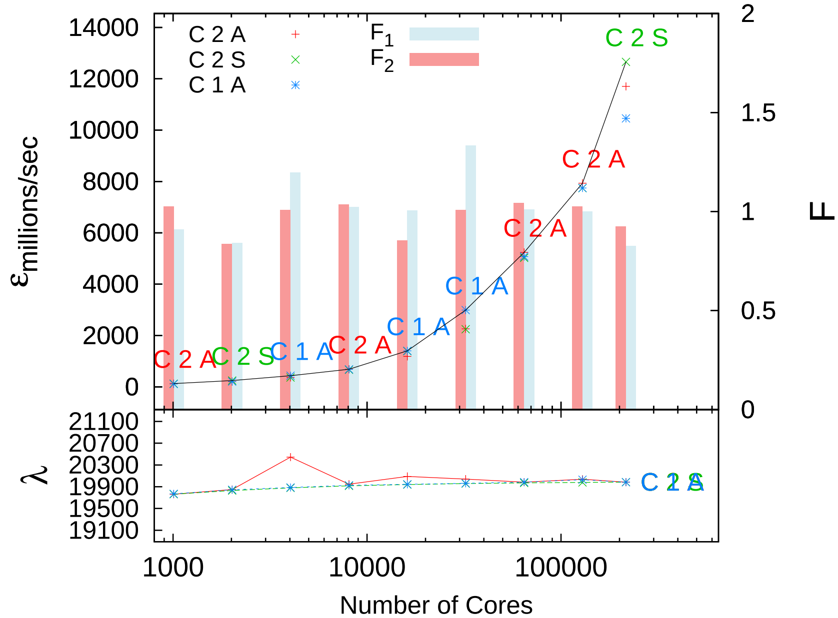
lookahead. The "structure" of the $\mu\pi$ benchmarks is simply *LPX* (i.e., number of virtual MPI ranks multiplexed on each real rank), where values of 128 and 1024 were chosen to showcase light and very heavy multiplexing. An MPI rank is the unique integer assigned to each task (typically, a UNIX process) in a parallel application based on MPI. Each virtual MPI rank is the simulated counterpart of the real MPI rank that appears in the simulated scenario.

The simulation end times were set to 1000 simulated seconds for all RCPHOLD runs, and 168 simulated hours (1 week of disease spread=24 hours/day $\times$ 7 days) for all RCREDIF runs. $\mu\pi$ barrier test simulated one virtual barrier for all *LPX*, while $\mu\pi$ ping test simulated up to 1KiB and 64KiB of data transfer in the *LPX*=1024 and *LPX*=128 structures, respectively.

For the following charts, ε denotes the aggregate committed event rate in millions of events/sec which is plotted on the primary ordinate. Each individual data point for the three GVT algorithms tested is plotted while the best performing GVT algorithm (i.e., the algorithm achieving the highest ε) is noted at each core count. A line joining the maxima is drawn through each of these best-performing numbers to visually show a trendline of performance as the simulation is scaled.

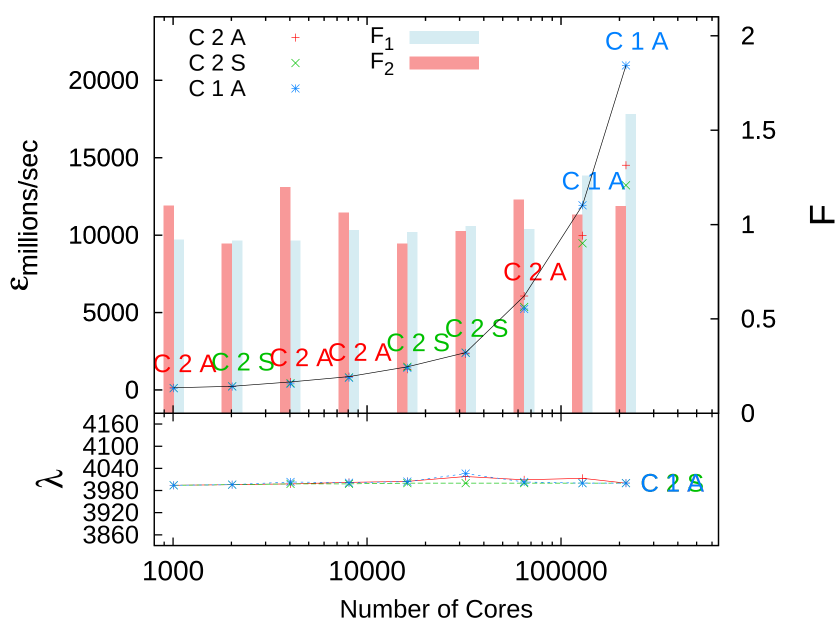
Additionally, charts include F bars which denote the factor of improvement over the most commonly used PDES implementations, namely, the `(. 2 S)`, which is the synchronous two-sided GVT algorithm. This factor is plotted for the asynchronous GVT algorithms *i.e.*, `(. 2 A)` and `(. 1 A)` on the secondary ordinate. Secondary plots on the following charts may include λ, which denotes the number of GVT epochs, ρ, which denotes the maximum number of rollbacks occurring on a single core within the entire simulation for selected optimistic executions and α which signifies the average number of rollbacks per LP. The symbols and notations are summarized in Table I.

Every scenario is scaled up to at least 129,024 cores. Some executions were scaled to 216,000 cores (not all could be executed at the full scale of 216,000 cores, due to constraints on the number of hours allocated to us on the machine).



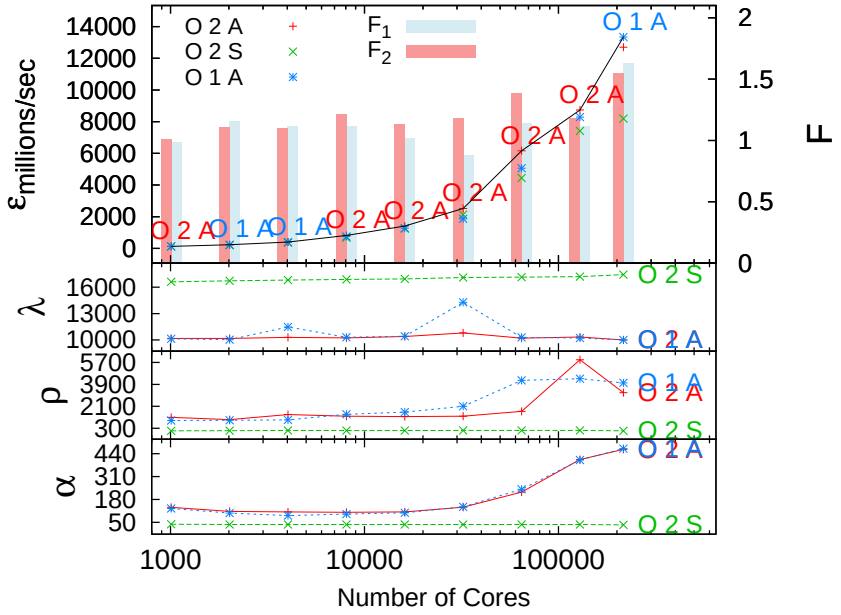
(a) LA=0.1, (10,1000)

Fig. 5: RCPHOLD Conservative Synchronization, Low Lookahead



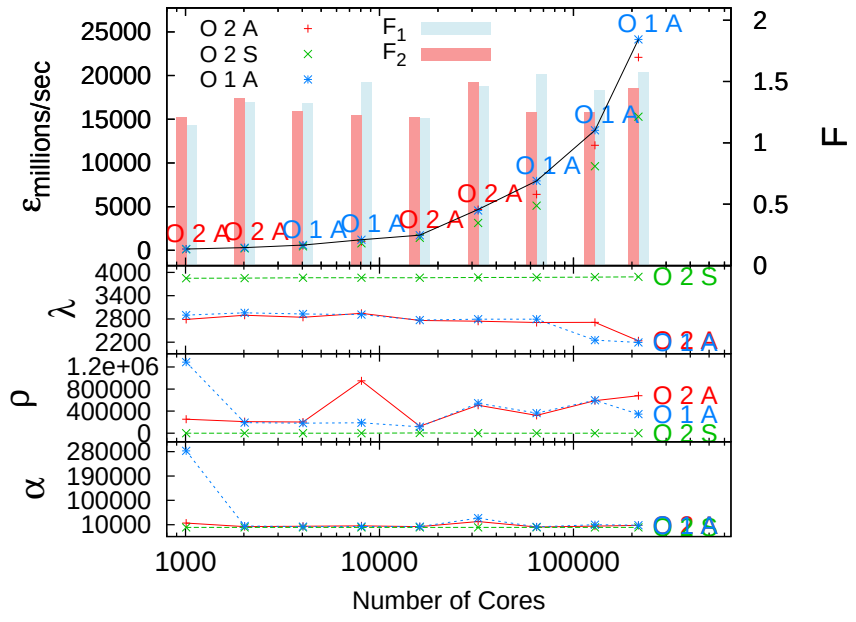
(a) LA=0.5, (10,1000)

Fig. 6: RCPHOLD Conservative Synchronization, High Lookahead



(a) LA=0.1, (10,1000)

Fig. 7: RCPHOLD Optimistic Synchronization, Low Lookahead



(a) LA=0.5, (10,1000)

Fig. 8: RCPHOLD Optimistic Synchronization, High Lookahead

4.3. RCPHOLD Results

Conservatively synchronized RCPHOLD benchmarks at low lookahead of 0.1 shown in Figure 5a (and Figure 14a in the supplement) scaled to 216,000 cores, we observe in nearly all cases that asynchronous GVT algorithms performed no worse than synchronous two-sided GVT, and sometimes provided significant improvements in ε – up to 1.5× in the structure of `(100,100)`. Interestingly, λ remained nearly the same for all GVT algorithms and holds through all tested core counts. Thus, the runtime difference and performance improvement shown by both asynchronous GVT algorithms is resulting from decrease in wall-clock time consumed by these algorithms per GVT computation.

At a high lookahead of 0.5 shown in Figure 6a (and Figure 15a in the supplement) with data to 216,000 cores, we observe similar performance from both synchronous and asynchronous GVT algorithms at smaller scales. As the simulation is scaled to 32K processor cores and beyond, asynchronous GVT algorithms tend to cope with larger number of processor cores better as ε improvements exceeding 1.5× is observed in the `(100,100)` case. We can reason here that the increased amount of lookahead lowers the total amount of synchronization burden across the entire simulation which becomes increasingly more taxing as the simulation is spread across more processor cores. A 5× decrease is observed in λ, which is correlated with 5× increase in the amount of lookahead, compared to the low lookahead case of 0.1, as expected.

Optimistically synchronized RCPHOLD provides further insight into how the speed, behavior and quality of information delivered by the underlying GVT algorithms can significantly impact the performance of this particular PDES benchmark.

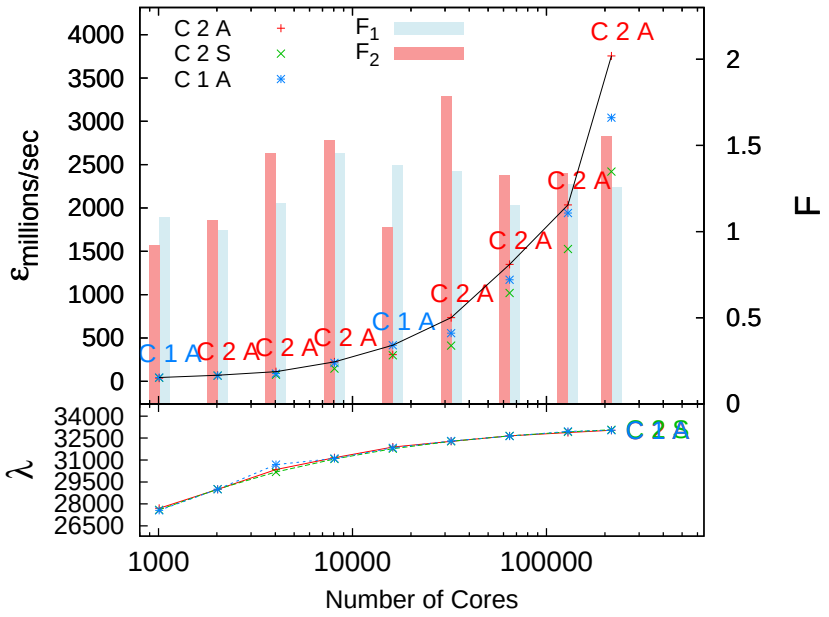
In the RCPHOLD optimistic scenarios shown in Figure 7 and Figure 8 we see that, in nearly all cases, both asynchronous GVT algorithms provide at least the performance of synchronous two-sided GVT algorithm, but can often accelerate the simulation much faster showing, consistently 1.2× to over 1.5× the performance of synchronous two-sided GVT, especially in the `(10,1000)` structure cases.

The striking detail that comes forth through all of the RCPHOLD optimistic charts is the significant difference in λ for the synchronous two-sided GVT algorithm. Frequent GVT computation is not necessarily a detriment to overall performance. In fact, having fresh GVT information can reduce the number of potential incorrect events processed (and thus the number of rollbacks) in an optimistic parallel simulation. However, this generalization only holds if the cost of the GVT computation is relatively inexpensive compared to the cost of rollback. Since RCPHOLD is a synthetic benchmark that is not computationally intensive, rollback costs are relatively inexpensive. *Thus, for RCPHOLD, the rollback cost is significantly less than GVT computation cost.*

We can clearly observe these dynamics in RCPHOLD. λ is more than 1.5× in the synchronous two-sided GVT cases over both the asynchronous cases, yet there are no rollbacks in the synchronous two-sided GVT cases while there are rollbacks present in the both asynchronous cases. We see that in certain cases, such as shown in `(10,1000)` structure in Figure 7a and Figure 21a, at larger core counts, the ε gap widens as the cost per λ increases with the number cores. It is clear here that the quality of the GVT information delivered by both asynchronous GVT algorithms is no less than, if not better than, that of the synchronous two-sided GVT algorithm. Thus, both asynchronous GVT algorithms incur less overhead through expedited GVT computations in addition to smaller λ in the scenarios with lookahead greater than 0.01.

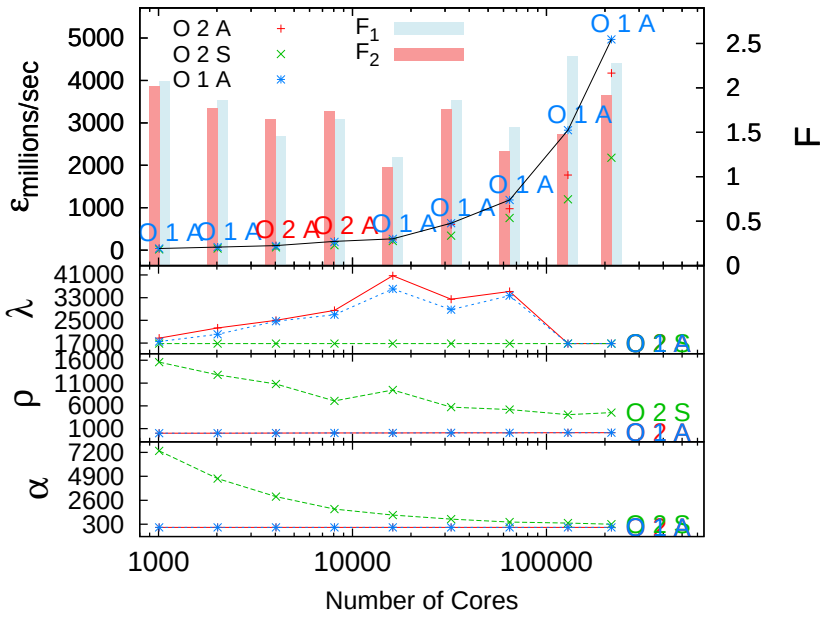
4.4. RCREDIF Results

RCREDIF with low lookahead of 0.01 is shown in Figure 9a and Figure 22a with data up to 216,000 cores. Similar to RCPHOLD, we observe that both asynchronous GVT



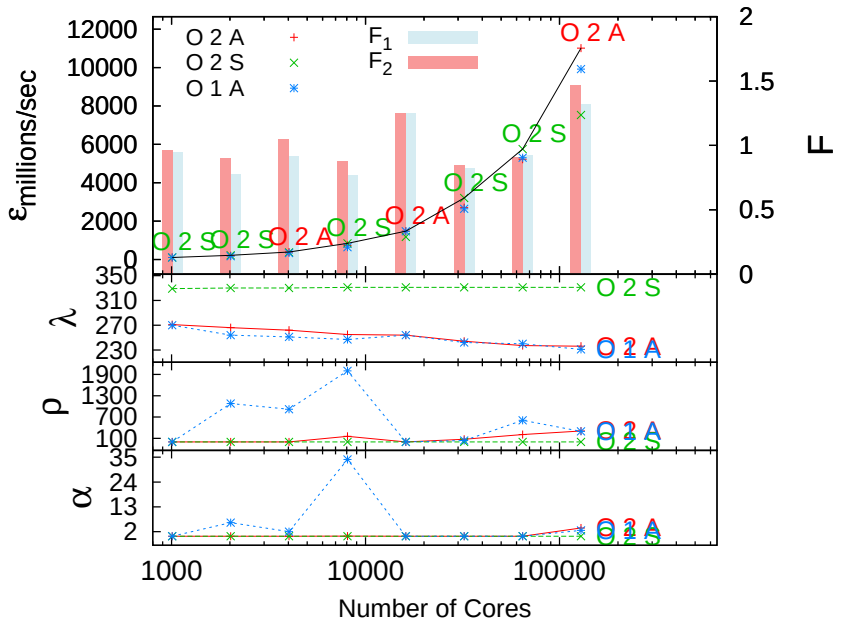
(a) LA=0.01, (10,1000)

Fig. 9: RCREDIF Conservative Synchronization, Low Lookahead



(a) LA=0.01, (10,1000)

Fig. 10: RCREDIF Optimistic Synchronization, Low Lookahead



(a) LA=1, (10,1000)

Fig. 11: RCREDIF Optimistic Synchronization, High Lookahead

algorithms provide better performance than the synchronous two-sided GVT algorithm at all core counts. λ remains nearly identical for all GVT algorithms as the RCREDIF application is scaled out. Clearly, the cost per λ is the differentiating determinant for ε and thus, F. Scenarios at lookahead of 0.1 shown in Figure 25a and Figure 25b (scaled to 129,024 cores) follow similar trends to the lookahead cases of 0.1. In the structure of `(10,1000)` the synchronous two-sided GVT algorithm slightly edges the asynchronous GVT algorithms at up to 32,256 cores. After this point, the asynchronous GVT algorithms pull ahead in a rather extraordinary fashion. In fact, in these low lookahead scenarios in Figure 25 the differences between GVT algorithms is minimal until a certain amount of scale is reached. After such a scaling point, the performance difference in ε is significant as we can observe improvements in F exceeding a factor of 1.5$\times$ up to nearly 2.5$\times$ at the largest scales tested.

In contrast to the RCPHOLD synthetic benchmark, RCREDIF is a non-synthetic application that uses significant computation per event. Thus, rollbacks are expensive in comparison to those found in RCPHOLD. At a low lookahead of 0.01 under optimistic synchronization for RCREDIF shown in Figure 10a and Figure 23a scaled to 216,000 cores, we observe that ρ and α for the synchronous two-sided GVT algorithm is significant while both asynchronous GVT algorithms very little rollback. This inversely correlates with ε where we observe 1.2$\times$ to nearly 3$\times$ F. In these low lookahead scenarios, λ tends to be very large (i.e., 16K to 30K+ computations) over the course of the execution. The speed and frequency of the GVT algorithm comes into play for these small lookaheads. Both asynchronous GVT algorithms exhibit larger λ, providing more up-to-date GVT information without sacrificing the simulation speed of event computation. The synchronous two-sided GVT algorithm on the other hand synchronizes less frequently, up to nearly 50% less, yet performs significantly worse at scale. The speed of the GVT algorithm clearly impacts the event execution dynamics: as potentially

more incorrect events are executed they must ultimately be rolled back, incurring significant synchronization overhead cost. This is compounded on top of higher rollback costs in RCREDIF.

We observe that α decreases steadily as the simulation scales in lookahead cases of 0.01, yet does not impact the performance gap with respect to ε compared to the asynchronous GVT algorithms, the one-sided approach in particular. The combination of a slower GVT computation speed and presence of larger ρ lead to the ε difference. It is also interesting to note that λ decreases for the asynchronous GVT algorithms while maintaining very good rollback characteristics resulting in very little overhead. The F observed in some of the low lookahead scenarios shown in Figure 27 emphasizes the potential performance gain that can be achieved by exploiting lower layer communication interfaces such as one-sided Portals.

Figure 12b shows speedup for the best and worst case committed event rates for RCREDIF. The best and worst observed ε for RCREDIF are both using conservative synchronization at lookahead of 1 with a structure of `(100, 100)` and lookahead of 0.01 with a structure of `(10, 1000)`, respectively.

4.5. $\mu\pi$ Results

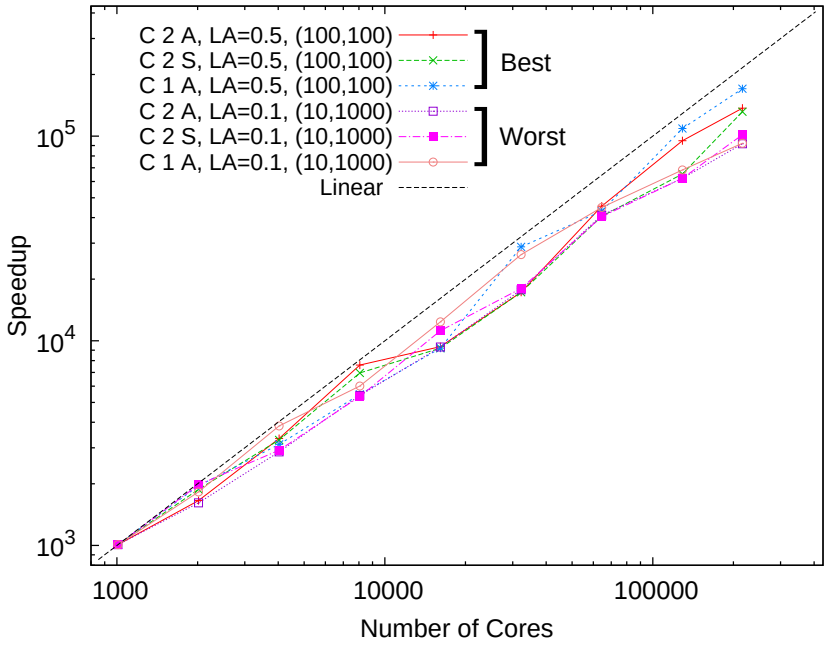
In the $\mu\pi$ benchmarks as shown in Figure 13, the performance difference between synchronous two-sided GVT algorithms and asynchronous GVT algorithms are clearly pronounced. Here, λ is approximately equal for all scenarios, thus indicating that the time to complete GVT computations in the synchronous two-sided GVT algorithm is significantly longer than both asynchronous GVT cases. We observe a 1.67$\times$ performance improvement in runtime for the asynchronous one-sided GVT algorithm over the synchronous two-sided GVT algorithm at 216,000 processor cores simulating over 221 million virtual MPI ranks in the *LPX*=1024 case for barrier test as shown in Figure 13a. Similarly for the ping test shown in Figure 13b, there is a 2.07$\times$ improvement in runtime for the asynchronous one-sided GVT algorithm over the synchronous two-sided GVT algorithm at the same scale for *LPX*=1024. For the lightly multiplexed cases of *LPX*=128, the runtime performance differential between synchronous and asynchronous GVT algorithms begins to appear at scale when the number of cores exceeds approximately 8K.

The difference in performance between GVT algorithms can be attributed to the time-slicing nature of process-oriented PDES where multiple virtual threads are multiplexed on top of a single real execution thread of the main loop. As the number of virtual contexts are increased, the proportional amount of time taken by the GVT algorithm becomes larger in relation to the amount of time given per context switch to each virtual thread. Thus, the effects of a slower, synchronous GVT algorithm becomes apparent on high multiplexing counts.

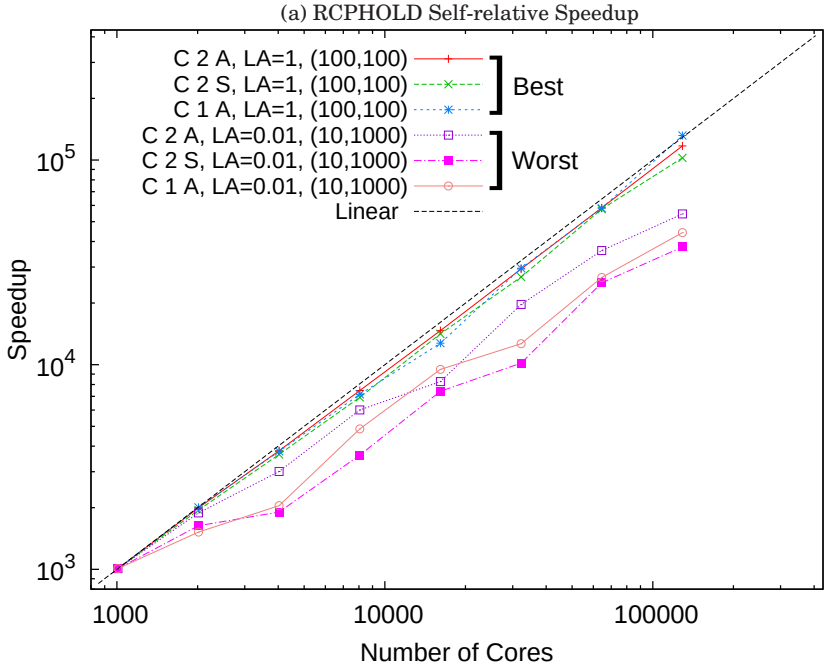
4.6. Performance Summary

A cost trade-off is observed between GVT frequency and rollbacks. Executions which incur higher per-event rollback costs can benefit from more frequent GVT computations. GVT algorithms that complete faster, such as in the asynchronous approaches, can lead to significant performance gains by minimizing rollbacks at the relatively smaller expense of more frequent GVT computations. Also, the cost per rollback does not remain constant as executions scale. RCREDIF shows that, as an execution is scaled out, the GVT algorithms which lead to little or no rollback provides significant gains in simulation performance.

Asynchronous GVT algorithms almost always performed at least as well as their synchronous counterpart. In the majority of cases, the asynchronous GVT algorithms accelerate the execution by spending less time in synchronization overheads. The



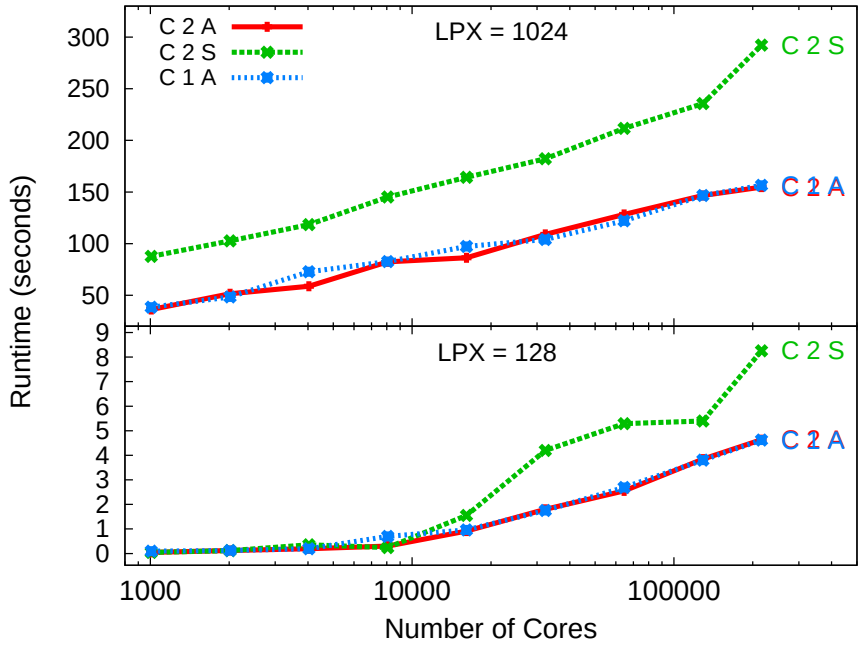
(a) RCPHOLD Self-relative Speedup



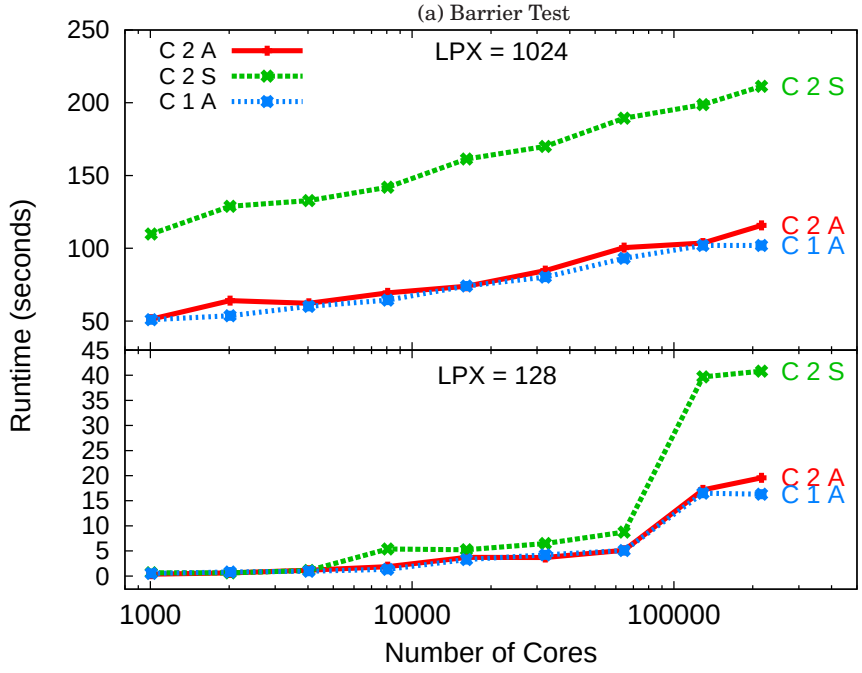
(b) RCREDIF Self-relative Speedup

Fig. 12: Self-relative Speedups for Best and Worst Committed Event Rate Trends

notable exception to this rule is observed with optimistic execution at high lookahead, where the synchronous two-sided GVT algorithm tends to synchronize more fre-



(a) Barrier Test



(b) Ping Test

Fig. 13: $\mu\pi$ Runtime Performance

quently, in effect, reducing the staggered nature of execution and rolled-back event

computation. However, this exception only seems to be limited to executions with less than 10^5 cores.

Process-oriented discrete event execution (exercised with $\mu\pi$), which time-multiplex multiple thread contexts on a single core, are seen to benefit the most from asynchronous GVT algorithms at large multiplexing levels and/or at larger core counts. As the amount of processor time per thread becomes more scarce at higher multiplexing counts, the relative amount of time spent in GVT rises. Thus, the asynchronous nature of GVT algorithm is beneficial in allowing events to be processed asynchronously with GVT computation.

In the fastest RCPHOLD runs, the event rate was nearly 54 billion events per second on 216,000 cores. The best self-relative speedup for RCPHOLD exceeded 170,600 on 216,000 cores. The best self-relative speedup for RCREDIF exceeded 131,790 on 129,024 cores (superlinearity attributable to a suboptimal, cache-limited performance of the baseline run on 1008 cores). In the largest runs of RCREDIF on 216,000 cores, over 2.1 billion individuals were simulated. With $\mu\pi$, a total of 221,184,000 virtual MPI ranks were simulated, representing the largest case of process-oriented discrete event mode of execution to date. In the barrier test of $\mu\pi$, one-sided GVT gave $1.67\times$ faster execution than blocking GVT. The ping test of $\mu\pi$ executed over $2.07\times$ faster with one-sided GVT than blocking GVT.

5. RELATED WORK

Research in efficient virtual time synchronization experienced rich development over the past few decades. In the early 1990s, after parallel/distributed discrete event simulation began gaining sufficient attention in the research community, a spurt of synchronization algorithms were proposed, for use in conservative schemes, optimistic schemes, or both. A rich variety of parameters were considered, such as the presence or absence of first-in-first-out (FIFO) guarantees, message send-back protocols, virtual time horizons and windows, message acknowledgments, and different network architectures or topologies such as token rings and trees. These advancements included seminal works such as the Chandy Misra Bryant (null message) algorithm [Chandy and Misra 1981], the Time Warp algorithm [Jefferson and Sowzral 1982; Jefferson 1985], Samadi's algorithm [Samadi 1985] based on message acknowledgments, Mattern's algorithm [Mattern 1993] based on distributed snap shots, Lin-Lazowska algorithm [Lin and Lazowska 1990] based on combination of time stamped histories and message sequence numbers, and the Bauer-Sporrer [Bauer and Sporrer 1992] algorithm based on centralized resolution on FIFO network channels, and many other variants (e.g., [Su and Seitz 1989; Steinman 1992; Reiher et al. 1990; Preiss et al. 1991; Nicol 1993a; 1993b; Lipton and Mizell 1990; Konas and Yew 1992; Felderman and Kleinrock 1991; DeVries 1990]). The majority of them were analyzed for theoretical correctness of operation and analytical complexity measures such as latency between successive time advances [Gomes et al. 1998; Fujimoto 1999].

Additional innovative directions, such as hardware-assisted and shared memory-optimized algorithms, were pursued in early and mid 1990s. Reynolds et al. [Reynolds et al. 1993] proposed hardware support for GVT computations, using a specialized co-processor style of interface to central processing units to offload reduction operations onto the custom-designed co-processors and their interconnection network. Perhaps due to the hardware limitations of their times, performance study was only limited to a few dozen processors. Rosu et al. [Rosu et al. 1997] proposed offloading GVT computations to programmable, general-purpose network cards, again limited in scale by the network hardware technologies of the time. Fujimoto and Hybinette [Fujimoto and Hybinette 1997] developed a fast algorithm for GVT computation optimized for shared

memory machines, carefully avoiding any locking (semaphore) costs among concurrent threads of execution.

Later, in the late 1990s and early 2000s, synchronization gained additional attention largely in the context of the Time Management interface of the High Level Architecture, and in the context of large-scale computer network simulations. The former saw real-time (receive ordered) executions scaled to 10^3 processors, but time-constrained executions were limited to 10^2 processors using centralized run time infrastructures. The latter saw conservative time synchronization advanced to the 10^3-processor scale, using global reductions-based algorithms [Fujimoto et al. 2003] as well as null message variants exploiting locality of communication [Park et al. 2004]. Also, virtual time algorithms for operation over combinations of unreliable and reliable network channels for events and synchronization messages were developed [Perumalla and Fujimoto 2001] and applied to both HLA simulations as well as network simulations.

In mid 2000s, research continued, attempting further scaling [Chen and Szymanski 2005; 2008] using a centralized algorithm on up to 1000 processor cores. Also, ways to combine real time with virtual time advances were explored [McLean et al. 2004; Bauer Jr. et al. 2005]. A GVT algorithm based on "network atomic operations" was designed by Bauer et al. [Bauer Jr. et al. 2005], and its performance was evaluated on a small scale of 16 processors.

In the late 2000s, a marked shift occurred, catching up to the dramatically increasing sizes of the supercomputing installations that began to emerge with 10^4 processor cores for the first time. The focus of some of the GVT algorithm design and implementation, as a result, shifted to scalability, aimed at sustaining discrete event execution on some of the largest computing installations. Feasibility of executing conservative, optimistic and mixed-mode executions on up to 32,768 processor cores was first realized on Blue Gene/L platforms [Perumalla 2007], followed by speed and efficiency improvements to optimistic execution on Blue Gene/L [Holder and Carothers 2008; Bauer Jr. et al. 2009]. Additional analyses of the event dynamics between conservative and optimistic execution were compared with synthetic benchmarks on 16,384 Blue Gene/L cores [Carothers and Perumalla 2010]. In the line of evolution, the next largest supercomputing installations emerged, containing on the order of 10^5–10^6 cores. Execution of PDES at this scale is the focus of this paper (a preliminary version of this work appeared in [Perumalla et al. 2011]).

6. CONCLUSIONS AND FUTURE WORK

The feasibility of executing discrete event execution, both in conservative and optimistic modes, on some of the largest parallel processing platforms has been shown here. Using a comprehensive experimental study that covers a variety of PDES performance parameters on dissimilar applications, detailed quantitative performance data has been documented on up to 216,000 processor cores of a Cray XT5 system. Performance of nearly 54 billion events per wall clock second has been logged in some of the fastest runs. The performance data can be helpful to both users and researchers of discrete event execution in terms of the scales and speeds that can be expected for different combinations of modeling and system parameters. The experiment results advance the levels of PDES capabilities on massively parallel platforms reported in the literature. A new GVT algorithm based on one-sided communication has been proposed and studied at scale for the first time here, as one of three variants in the space of synchronous *vs.* asynchronous computation and one-sided *vs.* two-sided communication. The results highlight PDES as a potential candidate in the class of supercomputing applications, whose benefits can be explored in various domains such as Internet simulations, vehicular transportation simulations, and social behavioral simulations.

Integration of large-scale agent-based simulation with discrete event execution is part of our future work, along with evaluation with more discrete event applications. Porting and optimizing to the next generation of network interconnects (such as the Gemini and Aries lines of networks in multi-petascale Cray systems) remains to be explored. Also of immediate interest is the exploration of the interplay with accelerator technologies that are being incorporated into the largest supercomputing platforms. It would also be interesting to explore the use of one-sided communication for event messaging as well, complementing that for GVT algorithm communication.

REFERENCES

ALMÁSI, G., HEIDELBERGER, P., ARCHER, C. J., MARTORELL, X., ERWAY, C. C., MOREIRA, J. E., STEINMACHER-BUROW, B., AND ZHENG, Y. 2005. Optimization of mpi collective communication on bluegene/l systems. In *Annual International Conference on Supercomputing*. ACM, 253–262.

BAUER, H. AND SPORRER, C. 1992. Distributed logic simulation and an approach to asynchronous gvt-calculation. In *Workshop on Parallel and Distributed Simulation*. IEEE Computer Society.

BAUER JR., D. W., CAROTHERS, C. D., AND HOLDER, A. 2009. Scalable time warp on blue gene supercomputers. In *Workshop on Principles of Advanced and Distributed Simulation*. 35–44.

BAUER JR., D. W., YAUN, G., CAROTHERS, C. D., YUKSEL, M., AND KALYANARAMAN, S. 2005. Seven-o-clock: A new distributed gvt algorithm using network atomic operations. In *Workshop on Principles of Advanced and Distributed Simulation*. IEEE Computer Society.

BRIGHTWELL, R., HUDSON, T., PEDRETTI, K., RIESEN, R., AND UNDERWOOD, K. 2005. Implementation and performance of portals 3.3 on the cray xt3. In *Cluster Computing*. IEEE Computer Society, 1–10.

CAROTHERS, C. D. AND PERUMALLA, K. S. 2010. On deciding between conservative and optimistic approaches on massively parallel platforms. In *Winter Simulation Conference*. IEEE Computer Society, 678–687.

CHANDY, K. M. AND MISRA, J. 1981. Asynchronous distributed simulation via a sequence of parallel computations. *Communications fo the ACM 24,* 4, 198–205.

CHEN, D. AND SZYMANSKI, B. K. 2005. Dsim: Scaling time warp to 1,033 processors. In *Winter Simulation Conference*. IEEE, Orlando, FL.

CHEN, G. G. AND SZYMANSKI, B. K. 2008. Time quantum gvt: A scalable computation of the global virtual time in parallel discrete event simulations. *Scalable Computing: Practice and Experience 8,* 4, 423–435.

CHOE, M. AND TROPPER, C. 1998. An efficient gvt computation using snapshots. In *In Proceedings of the Conference on Computer Simulation Methods and Applications, The Society for the Computer Simulation*. 33–43.

DEVRIES, R. C. 1990. Reducing null messages in misra's distributed discrete event simulation method. *IEEE Transactions on Software Engineering 16,* 1, 82–91. january 1990.

FARAJ, A., KUMAR, S., SMITH, B., MAMIDALA, A., AND GUNNELS, J. 2009. Mpi collective communications on the blue gene/p supercomputer: Algorithms and optimizations. In *High Performance Interconnects. IEEE Symposium on*. 63–72.

FELDERMAN, R. AND KLEINROCK, L. 1991. Two processor time warp analysis: Some results on a unifying approach. In *Advances in Parallel and Distributed Simulation*. SCS Simulation Series, vol. 23. 3–10.

FUJIMOTO, R. M. 1999. *Parallel and Distributed Simulation Systems* 1st Ed. John Wiley & Sons, Inc., New York, NY, USA.

FUJIMOTO, R. M. AND HYBINETTE, M. 1997. Computing global virtual time in shared memory multiprocessors. *ACM Transactions on Modeling and Computer Simulation 7,* 4, 425–446.

FUJIMOTO, R. M., PERUMALLA, K. S., WU, H., AMMAR, M., AND RILEY, G. F. 2003. Large-scale network simulation: How big? how fast? In *International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. IEEE.

GOMES, F., CLEARY, J., AND UNGER, B. 1998. A survey of gvt algorithms. Technical Report pages.cpsc.ucalgary.ca/∼gomes/PAPERS/GVT.ps, University of Calgary, Canada.

HOLDER, A. O. AND CAROTHERS, C. D. 2008. Analysis of time warp on a 32,768 processor ibm blue gene/l supercomputer. In *2008 Proceedings European Modeling and Simulation Symposium (EMSS)*.

JEFFERSON, D. R. 1985. Virtual time. *ACM Transactions on Programming Languages and Systems 7*, 404–425.

JEFFERSON, D. R. AND SOWZRAL, H. 1982. Fast concurrent simulation using the time warp mechanism, part i: Local control. Technical Report to the United States Air Force N-1906-AF, Rand Corporation, Santa Monica, CA, USA.

KONAS, P. AND YEW, P.-C. 1992. Synchronous parallel discrete event simulation on shared memory multiprocessors. In *Workshop on Parallel and Distributed Simulation*. Vol. 24. SCS Simulation Series, 12–21.

LIN, Y.-B. AND LAZOWSKA, E. 1990. Determining the global virtual time in a distributed simulation. In *Proceedings of the International Conference on Parallel Processing III*. 201–209.

LIPTON, R. J. AND MIZELL, D. W. 1990. Time warp vs. chandy-misra: A worst-case comparison. In *Proceedings of the SCS Multiconference on Distributed Simulation*. SCS Simulation Series, vol. 22. 137–143.

MATTERN, F. 1993. Efficient algorithms for distributed snapshots and global virtual time approximation. *J. Parallel Distrib. Comput. 18*, 423–434.

MCLEAN, T., FUJIMOTO, R., AND FITZGIBBONS, B. 2004. Middleware for real-time distributed simulations. *Concurrency Computation: Practice and Experience 16*, 1483–1501.

NICOL, D. 1993a. Global synchronization for optimistic parallel discrete event simulation. In *Workshop on Parallel and Distributed Simulation*. IEEE Computer Society, San Diego, CA, USA.

NICOL, D. M. 1993b. The cost of conservative synchronization in parallel discrete event simulations. *Journal of the Association for Computing Machinery 40,* 2, 304–333.

PARK, A. J., FUJIMOTO, R. M., AND PERUMALLA, K. S. 2004. Conservative synchronization of large-scale network simulations. In *Workshop on Parallel and Distributed Simulation*. IEEE Computer Society.

PERUMALLA, K. AND FUJIMOTO, R. 2001. Virtual time synchronization over unreliable network transport. In *Workshop on Parallel and Distributed Simulation*. IEEE Computer Society, 129–136.

PERUMALLA, K. S. 2005. μsik: A micro-kernel for parallel/distributed simulation systems. In *Workshop on Principles of Advanced and Distributed Simulation*. IEEE Computer Society, 59–68.

PERUMALLA, K. S. 2007. Scaling time warp-based discrete event execution to 10^4 processors on the blue gene supercomputer. In *International Conference on Computing Frontiers*. Ischia, Italy, 69–76.

PERUMALLA, K. S. 2010. $\mu\pi$: A scalable and transparent system for simulating mpi programs. In *Proceedings of the 3rd International Conference on SIMUTools*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

PERUMALLA, K. S. AND PARK, A. J. 2011. Improving multi-million virtual rank execution in $\mu\pi$. In *International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. IEEE, 454–457.

PERUMALLA, K. S., PARK, A. J., AND TIPPARAJU, V. 2011. Gvt algorithms and discrete event dynamics on 129k+ processor cores. In *International Conference on High Performance Computing*. IEEE.

PERUMALLA, K. S. AND SEAL, S. K. 2011. Discrete event modeling and massively parallel execution of epidemic outbreak phenomena. *Transactions of the Society for Modeling and Simulation International in print*, doi:10.1177/0037549711413001.

PREISS, B., LOUCKS, W., MACINTYRE, I., AND FIELD, J. 1991. Null message cancellation in conservative distributed simulation. In *Advances in Parallel and Distributed Simulation*. Vol. 23. SCS Simulation Series, 33–38.

REIHER, P., WIELAND, F., AND HONTALAS, P. 1990. Providing determinism in the time warp operating system -costs, benefits, and implications. In *Proceedings of the Workshop on Experimental Distributed Systems*. IEEE, Huntsville, Alabama, 113–118.

REYNOLDS, PAUL, J., PANCERELLA, C., AND SRINIVASAN, S. 1993. Design and performance analysis of hardware support for parallel simulations. *J. of Parallel and Distributed Computing 18,* 4, 435–453.

ROSU, M., SCHWAN, K., AND FUJIMOTO, R. M. 1997. Supporting parallel applications on clusters of workstations: The intelligent network interface approach. In *IEEE Symposium on High Performance Distributed Computing*.

SAMADI, B. 1985. Distributed simulation, algorithms and performance analysis. Ph.D. thesis, Department of Computer Science, University of California, Los Angeles.

STEINMAN, J. S. 1992. Speedes: A multiple-synchronization environment for parallel discrete event simulation. *International Journal on Computer Simulation*, 251–286.

SU, W. K. AND SEITZ, C. L. 1989. Variants of the chandy-misra-bryant distributed discrete event simulation algorithm. In *Proceedings of the SCS Multiconference on Distributed Simulation*. SCS Simulation Series Series, vol. 21. Society for Computer Simulation, 38–43.

7. SUPPLEMENTAL PERFORMANCE STUDY DATA AND ANALYSIS

7.1. Additional RCPHOLD Results

At very low lookahead of 0.01 for conservative synchronization, the simulation is scaled up to 16,128 processor cores. For structure of `(10,1000)` shown in Figure 18a, the synchronous two-sided GVT algorithm performs nearly the same as the asynchronous variants. However, even as soon as reaching the 4,032-core mark, the asynchronous GVT algorithms begin to pull ahead while λ remains relatively constant. We can deduce that the amount of time to complete GVT computations is the primary differentiating factor, resulting in improved ε. In the `(100,100)` structure scenario shown in Figure 18b, we see a similar trend appear at 8,064 cores.

At a very high lookahead of 1 shown in Figure 19a and Figure 19b scaled to 16,128 cores, both asynchronous GVT algorithms outperform the synchronous two-sided GVT algorithm in nearly all cases up to 16,128 processor cores. In the `(100,100)` structure case, the δ performance improvement is on average much higher due to the smaller event densities and spare event computation compared to the `(10,1000)` case. With less local events to process per LP, the relative cost in the GVT computation is higher. Thus the speed in the GVT computation can significantly improve the overall performance and ε rates of the simulation for not only the LA of 1 case, but for all structure `(100,100)` cases for RCPHOLD.

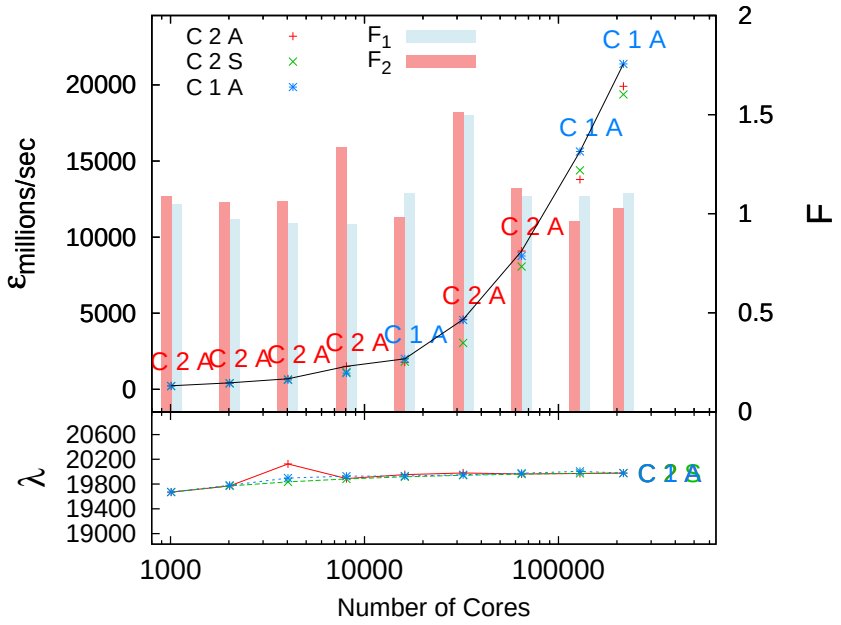
For optimistic simulation, in the low lookahead of 0.01 with structure of `(10,1000)` scenario shown in Figure 20a, the synchronous two-sided GVT algorithm outperforms both asynchronous GVT algorithms for up to 16,128 cores. For this scenario, λ is $1.5\times$ to $3\times$ lower than that of the asynchronous GVT algorithms with very little difference in ρ and α. Thus the significant increase in λ overhead translates to poorer performance by the asynchronous GVT algorithms for this scenario. In the `(100,100)` structure case shown in Figure 20b, we observe that the asynchronous GVT algorithms slightly edge the synchronous two-sided GVT algorithm although λ is lower for the synchronous two-sided GVT algorithm. However, in this scenario, the difference is not as significant as the `(10,1000)` structure case with the asynchronous two-sided GVT algorithm incurring nearly $2\times$ the amount of λ at in the worst cases. The reason for the asynchronous one-sided GVT algorithm exhibiting better ε yet with higher λ can be attributed to the speed of the GVT algorithm compared to the synchronous two-sided algorithm.

It is interesting to note in Figure 16a, where machine conditions are nearly perfect and staggering of LP execution is minimized, we see a significant performance gain of ε for the asynchronous GVT algorithms. At the final data points at 216,000 cores, we observe low amount of λ, nearly zero ρ and a non-increasing amount of α from 129,024 cores results in the highest F gain for any lookahead at structure of `(100,100)` at scale.

Figure 12a shows the overall speedup trends for the best and worst committed event rate trends among scenarios of lookahead 0.1 and 0.5. Speedup is measured over the base of 1,008 cores for their respective GVT algorithm (i.e., self-relative speedup). The best and worst observed ε for RCPHOLD are both using conservative synchronization at lookahead of 0.5 with a structure of `(100, 100)` and lookahead of 0.1 with a structure of `(10, 1000)`, respectively. For most data points, the asynchronous GVT algorithms outperform the synchronous two-sided GVT algorithm.

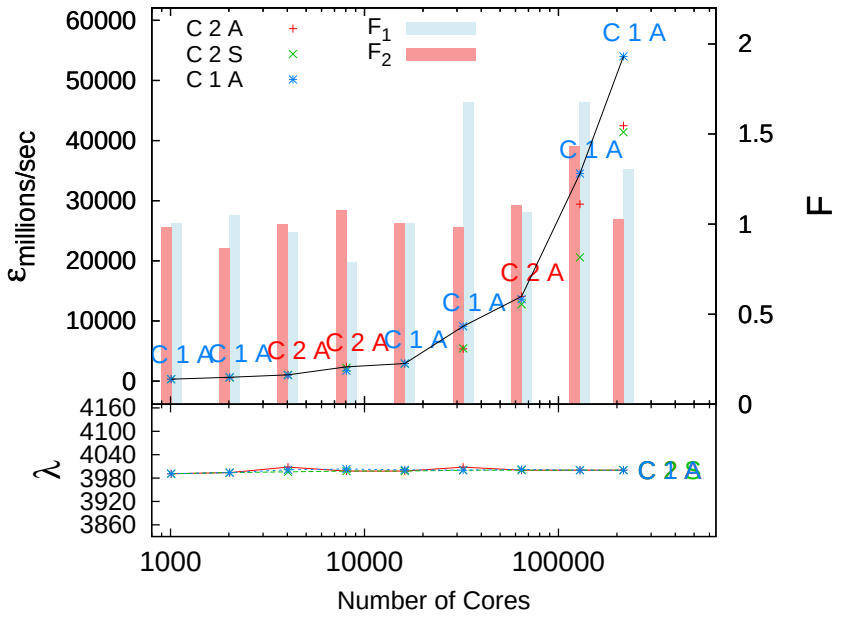
7.2. Additional RCREDIF Results

RCREDIF at very high lookaheads of 1 for conservative synchronization as shown in Figure 29 exhibit similar behavior to that of high lookaheads of 0.5. However, the efficiency gains at high scales due to asynchronous GVT computations is less pronounced. This can be attributed to the reduced frequency of GVT computation and thus the non-



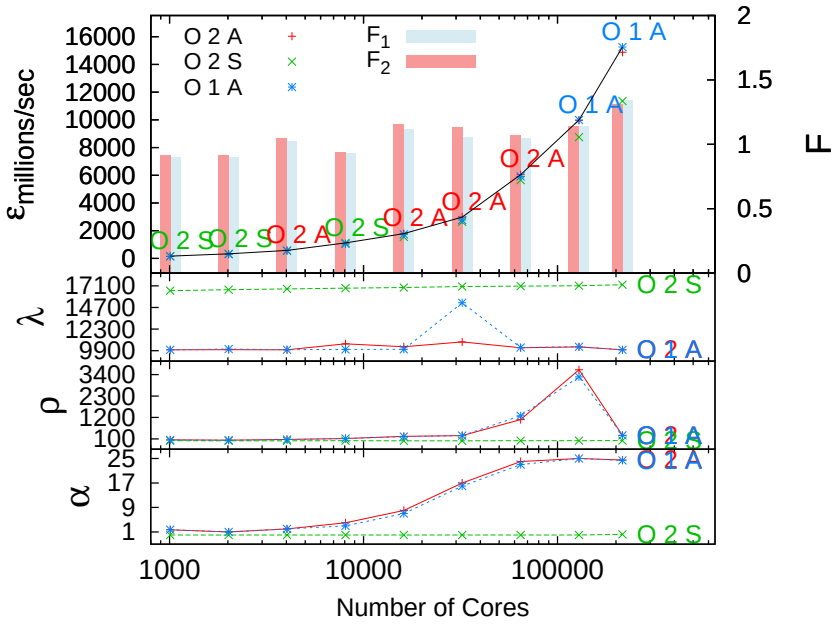
(a) LA=0.1, (100,100)

Fig. 14: RCPHOLD Conservative Synchronization, Low Lookahead



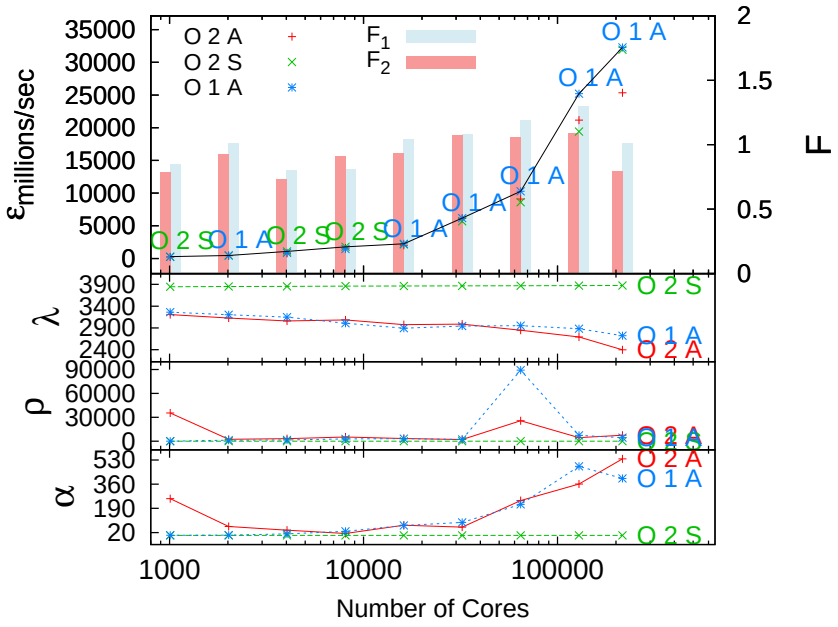
(a) LA=0.5, (100,100)

Fig. 15: RCPHOLD Conservative Synchronization, High Lookahead



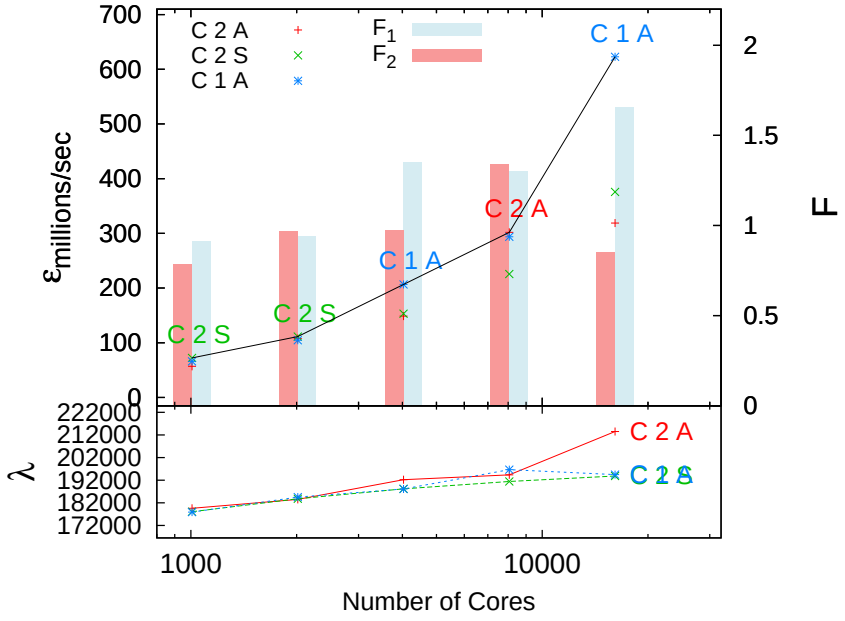
(a) LA=0.1, (100,100)

Fig. 16: RCPHOLD Optimistic Synchronization, Low Lookahead

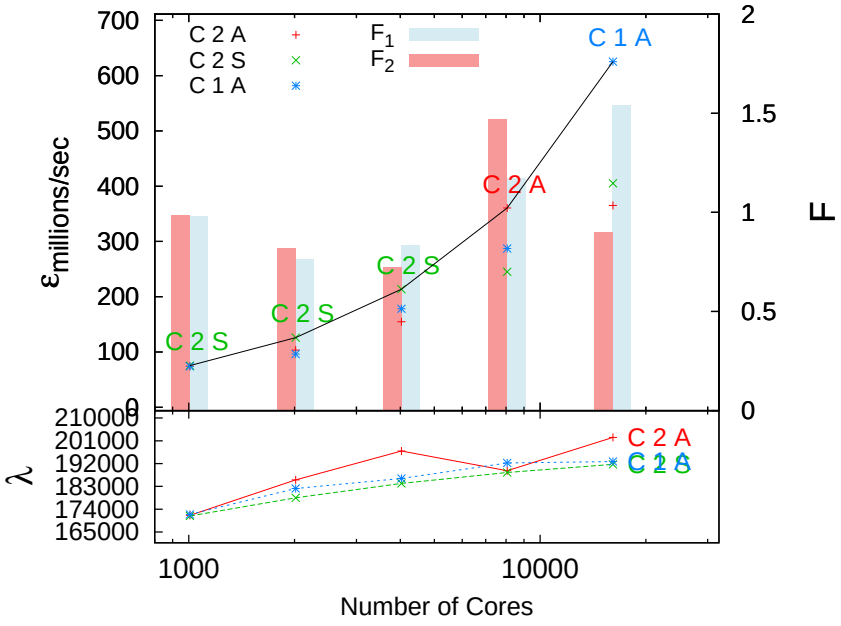


(a) LA=0.5, (100,100)

Fig. 17: RCPHOLD Optimistic Synchronization, High Lookahead

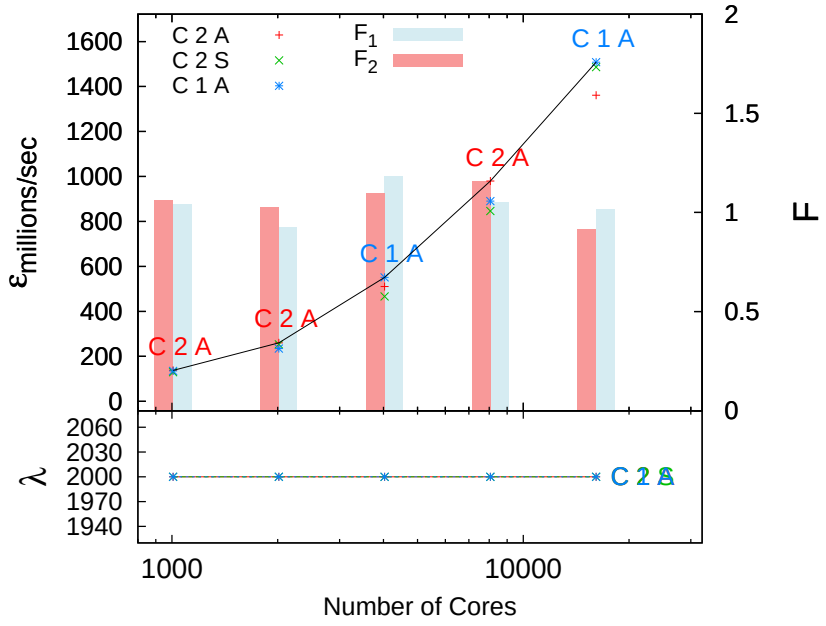


(a) LA=0.01, (10,1000)

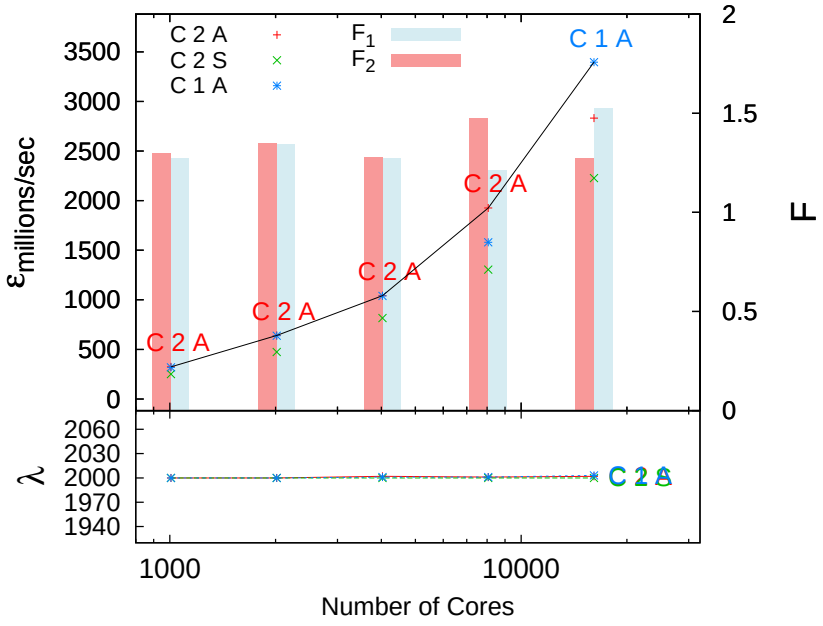


(b) LA=0.01, (100,100)

Fig. 18: RCPHOLD Conservative Synchronization, Very Low Lookahead

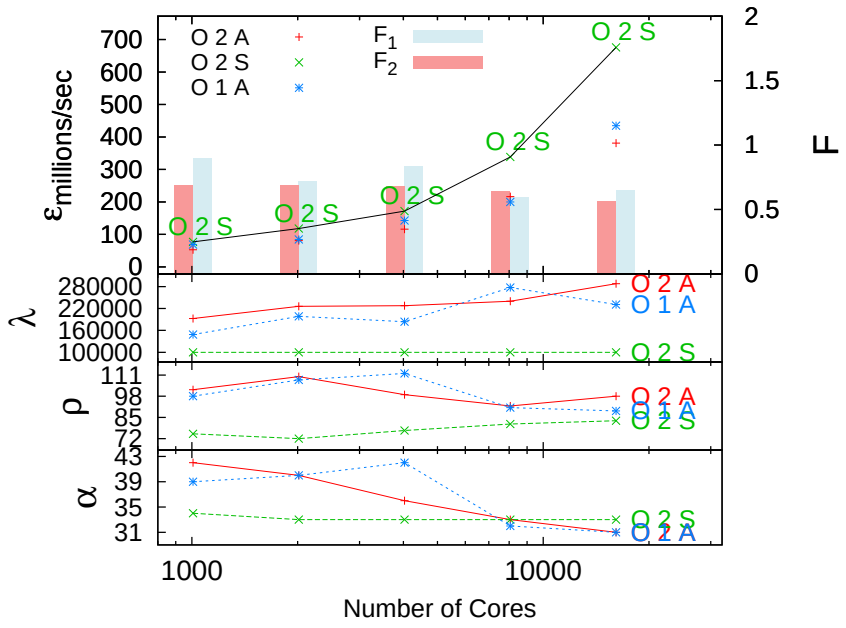


(a) LA=1, (10,1000)

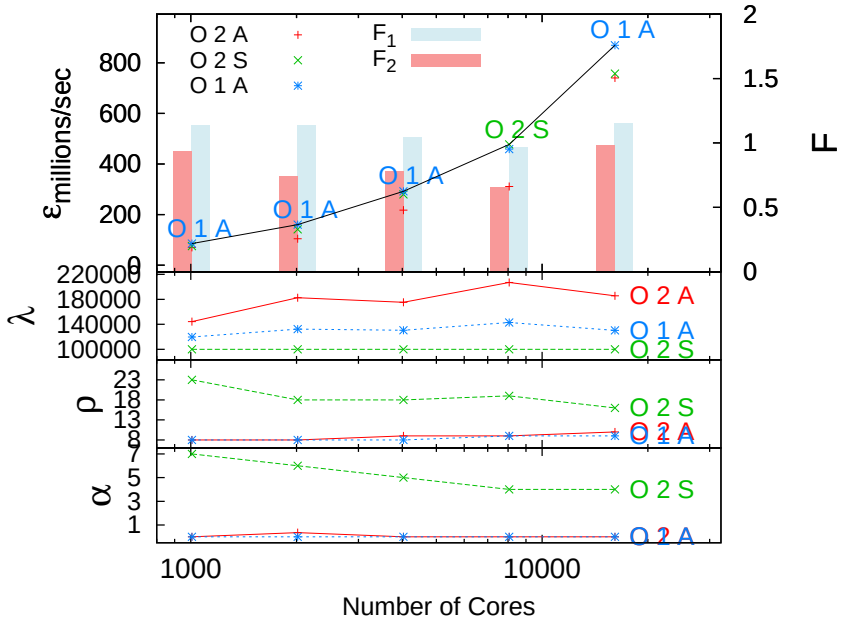


(b) LA=1, (100,100)

Fig. 19: RCPHOLD Conservative Synchronization, Very High Lookahead

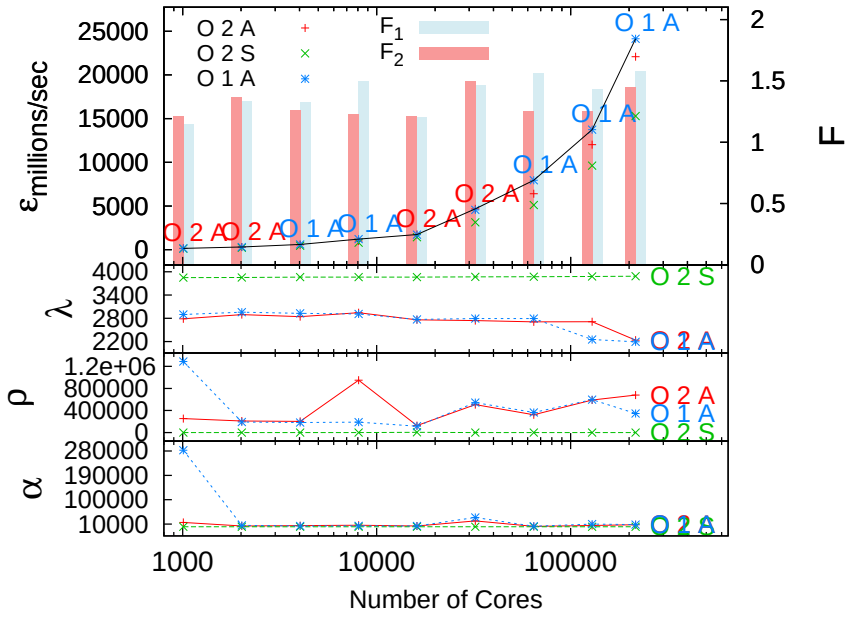


(a) LA=0.01, (10,1000)

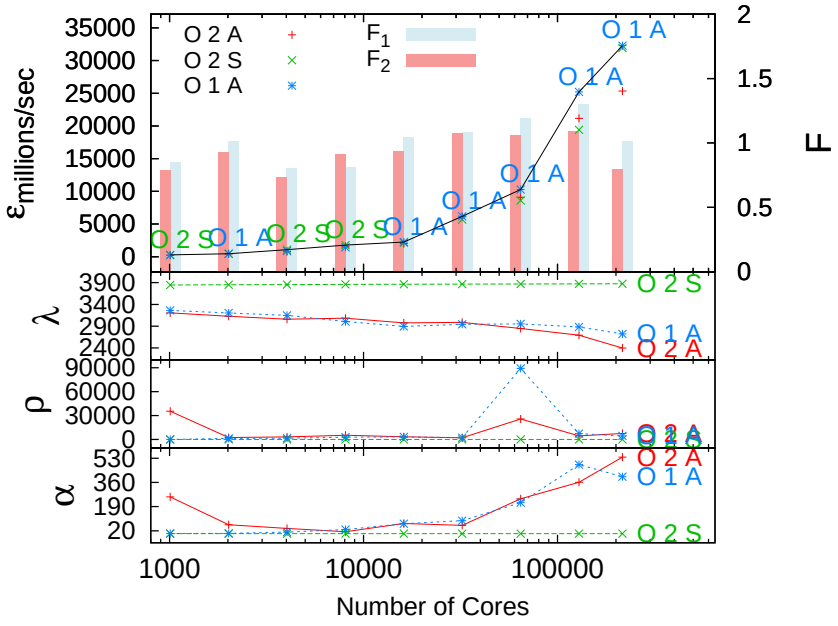


(b) LA=0.01, (100,100)

Fig. 20: RCPHOLD Optimistic Synchronization, Very Low Lookahead

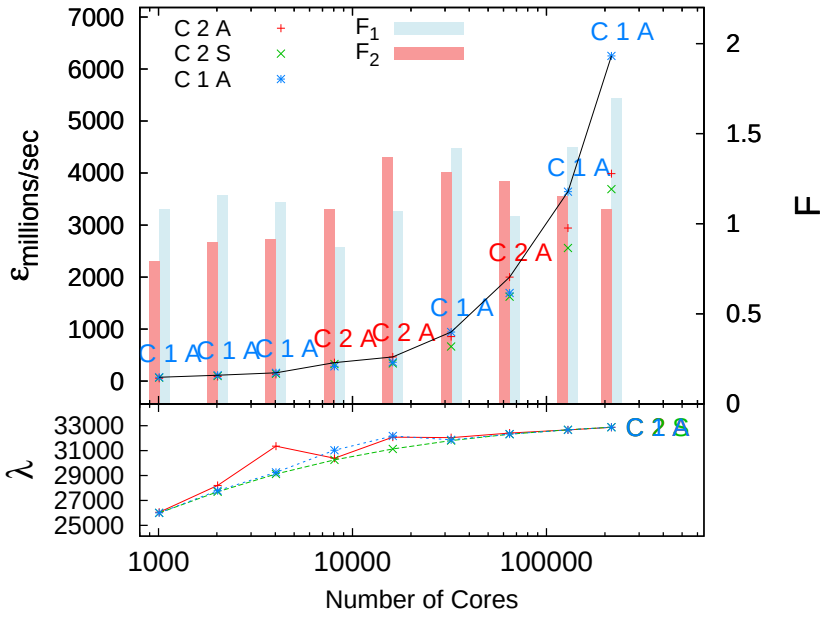


(a) LA=0.5, (10,1000)



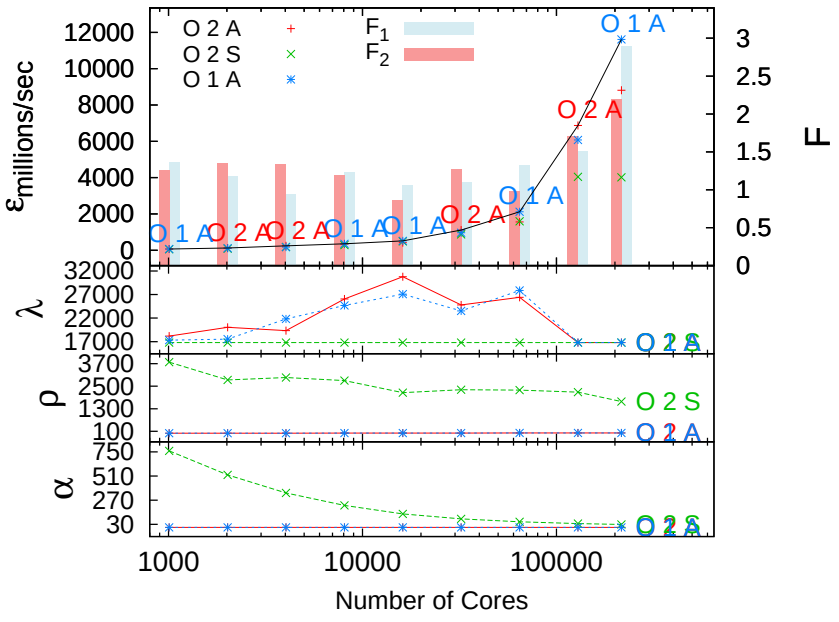
(b) LA=0.5, (100,100)

Fig. 21: RCPHOLD Optimistic Synchronization, Very High Lookahead



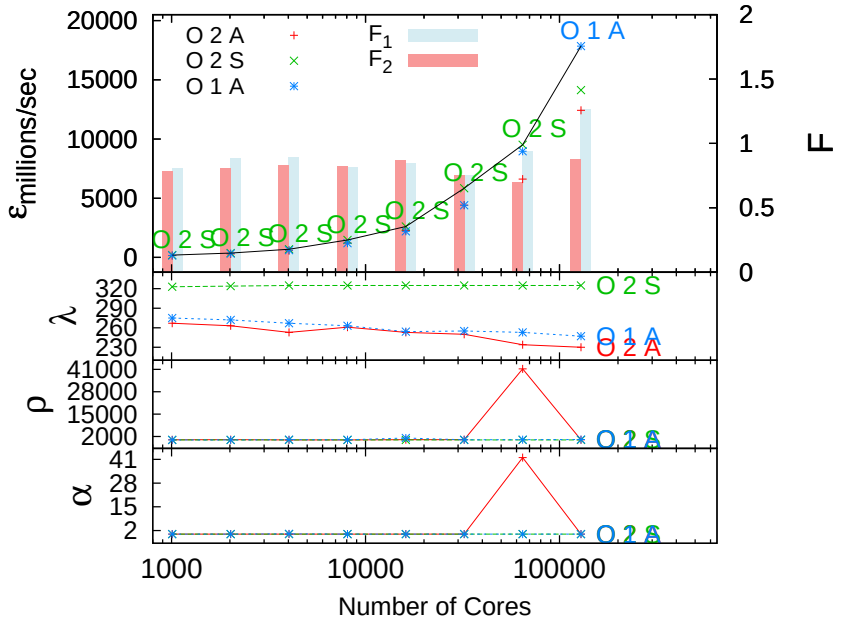
(a) LA=0.01, (100,100)

Fig. 22: RCREDIF Conservative Synchronization, Low Lookahead



(a) LA=0.01, (100,100)

Fig. 23: RCREDIF Optimistic Synchronization, Low Lookahead



(a) LA=1, (100,100)

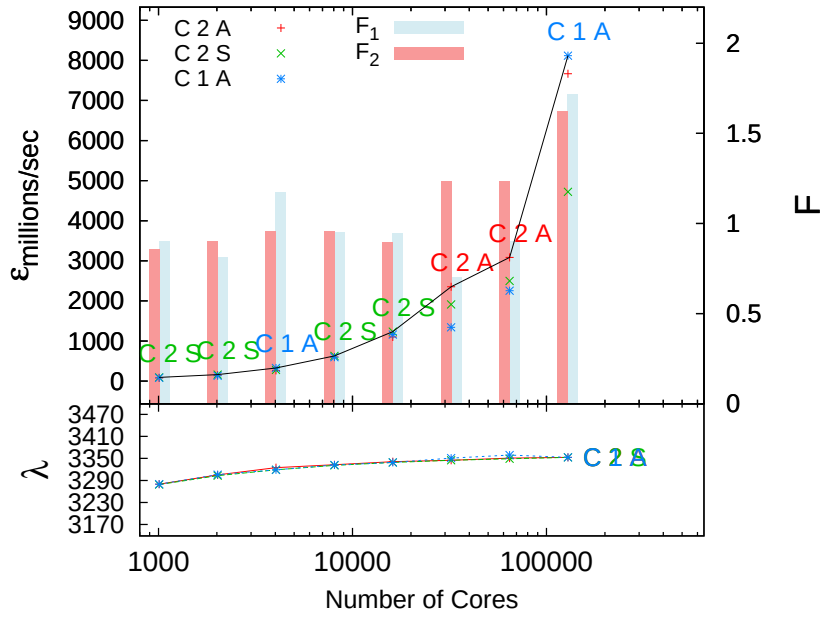
Fig. 24: RCREDIF Optimistic Synchronization, High Lookahead

interferring impact of the computation on the forward progress of the simulation as seen by the halved λ value, as expected, compared to that of lookahead of 0.5.

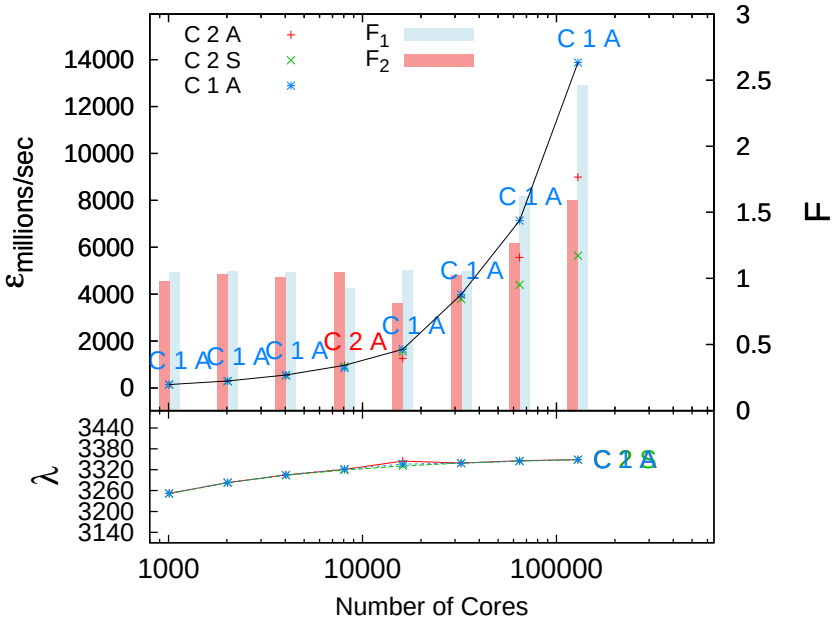
Figure 30 shows the accompanying simulation using optimistic synchronization at a lookahead of 1. We observe similar synchronization behavior to that of lookahead of 0.5 where the synchronous two-sided GVT computation performs well up until the largest of scales. At the 129,024 core mark, we observe the asynchronous GVT computations perform better. Although further experimentation is needed to uncover reasons why this occurs, the size of the simulation may begin to burden the network which may compete with GVT computations. Asynchrony and one-sided communications in these cases could provide the necessary flexibility to provide timely information needed for simulation progress.

In the cases of higher lookahead values of 0.5 and 1 shown in Figure 26 scaled to 129,024 cores, the difference between the GVT algorithms is lessened due to the relatively small λ during the simulation execution. Even in most of these cases, we observe that both asynchronous GVT algorithms provide ε performance on par with that of the synchronous two-sided GVT algorithm, if not better. This clearly becomes the case at 129,024 cores where the use of the asynchronous GVT algorithms, and one-sided communication in particular, provide significant performance gains.

Figure 27a and Figure 27b show RCREDIF scenarios for lookahead of 0.1 to 129,024 cores. For the particular structure of `(10,1000)` it is interesting to observe that the synchronous two-sided GVT algorithm outperform both asynchronous GVT algorithms to scale. The differences in λ, ρ and α do not suggest such a gap, but a possible explanation may rest with the synchronizing effect of the synchronous two-sided GVT algorithm which essentially forces the simulation to globally synchronize every so often which may be advantageous for this particular structure at this lookahead. This is further evidenced by the `(100,100)` structure which exhibits similar λ, ρ and α trends

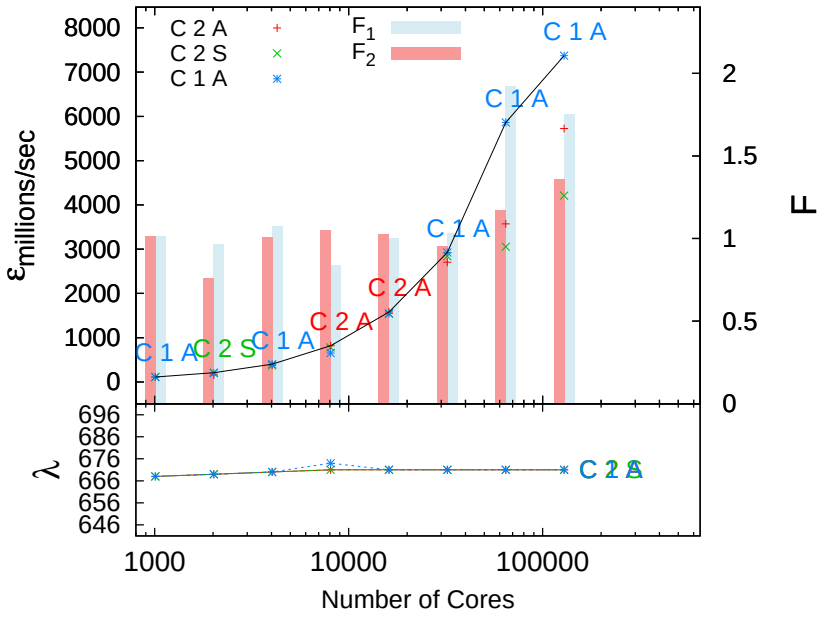


(a) LA=0.1, (10,1000)

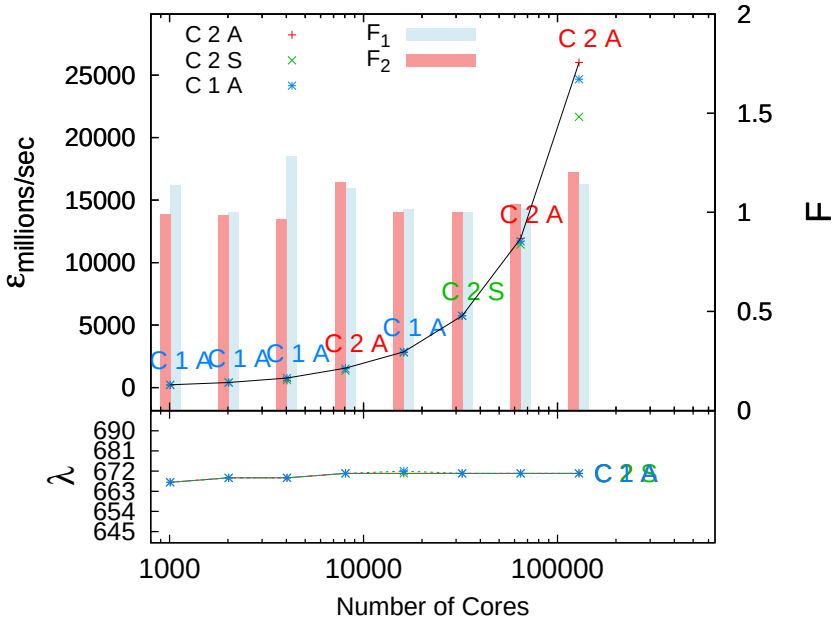


(b) LA=0.1, (100,100)

Fig. 25: RCREDIF Conservative Synchronization, Low Lookahead

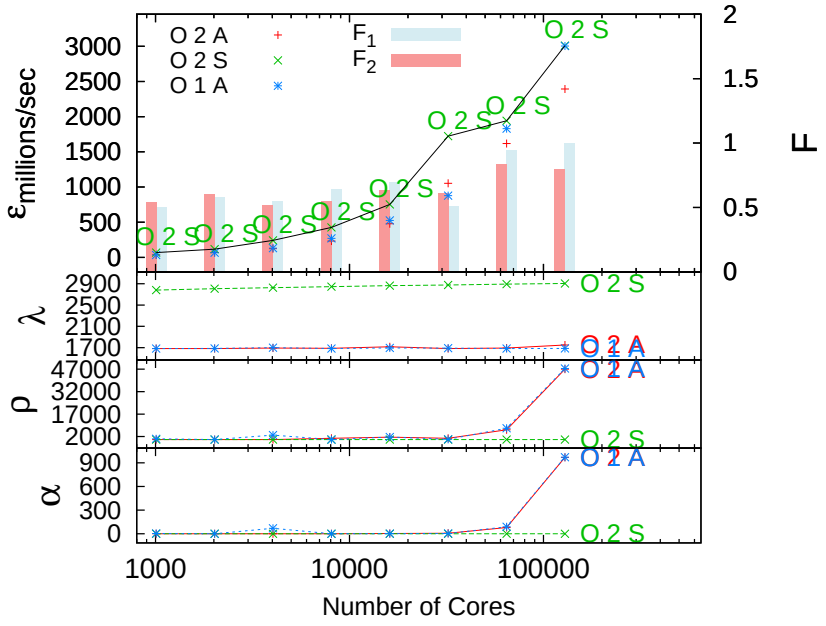


(a) LA=0.5, (10,1000)

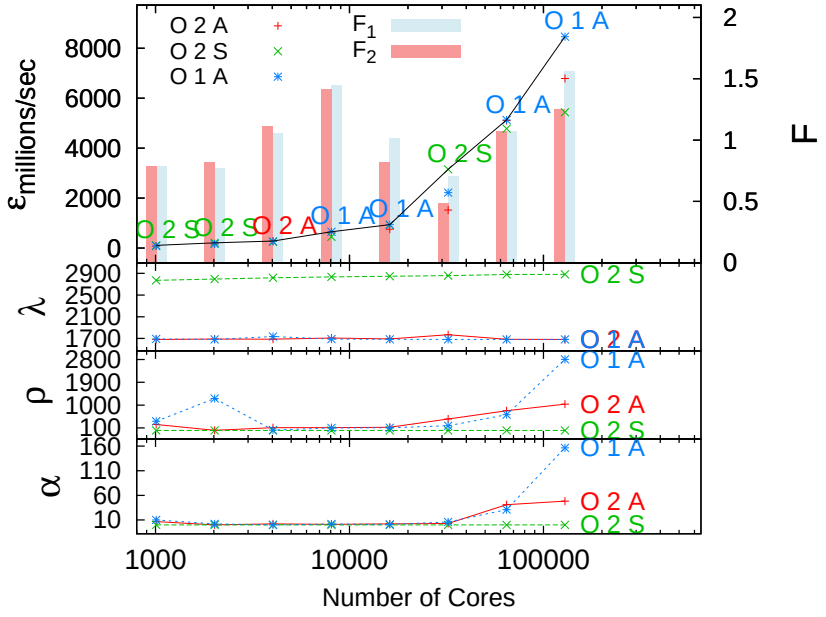


(b) LA=0.5, (100,100)

Fig. 26: RCREDIF Conservative Synchronization, High Lookahead

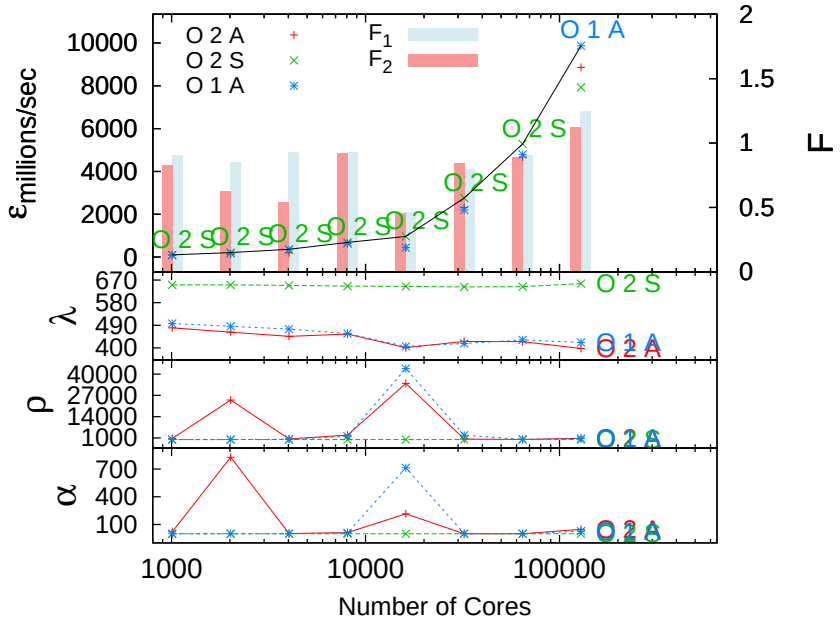


(a) LA=0.1, (10,1000)

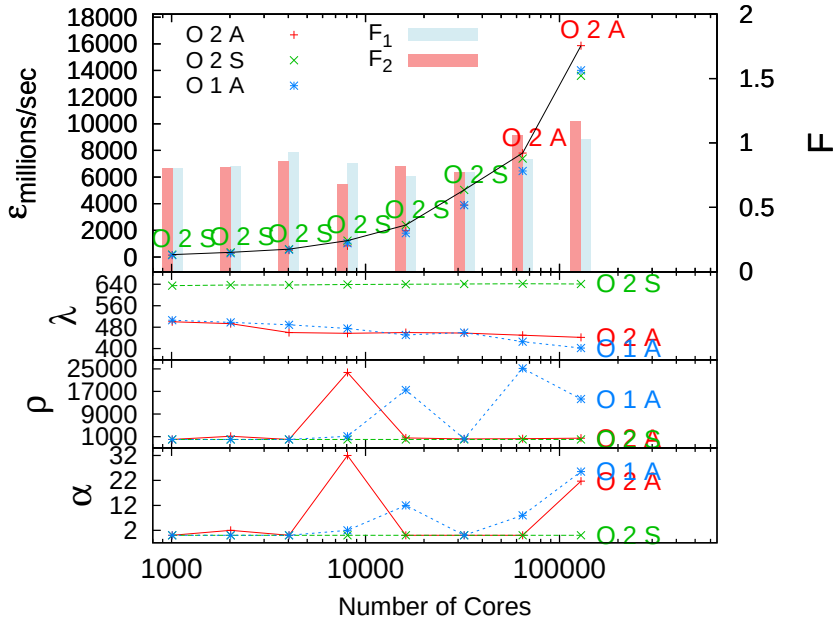


(b) LA=0.1, (100,100)

Fig. 27: RCREDIF Optimistic Synchronization, Low Lookahead

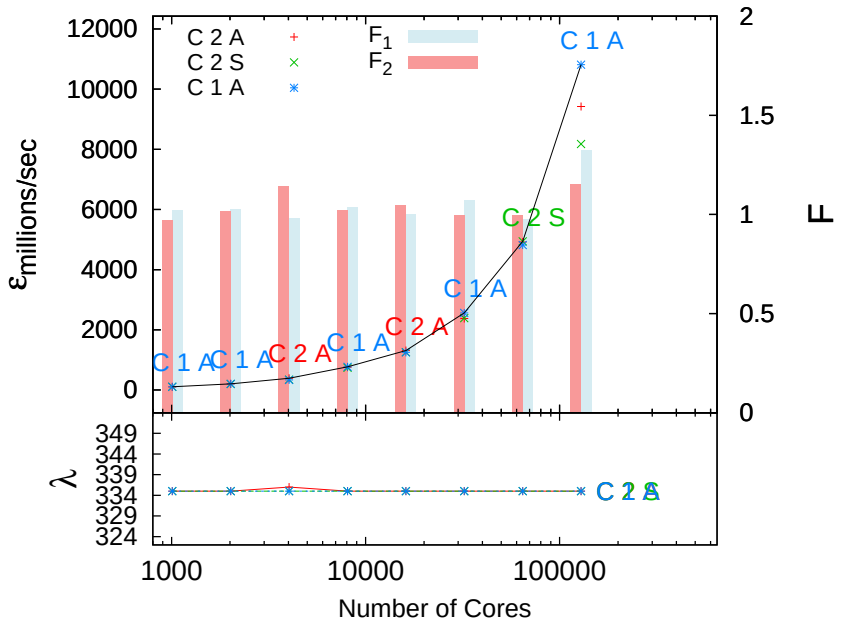


(a) LA=0.5, (10,1000)

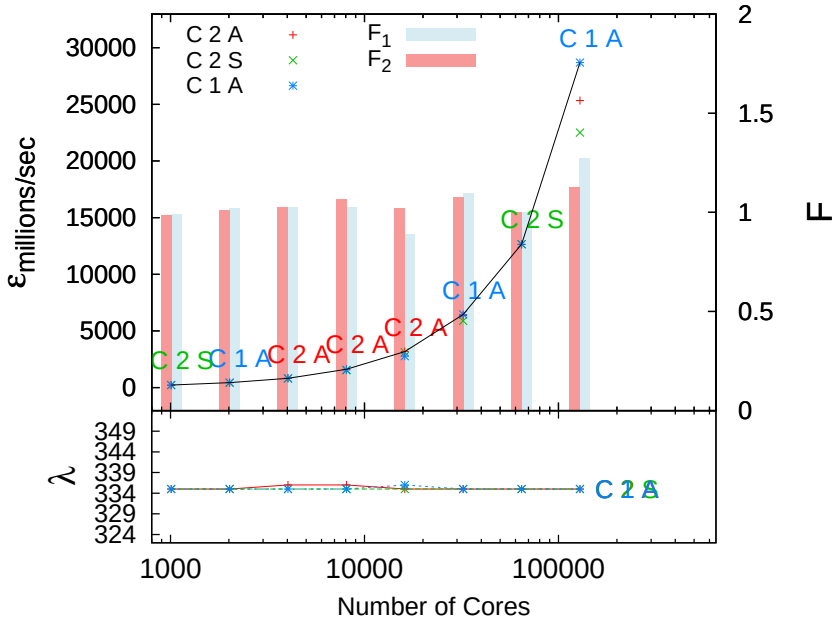


(b) LA=0.5, (100,100)

Fig. 28: RCREDIF Optimistic Synchronization, High Lookahead

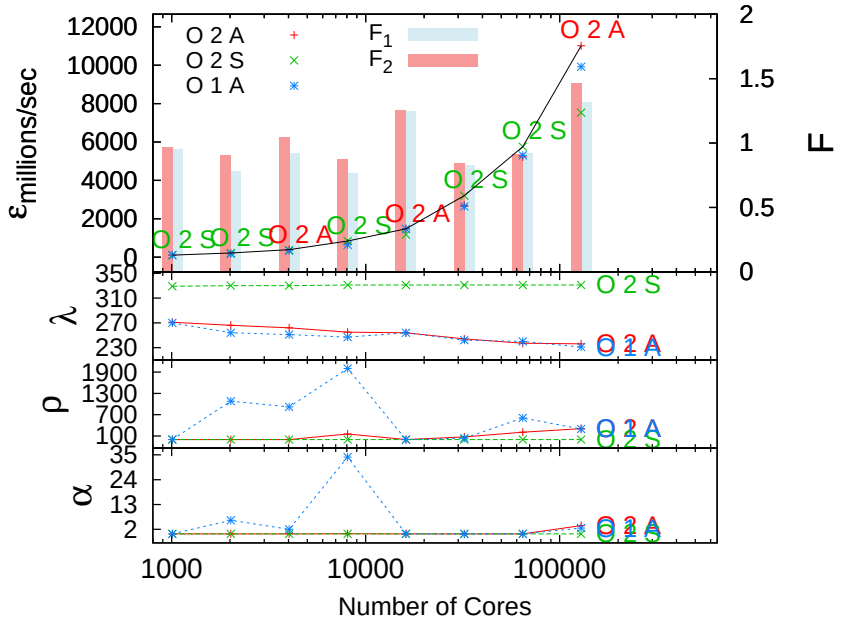


(a) LA=1, (10,1000)

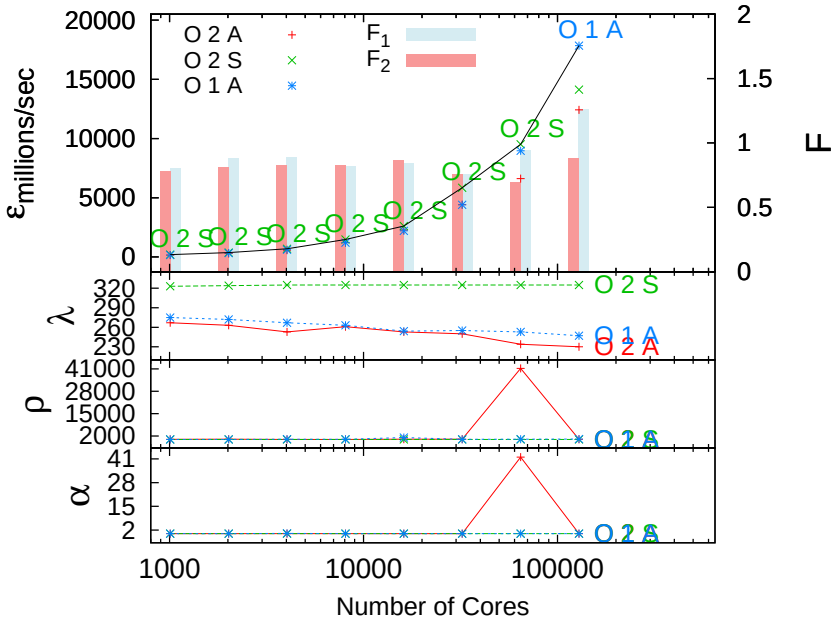


(b) LA=1, (100,100)

Fig. 29: RCREDIF Conservative Synchronization, Very High Lookahead



(a) LA=1, (10,1000)



(b) LA=1, (100,100)

Fig. 30: RCREDIF Optimistic Synchronization, Very High Lookahead

but does not show an across-the-board speed advantage for the synchronous two-sided GVT algorithm.

For high lookahead values of 0.5 shown in Figure 28, the trends clearly favor the synchronous two-sided GVT algorithms up to 64,512 cores. Generally, the synchronous two-sided GVT algorithm incurs nearly no rollbacks while both asynchronous GVT algorithms do. Since λ is reduced significantly in these cases due to high lookahead, each GVT advance is important to ensuring minimal amount of rollbacks. The scenarios employing the synchronous two-sided GVT algorithm synchronize more frequently: up to 50% more than both asynchronous cases. However, since λ in these runs are relatively small compared to the total elapsed runtime of the simulation, the additional number of λ do not significantly interfere with event computations. This tends to prevent excessive rollbacks and thus lower ε performance as shown in the respective charts. At 129,024 processors, the asynchronous GVT algorithms outperform the synchronous two-sided GVT algorithm. With λ and ρ metrics remaining consistent with the prior data point at 64,512 cores, the drop-off in performance for synchronous two-sided GVT performance might be attributed to the increased wallclock time incurred per λ. Further experimentation is needed to verify the cause of the performance degradation for synchronous two-sided GVT at very large-scale for RCREDIF.

Received February 2012; revised April 2013; accepted TBD 2013

ACKNOWLEDGMENTS

This paper has been authored by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the U.S. Department of Energy (DOE). Accordingly, the United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. This research was supported by the Early Career Research Program of the DOE Office of Science, Advanced Scientific Computing Research. The research used resources of the National Center for Computational Sciences (NCCS) at Oak Ridge National Laboratory, which is supported by the DOE Office of Science.