LogGOPSim – Simulating Large-Scale Applications in the LogGOPS Model

Torsten Hoefler
University of Illinois at Urbana-Champaign
Urbana IL 61801, USA

Timo Schneider
Open Systems Lab, Indiana University
Bloomington IN 47405, USA

Andrew Lumsdaine
Open Systems Lab, Indiana University
Bloomington IN 47405, USA

ABSTRACT

We introduce LogGOPSim—a fast simulation framework for parallel algorithms at large-scale. LogGOPSim utilizes a slightly extended version of the well-known LogGPS model in combination with full MPI message matching semantics and detailed simulation of collective operations. In addition, it enables simulation in the traditional LogP, LogGP, and LogGPS models. Its simple and fast single-queue design computes more than 1 million events per second on a single processor and enables large-scale simulations of more than 8 million processes. LogGOPSim also supports the simulation of full MPI applications by reading and simulating MPI profiling traces. We analyze the accuracy and the performance of the simulation and propose a simple extrapolation scheme for parallel applications. Our scheme extrapolates collective operations with high accuracy by rebuilding the communication pattern. Point-to-point operation patterns can be copied in the extrapolation and thus retain the main characteristics of scalable parallel applications.

1. INTRODUCTION

Parallel application simulation has long been used to investigate the performance of applications in different environments. Several simulators, such as PSINS, DIMEMAS, or BigSim, are already used by application developers as a part of their development tool chain. Simulations are very effective to discover the sources of performance and correctness bugs in parallel programs [2]. Simulations can also be used to check the effects of architectural changes (e.g., cache-sizes or network parameters) or changes in the middleware (e.g., different collective algorithms) on parallel applications.

Traditionally, simulators can be divided into three classes: application, architecture, and full-system simulators. We note that this distinction is not always strict and the classes blur together sometimes. Application simulators usually focus on properties of the application or algorithm, such as synchronization behavior of happens-before relations [10]. Application communication traces are often sufficient to reflect the critical properties of an application [4, 26]. Architecture simulators often employ a detailed model of one or more components of a parallel architecture (e.g., communication network, memory subsystem, or CPUs/caches), but are often only applied to traces of application kernels due to the high costs of a detailed simulation. Full system simulators aim to provide a detailed model of a parallel architecture such that whole applications, including complete software stacks, can execute in them. However, full system simulations can be expensive due to the need to emulate the whole Instruction Set Architecture (ISA) of the target machine. Common to all simulators is a fundamental trade-off between simulation accuracy (i.e., detail of the model) and simulation time.

Our work aims to study parallel application and algorithm behavior under the influence of different network and system models at large-scale. The main goal is to simulate short phases of applications under the LogGOPS model (such as a small set of collective operations) with up to 8 million processes on a single CPU to analyze the scaling of the algorithm. Applications with a reasonable number of messages (typical executions of 5 minutes or more) should be possible up to 50,000 processes. In order to allow scaling to large node counts with comparatively little simulation and execution resources, we chose a trace-based method to gather the characteristics of the target applications and trace extrapolation for large-scale simulations. The specific contributions of our work are:

- a simple and efficient discrete event simulation kernel for LogGOPS
- a full LogGOPS simulation framework for parallel applications
- a tool-chain to perform trace gathering, simulation, and visualizations
- a simple and effective trace extrapolation scheme
- a simulation analysis of the two scalable applications Sweep3D and MILC

1.1 Related Work

Focusing on application simulation, we omit references to architecture simulation. An up-to-date list of architecture and full-system simulators can be found on Derek Hower et al.'s "WWW Computer Architecture Page" [16].

Several parallel application simulators have been proposed in the past. They can roughly be divided into the two classes by their simulation technique: The first technique is full system simulation where an application is executed in a simulator that fully emulates the target architecture. There exist two approaches to increase the performance of such a simulation. Direct execution uses the host architecture to execute parts of the code natively in order to accelerate the simulation. Examples for simulators in this class are BigSim [27], SILAS [10], and MPI-SIM [5]. The memory requirements of direct execution are very high and the system used to generate the traces usually has to have as much memory as the target system which often makes this approach impossible for large-scale simulations. Adve et al. [1] uses techniques from static analysis to reduce the memory footprint of such simulations. The second class, trace-driven or post mortem simulation, uses traces of the application execution to extract significant properties for the simulation. Traces enable easy scaling of accuracy vs. simulation time by changing the granularity of the information in the trace. Examples are PSINS [26], DIMEMAS [23] and Rugina's and Schauser's simulator [25].

Our work is similar to the latter category with two main differences: (1) the simulator has been tuned for small trace files and fast simulation of large-scale systems and (2) collective operations are replaced by a set of point-to-point algorithms which are optimized to the underlying network, such as an MPI implementation would do. The latter differs significantly from previous approaches in PSINS or DIMEMAS where collective operations are modeled as global synchronization and some analytic model. Our design preserves the fact that not all collective operations are synchronizing (see [22], e.g., MPI_Bcast() usually does not incur global synchronization) and that the arrival pattern at a collective and the leave pattern depend on the actual collective implementation.

1.2 The LogP Model Family

The original LogP model by Culler et al. [6] models message passing systems with the parameters L, o, g, and P. The model generally assumes that all P processors in a network are connected with bidirectional channels and only fixed-size (small) messages are communicated. The latency parameter L models the maximum latency among any two processors in the system. The parameter o reflects the CPU overhead per message and the parameter g the time between two message injections into the network. Thus, the model considers pipelining in the network and a maximum of L/g packets can be in flight between two endpoints. The LogP model assume a network that is free of contention, that is, the capacity of the network is at least $P^2 \cdot L/g$.

The LogGP model by Alexandrov et al. [3] adds an additional parameter G—the cost per byte of messages—to the LogP model. This models the capability of most networks to transmit large messages relatively fast (fragmentation and reassembly in hardware). Thus, a cost-per-byte metric (G) is more accurate then modeling multiple small messages (limited by g) in LogP.

The LogGPS model by Ino, Fujimoto, and Hagihara [17] adds sender synchronization to the model. Message passing of large messages is often performed by sending small control messages to the receiver to check if enough buffering is available. This so called *rendezvous protocol* causes the sender to be synchronized with the receiver, that is, the message cannot be sent before the receiver is ready to receive it. The new parameter S determines the message-size threshold for synchronizing sends.

Other LogP extensions, such as pLogP [18], LogPC [21], and LogfP [12] are not included in this work, but might be considered in future work.

1.3 The LogGOPS Model

The most important advantages of the LogGP model over simple latency-bandwidth models ($T = \alpha + \beta \cdot s$) is the ability to model network pipelining and computation/communication overlap accurately. The latter is achieved by distinguishing the CPU and network parts of the overhead, o and g respectively. Simpler models would fold both parameters into one (α) and one would prohibit the determination of CPU usage of communications. If CPU usage and overlap are not important, LogGP analyses, such as [3], fold the two parameters into one (e.g., by defining $o < g$) and essentially fall back to the simpler model. However, in our simulations, we carefully distinguish both parameters in order to model overlap of communication and computation accurately.

One shortcoming of the LogGPS model is that it models only a constant overhead per message send which is independent of the message size. While this is correct on some architectures (e.g., when performing remote direct memory access (RDMA) with preregistered memory), other architectures incur some cost per byte (e.g., TCP packet processing or memory registration in InfiniBand [11, 20]).

We use the Netgauge tool to investigate the scaling of the CPU (send) overhead with the message size. The highly accurate measurement technique is outlined below and described in detail in [11]. The main problem in network measurements is that, in the general case of non-synchronous clocks, time differences can only be measured on a single node. Thus, we employ a client-server model where the client sends multiple (n) messages (of varying size s) to a server and receives an acknowledgment from the server. The client waits for a specified time (the delay d) between message transmissions. A round-trip time with n messages and a delay d is called RTT_n^d in the following. The benchmark ensures that $d > s \cdot G + g$ and the client takes the time from the first message to the acknowledgment and we can compute o via (cf. Equation 6 in [11])

$$o = \frac{RTT_n^d - RTT_1^0}{n-1} - d.$$

Figure 1 shows the measurement scheme in detail for $n = 3$.

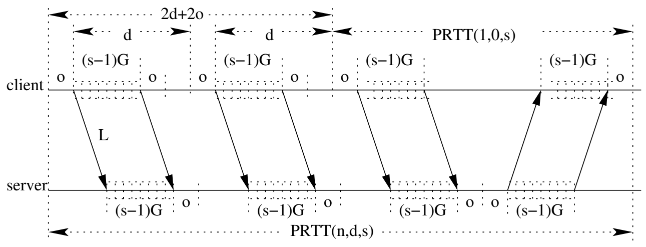

Figure 1: Measurement of the LogGP Overhead for $n = 3$.

Figure 2 shows the measured overheads for four different systems: an InfiniBand cluster (Odin), a Myrinet 2000 cluster (Big Red), a BlueGene/P, and an XT-4 (Jaguar). All machines show an increase of the overhead with growing message sizes. We propose

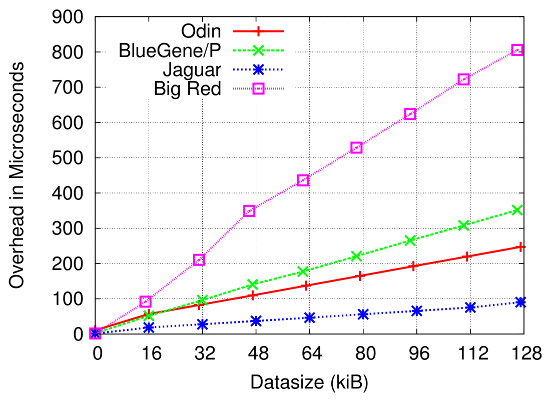

Figure 2: Measured overheads for different systems.

to use a linear function ($f(s) = a \cdot s + b$) to fit the curves. This method is very similar to g and G in the LogGP model where we would simply replace $a \rightarrow G$ and $b \rightarrow g$. Following the LogGP model, we introduce the new parameter O in order to model the *overhead per byte*, replacing $a \rightarrow O$ and $b \rightarrow o$ in the fitted linear equation. This resembles the LogGP idea and accurately captures the growing overhead while not adding too much complexity to the LogGPS model. Our measured values for O are $1.4ns$, $6.2ns$, $2.5ns$, and $0.6ns$ for Odin, Big Red, BlueGene/P, and Jaguar, respectively. The asymptotic standard error of the least squares fit was less than 1% for all systems.

2. LARGE-SCALE SIMULATIONS

Our goal is the simulation of large-scale applications with more than 8 million processes. We mainly focus on the communication part and the LogGOPS model. Computations are simply modeled as delays and communications as send/receive pairs. We chose a dialect of the Group Operation Assembly Language (GOAL) to express all operations. GOAL is a language to define complex synchronous and asynchronous parallel algorithms.

2.1 Group Operation Assembly Language

GOAL, first proposed in [15] to express efficient offload for blocking and nonblocking collective operations, can express arbitrary parallel applications. The computation is expressed as a process-local task dependency graph with remote dependencies mandated by happens-before relations in message passing systems. GOAL defines three different types of tasks: send, receive, and computation (loclop). Tasks are arranged as a directed acyclic graph, called *schedule*, with dependencies among the tasks as edges. We use a GOAL schedule to represent the computation and communication of a single process. A parallel program with P processes is represented by P GOAL schedules. For example, the following schedule models process 0 that computes for 100 microseconds and then exchanges 10 bytes of data with process 1:

```
rank 0 {
 l1: calc 100 cpu 0
 l2: send 10b to 1 tag 0 cpu 0 nic 0
 l3: recv 10b from 1 tag 0 cpu 0 nic 0
 l2 requires l1
}
```

The values tag, CPU, and NIC are optional (default to 0) and enable the modeling of MPI-like message matching semantics, multi-CPU systems, and multi-NIC systems, respectively.

This textual GOAL representation is converted into a binary GOAL schedule for faster cache-efficient simulation (cf. serialization in [15]).

2.2 The Simulator Design

The simulator consists of two main parts: the parser that reads binary GOAL schedules and the simulation core which executes the simulation. One of the main goals was to keep both parts as simple and as fast as possible. The parser exposes all P schedules and manages dependencies and execution order. It returns a list of executable operations and offers an interface to mark operations as executed.

The simulation core is based on a single priority queue (or heap), the *active queue* (AQ). The AQ contains all executable operations of all processes and their earliest start times. As soon as an operation is retrieved from the parser, it will be added to the queue. Each operation in the queue is annotated with an execution time and a process number and the operation that can execute next is returned first. Figure 3 shows the program flow of the core simulation.

MPI Message Matching.

To simulate full message passing semantics, we add an *unexpected queue* (UQ) and a *receive queue* (RQ) for each process. Messages are matched with MPI semantics, that is, the tuple (tag, source) and the order determine matching. LogGOPSim supports indeterministic messaging (any_source, any_tag) as well as eager and rendezvous protocols.

To do this, we add a new operation type *message* (msg) which models messages in flight. To model nonblocking (immediate) messages, we use a special dependency type *immediate requires* (irequires) which can be satisfied when an operation *started* while the normal dependencies are satisfied when an operation *completed*. In MPI terms, an irequires would reflect a dependency on an immediate operation (e.g., isend or irecv) while a requires would be linked to the according wait (or test loop).

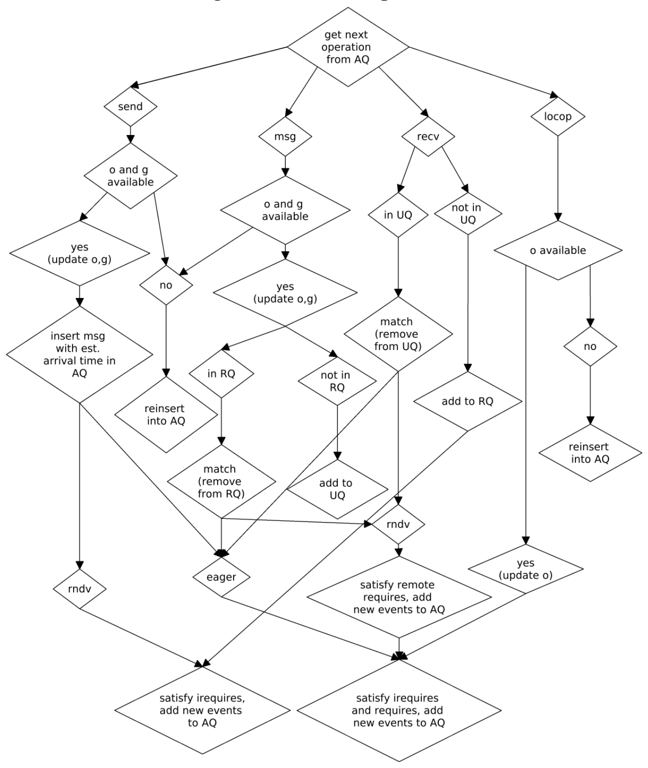

Figure 3: Simulation Core Program Flow.

The simulator supports multiple execution cores (CPUs) and multiple network cards (NICs). The process-specific simulation variable o_p^c is set to next time when o can be charged on CPU c at process p. Similarly, g_p^n indicates the next time when g or G can be charged at NIC n on process p. We will shortly discuss the flow for the four operation types (cf. the four top branches in Figure 3) in the following:

(1) If a send (process p, NIC n, and CPU c) is fetched from the AQ and o_p^c and g_p^n are available, then o_p^c and g_p^n are increased and a new operation of type message is inserted into the AQ with its time advanced by L and o. Else, if o_p^c or g_p^n are not available, the operation is reinserted into the AQ with the earliest execution time (max(o_p^c, g_p^n)). Started rendezvous sends satisfy only irequires while *eager* sends activate all dependencies. Requires of rendezvous messages are satisfied when the message completes (see (2)).

(2) If an incoming message (`msg`) is found and o_p^c and g_p^n are available then it is matched against the RQ, received if a matching element is found, and otherwise added to the UQ.

(3) If a new receive operation (`recv`) is found then it is matched against the UQ and immediately satisfied if the message was al-

ready received, otherwise a new entry is added to the RQ of the process.

(4) A local calculation (`loclop`) is simply executed if the CPU (o_p^c) is available and otherwise reinserted into the AQ with the start time of the next available o_p^c.

This single-queue design allows very fast and accurate LogGPS simulations due to the low complexity of the simulation core.

2.3 Limitations and Assumptions

As mandated by the LogGOPS model, our simulator ignores congestion in the network and assumes full *effective bisection bandwidth*. We showed in [14] that networks with full bisection bandwidth and static routing (e.g., InfiniBand or Ethernet (folded) Clos topologies) nearly meet this requirement (60-75%). Networks with adaptive routing on (folded) Clos topologies such as Quadrics [24] or Myrinet [7] can deliver the required full effective bisection bandwidth. However, torus-based networks as used in BlueGene/L, BlueGene/P, the Cray XT-4, and XT-5 series might suffer from congestion.

Our simulation scheme makes two assumptions that are not stated in the original LogGPS model: We assume that each message is delayed until o and g are available at the receiver. We model each channel as full duplex, that is, g and G are either charged to g_{send} or g_{recv} not both.

Our message-centric simulation approach also ignores effects of input and output (I/O) which can sometimes influence the runtime significantly or even dominate it.

3. ANALYTICAL MODELS

In this section, we compare the results delivered by the simulator with analytical LogGPS models for different communication patterns. This demonstrates the correctness of our LogGPS simulation.

It is important to understand that the times o, O and g, G can overlap because some networks can progress messages independently from the CPU once descriptors are posted. This very important mechanism models overlap of communication and computation. On networks which do not allow progression or overlap, one sets $o = g$ and $O = G$.

Linear Scatter.

The first pattern represents a linear broadcast or scatter where a single process sends a message to a set of other processes. If we define $sO = (s-1)\cdot O$ and $sG = (s-1)\cdot G$, then the LogGPS prediction for this communication is (for $s < S$):

$$T_{scat} = 2o + L + \max\{(P-2)o + (P-1)sO), \quad (P-2)g + (P-1)sG\}. \qquad (1)$$

The max-term satisfies the LogGOP property that CPU and network work independently (communication/computation overlap). For $s > S$, we have to add another $(P-1)\cdot L$ to account for the synchronization costs for each message.

Figure 4 shows the simulation output visualization for a linear broadcast/scatter communication with $L = 2.5\mu s$, $o = 1.5\mu s$, $g = 4\mu s$, $G = 6ns$, $O = 8ns$, $P = 8$, $S = 65{,}535$, and $s = 1{,}024$. We verified the results against equation 1 for different parameters and larger P and found that the simulator works accurately.

Linear Gather.

The second pattern that we investigate is a linear gather pattern where each process sends data to a designated root. This pattern causes congestion at the root process and serializes all incoming

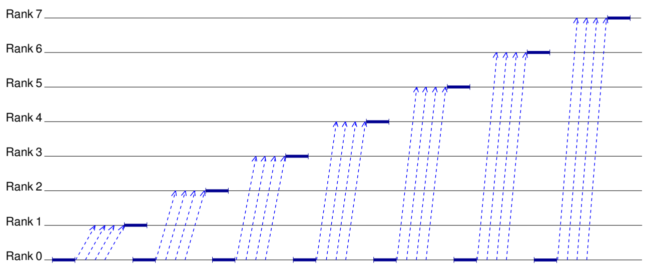

Figure 4: Linear Broadcast/Scatter Pattern.

data. The LogGPS model for $s < S$ for the linear scatter pattern is

$$\begin{aligned} T_{gat} &= 2o + L + \max\{(P-2)o + (P-1)sO), \quad (2)\\ & \quad (P-2)g + (P-1)sG\}, \text{ such that}\\ T_{gat} &= T_{scat}. \end{aligned}$$

Figure 5 shows the simulator output for the Gather pattern with the same parameters as for the scatter pattern. We also verified the results for different sets of parameters.

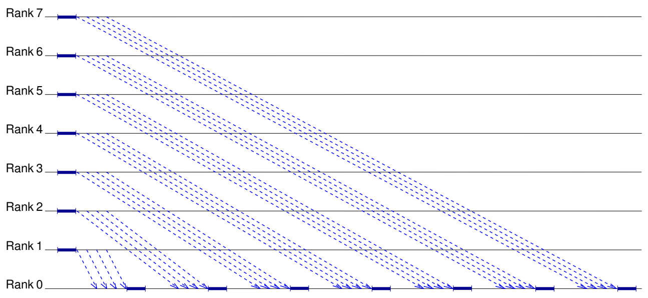

Figure 5: Linear Gather Pattern.

Binomial Tree Broadcast.

Tree patterns are a very important communication structure and are often used to implement broadcast or scatter operations. One particularly important pattern is the binomial tree. The runtime in the LogGPS model for $s < S$ is

$$T_{bino} = (2o + L + \max\{sO, sG\})\lceil \log_2 P\rceil. \qquad (3)$$

Figure 6 shows the simulation output for the binomial tree pattern with same parameters as before but $s = 1$ to improve its readability. We also verified this pattern extensively with different input parameters.

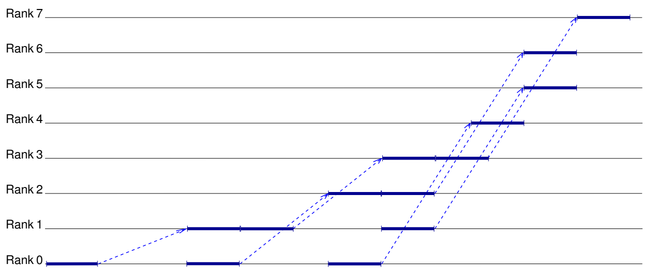

Figure 6: Binomial Tree Pattern.

Dissemination Pattern.

The last example pattern, the dissemination pattern, is often used for all-to-all data exchanges, barriers, or allreduce collective operations. The pattern has, like the binomial tree, $\lceil \log_2 P \rceil$ communication rounds. However, now, each process sends and receives each round from processes at exponentially growing distance. If we now assume that o is charged after the message arrived (e.g., a buffer copy), then each incoming message has to wait until the send finished and o has been charged. Thus, all processes complete at the same time after being delayed by this waiting time (δ) each round:

$$\delta = \begin{cases} (s-1)O - L : (s-1)O - L > 0 \\ 0 : \text{otherwise.} \end{cases} \quad (4)$$

$$T_{diss} = (\delta + 2o + L + \max\{sO, sG\}) \lceil \log_2 P \rceil \quad (5)$$

Figure 7 shows the simulation visualization for $s = 1$ and the previous parameters.

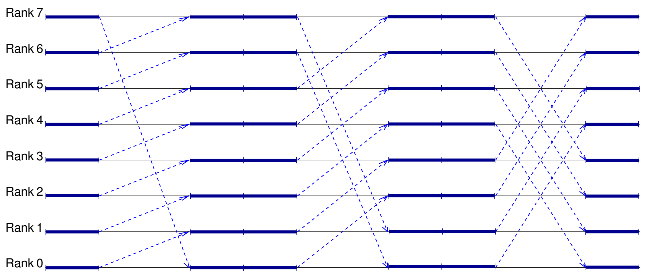

Figure 7: Dissemination Pattern.

We used the patterns explained above and many other test- and collective operation patterns (please refer to the source-code for a comprehensive list) to verify the correctness of the simulator.

4. EXPERIMENTAL EVALUATION

In the following, we discuss the absolute performance and the limitations of LogGOPSim. We begin the discussion with analyzing single collective operations at large scale and then show two application examples. With this, we want to compare the LogGOPS model with real-world settings. We see that simulations are reflecting real-world systems well despite fundamental restrictions in the model. However, our main emphasis lies in accurate LogGOPS simulations which allow us to make asymptotic statements about the scalability of applications in congestion-free networks rather than detailed system simulations.

4.1 Collective Operations at Large Scale

In this section, we analyze the accuracy and the performance of LogGOPSim for simple collective patterns at large scale. We chose two clusters, Odin and Big Red clusters as evaluation platforms to compare our simulations to real benchmarks. Odin consists of 2 GHz Opteron quad core InfiniBand (SDR) nodes with LogGOPS parameters: L=5.3μs, o=2.3μs, g=2μs, G=2.5ns, O=1ns, and S=32,768 bytes. Big Red consists of quad-core PPC 970 nodes connected with Myrinet 2000 with LogGOPS parameters: L=2.9μs, o=2.4μs, g=1.7μs, G=5ns, O=2ns, and S=32,768 bytes. We used Netgauge's LogGP [11] pattern to measure the parameters.

4.1.1 Simulation Accuracy

LogGOPSim offers different implementations for collective operations. We use broadcast based on binary and binomial trees as an example for this work. We used NBCBench [13] to accurately measure the execution time of single collective operations (to avoid pipelining, see [13]) and LogGOPSim to simulate the communication pattern. Figure 8 shows the prediction and the benchmark

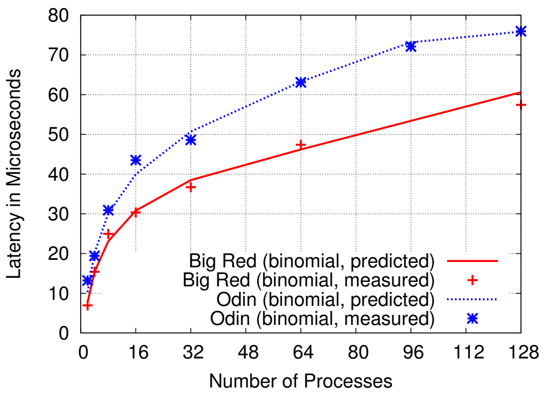

Figure 8: Prediction and Measurement Results for 1 Byte Broadcasts.

results for 1 byte messages on Odin. The lines indicate the simulation results and the dots the measurements. The average error for the prediction is less than 1% for both discussed systems and algorithms. We note, that small messages over InfiniBand (Odin) are not modeled well by LogGP (see the discussions about the LogfP model for InfiniBand in [12]) which makes the prediction of binomial tree algorithms with a large fan-out at the root inaccurate and leads to higher prediction errors. Myrinet does not exhibit such a behavior and also shows an error of less than 1% with binomial trees.

Figure 9 shows simulation results for a binomial broadcast of 128 kiB data (using the rendezvous protocol). The simulation underestimates the runtime over InfiniBand slightly (less than 16% at 128 processes). This is due to congestion in the real-world network which is ignored in the LogGOPS model.

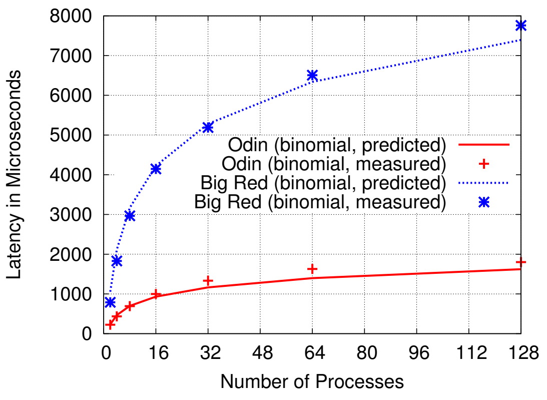

Figure 9: Prediction and Measurement Results for 128 kiB Broadcasts.

4.1.2 Simulation Performance and Limits

We test the scalability of our simulation on a 1.15 GHz Opteron workstation with 13 GiB available memory. Figure 10 shows the simulation runtime of 1,024 to 8 million processes executing a single binomial broadcast or a single allreduce operation implemented with the dissemination algorithm [9]. We exclude the (expensive) costs to generate the GOAL schedule and transform it into the binary representation because binary schedules can be used for multiple simulations with different parameters to offset the generation costs. The diagram shows that the simulation time grows linearly with the number of messages. The binomial tree broad-

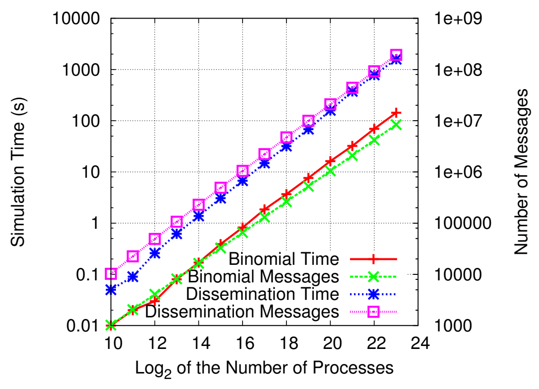

Figure 10: Simulation times.

cast sends $\Theta(P)$ messages while the dissemination algorithm needs $\Theta(P \log P)$ messages. The main limitation in the simulations is the memory consumption which increases linearly with the number of messages. In order to allow large-scale simulations, LogGOPSim enables an efficient out-of-core simulation mechanism by mapping the binary schedule into main memory (mmap) and can thus process schedules that are much larger than main memory (using the OS's paging mechanism). The simulation time is independent of the message size or the duration of local operations.

4.2 Application Simulations

To verify our simulator, we use the ASC application Sweep3D [19] version 2.2d which is frequently used to verify application simulations [1,5] and the MIMD Lattice Computation (MILC) `su3_rmd` benchmark [8] version 7.6.2.

Sweep3D solves a neutron transport problem on a 3D Cartesian geometry. The domain is represented by a rectangular IJK grid where each IJ plane is decomposed in one dimension and distributed. The K dimension is used for pipelining. The problem is solved in two nested iterative loops until convergence. Our weak-scaling runs showed between 6.45% and 13.4% communication overhead, mainly caused by point-to-point communication and various collective operations (4 broadcast, 3 barrier, 28 allreduce).

MILC is used to study quantum chromodynamics, the theory of strong interactions of subatomic physics, such as used in high energy and nuclear physics. MILC consists of a multiple codes for specific tasks. We used the "medium" NERSC MILC benchmark for the `su3_rmd` code. We varied the grid size for weak scaling from 16x16x16x16 on 16 processes to 32x32x32x16 on 128 processes keeping the grid as square as possible. MILC's communication overheads varied between 14.5% with 16 processes and 18.3% with 128 processes.

4.2.1 MPI Tracing Overhead

Our tracing library intercepts all MPI calls at the MPI profiling layer. Each wrapper call writes the absolute start and end time and full parameter list of the called function to a memory buffer. The buffer is flushed to disk by an asynchronous thread when it reaches a threshold. We left one core idle to run the writer thread and hide the disk I/O. With this technique, we achieved a profiling overhead of less than 0.1% for all runs.

4.2.2 Application Simulation Accuracy

We compared the runtime of Sweep3D with the simulation of the trace with the actual LogGPS parameters on Odin. Figure 11 shows the benchmarked runtime (dots) and the simulation results (lines). The average simulation error was below 2% in all experiments.

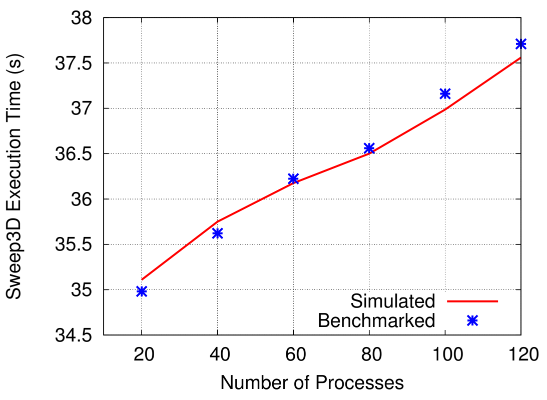

Figure 11: Sweep3D Simulation Results.

The runtime of MILC was also predicted accurately. Figure 12 shows the benchmark and simulation results for MILC on Odin.

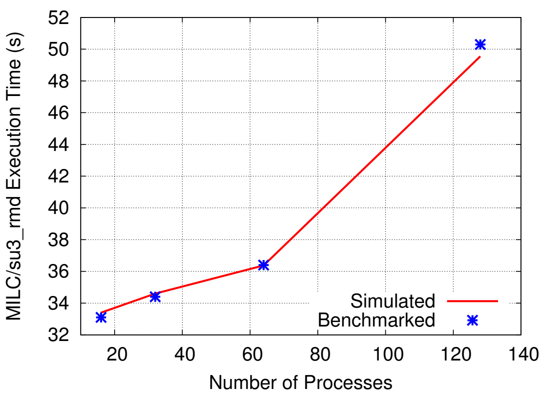

Figure 12: MILC/su3_rmd Simulation Results.

4.2.3 Extrapolating Application Traces

Our simulation tool chain also allows simple trace extrapolation to larger communicator sizes. The scheme simply copies point-to-point messages and renames source and target accordingly. Collective operations can be extrapolated accurately by simulating the collective algorithm at a larger scale, see Section 4.1.1. Figure 13 shows an example where a trace with two ranks and communications: allreduce, rank 0 sends to rank 1, allreduce, is extrapolated to 8 ranks. The horizontal lines show the time axis for each process and the vertical lines show messages. The point-to-point communication (solid lines in Figure 13) is copied while the two (dissemination) allreduce algorithms (dashed lines in Figure 13) are extrapolated without error. We note that this approach ignores application load imbalance and system effects (OS noise) which might become important at larger scale.

To test the accuracy of the extrapolation, we used a Sweep3D trace with 20 processes and extrapolated it to 40, 80, 100, and 120 processes. The extrapolation error (difference between benchmarked and extrapolated trace) was between 1.7% and 7.4%. A more detailed analysis shows that Sweep3D increases the number of point-to-point messages and their sizes with larger process counts (growing surface-to-volume ratio in our weak-scaling simulation). This effect is not modeled in our simple extrapolation, thus the extrapolation underestimates communication in Sweep3D

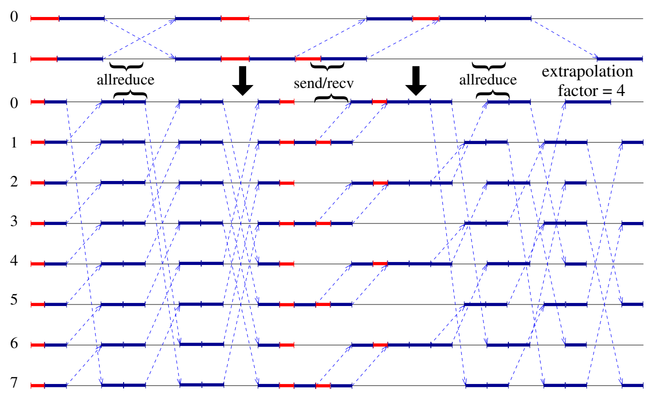

Figure 13: Schedule Extrapolation Example.

at scale. Another factor is that congestion might play a bigger role at scale but this is a problem in the abstract LogGOPS model, not the simulator.

The authors are aware that more accurate extrapolation and application evaluation schemes, such as used in BigSim, exist. However, such schemes often have different limits (memory consumption and runtime) and our simple trace extrapolation which accurately extrapolates collective operations and gives an estimation about point-to-point communication is a useful first approximation to the general problem of extrapolation.

4.2.4 *Application Simulation Performance*

The simulation performance of Sweep3D is shown in Figure 14. Simulating a benchmark running for 37.7 seconds with 120 processes on Odin takes 8.2 seconds (a simulation speedup of 552). As discussed before, the simulation time depends mostly on the number of simulated events (messages and locops) and our test-system processed 1,009,152 events/second.

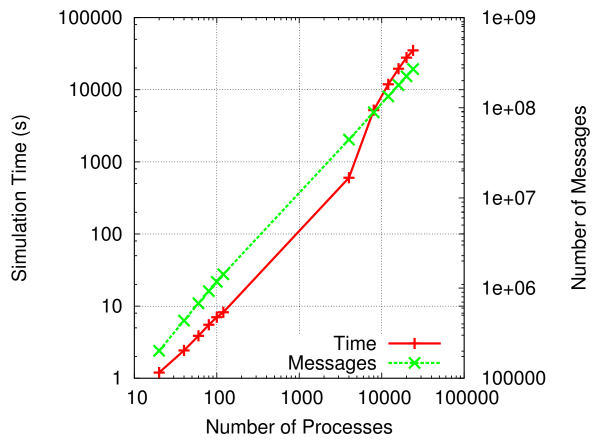

Figure 14: Sweep3D Simulation Times.

We extrapolated the run with 40 processes (0.4 million messages) to up to 28,000 processes (313 million messages). The execution time scales linearly with the number of messages up to 4,000 processes. Beginning from 8,000 simulated processes, the schedule exhausts the memory and the simulator works out-of-core (on local disk) which leads to the slowdown shown in Figure 14.

The simulated execution times indicate that Sweep3D and MILC scale well. However, we note that this is a lower bound because the extrapolation does not account for load imbalance and system effects (e.g., OS noise).

5. EXPLORATION OPPORTUNITIES

Our simulation offers now different avenues for exploring the behavior of parallel algorithms and applications with different LogGOPS parameters. One could investigate the isolated application's sensitivity to latency by varying L, to bandwidth by varying G (setting $O = 0$), or to message injection rate by varying g (setting $o = 0$). Another interesting test would be the application's potential to overlap communication and computation by setting $O = 0$ and $o = 0$ (ideal overlap) or $O = 0$ and o small (offloaded RDMA).

We analyzed the influence of four different changes to network parameters on the execution time for MILC and Sweep3D. We set $O = 0$ to model ideal RDMA overlap, $O = G$ to model no overlap, $G = 30$ to model a ten-times lower bandwidth and $L = 53000$ for a ten-times higher latency.

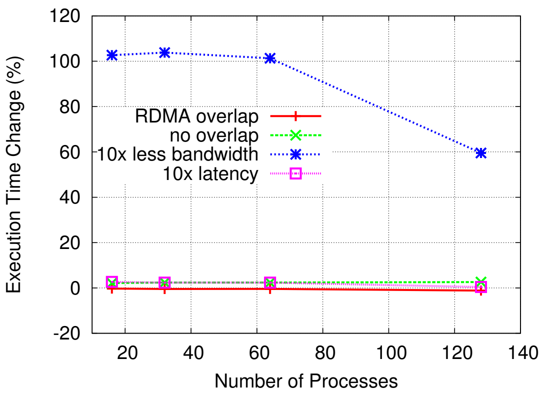

Figure 15: Influence of network parameters on MILC.

Figure 15 shows the influence of our changed parameters to MILC. We see that better RDMA overlap makes nearly no difference while no overlap reduces the performance slightly about 3%. This shows that the overlap capabilities (available computation during communication) seem to be exhausted with the current parameters. An increased latency has also nearly no effect on MILC indicating that the communication is bandwidth-bound. Decreasing the bandwidth causes indeed a significant slowdown of over 100% which supports the assumption that our traced MILC run is bandwidth bound.

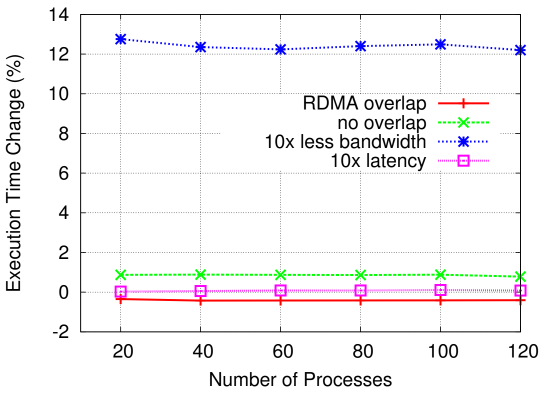

Figure 16: Influence of network parameters on Sweep3D.

Figure 16 shows the influence of the changed parameters to Sweep3D. The effect of overlap in the network is also marginal. Latency has nearly no impact while a ten-fold bandwidth-reduction slows the application by approximately 13%.

6. CONCLUSIONS AND FUTURE WORK

We proposed a simple simulator design for LogGOPS simulations and showed its efficiency at large-scale. We also showed that the simulations are very accurate on smaller systems and demonstrated the ability of the simulator to perform large-scale simulations with millions of processes in less than 30 minutes on a single CPU. Our simulator tool-chain can be downloaded from

`http://www.unixer.de/LogGOPSim`

The simulator allows easy experimenting with large-scale communication algorithms (collective operations and applications). All results in this paper merely show the opportunities for experimentation and we encourage readers to try simulations themselves.

We will further investigate application trace extrapolation techniques. We also plan to simulate the influence of operating system noise to applications at large scale with noise injection into the simulation.

Acknowledgments

We thank Steven Gottlieb (IU) and Greg Bauer (NCSA) for helpful discussions about the MILC code. This work was supported by the Department of Energy project FASTOS II (LAB 07-23).

7. REFERENCES

[1] V. S. Adve, R. Bagrodia, E. Deelman, T. Phan, and R. Sakellariou. Compiler-supported simulation of highly scalable parallel applications. In *SC'99: ACM/IEEE Supercomputing*.

[2] L. Albertsson. Holistic debugging – enabling instruction set simulation for software quality assurance. In *Intl. Symp. on Modeling, Analysis, and Simul. of Computer and Telecomm. Systems*, 2006.

[3] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman. LogGP: Incorporating Long Messages into the LogP Model. *Journal of Parallel and Distr. Computing*, 44(1):71–79, 1995.

[4] R. M. Badia, J. Labarta, and J. Gimenez. Dimemas: Predicting mpi applications behavior in grid environments. In *Workshop on Grid Applications and Programming Tools (GGF '03)*, 2003.

[5] R. Bagrodia, E. Deelman, and T. Phan. Parallel simulation of large-scale parallel applications. *Int. J. High Perform. Comput. Appl.*, 15(1):3–12, 2001.

[6] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: towards a realistic model of parallel computation. In *Princ. Pract. of Par. Progr.*, 1993.

[7] P. Geoffray and T. Hoefler. Adaptive Routing Strategies for Modern High Performance Networks. In *IEEE Symp. on High Perf. Interconnects*, 2008.

[8] S. Gottlieb, W. Liu, D. Toussaint, R. Renken, and R. Sugar. Hybrid-molecular-dynamics algorithms for the numerical simulation of quantum chromodynamics. *Physical Review D*, 35(8):2531–2542, 1987.

[9] D. Hengsen, R. Finkel, and U. Manber. Two Algorithms for Barrier Synchronization. *Int. J. Parallel Program.*, 17(1):1–17, 1988.

[10] M.-A. Hermanns, M. Geimer, F. Wolf, and B. Wylie. Verifying causality between distant performance phenomena in large-scale MPI applications. In *Conf. on Par., Distr., Network-Based Proc.*, 2009.

[11] T. Hoefler, A. Lichei, and W. Rehm. Low-Overhead LogGP Parameter Assessment for Modern Interconnection Networks. In *21st IEEE International Parallel & Distributed Processing Symposium*, 2007.

[12] T. Hoefler, T. Mehlan, F. Mietke, and W. Rehm. LogfP - A Model for small Messages in InfiniBand. In *20th Intl. Parallel and Distr. Proc. Symp. IPDPS'06 (PMEO-PDS 06)*, 2006.

[13] T. Hoefler, T. Schneider, and A. Lumsdaine. Accurately Measuring Collective Operations at Massive Scale. In *22nd IEEE Intl. Parallel & Distr. Processing Symposium (IPDPS, PMEO-PDS'06)*, 2008.

[14] T. Hoefler, T. Schneider, and A. Lumsdaine. Multistage Switches are not Crossbars: Effects of Static Routing in High-Performance Networks. In *2008 IEEE Intl. Conf. on Cluster Computing*, 2008.

[15] T. Hoefler, C. Siebert, and A. Lumsdaine. Group Operation Assembly Language - A Flexible Way to Express Collective Communication. In *ICPP-2009 - 38th Intl. Conf. on Parallel Processing*, 2009.

[16] D. Hower, L. Yen, M. Xu, M. Martin, D. Burger, and M. Hill. WWW Computer Architecture Page (Dec. 2009). `http://pages.cs.wisc.edu/~arch/www/tools.html`.

[17] F. Ino, N. Fujimoto, and K. Hagihara. LogGPS: A Parallel Computational Model for Synchronization Analysis. In *ACM Symposium on Principles and Practices of Parallel Progr.*, pages 133–142, 2001.

[18] T. Kielmann, H. E. Bal, and K. Verstoep. Fast Measurement of LogP Parameters for Message Passing Platforms. In *IPDPS Workshops on Parallel and Distr. Processing*, pages 1176–1183, 2000.

[19] K. Koch, R. Baker, and R. Alcouffe. Solution of the first order form of three-dimensional discrete ordinates equations on a massively parallel machine. In *Trans. of Am. Nucl. Soc.*, volume 65, 1992.

[20] F. Mietke, R. Baumgartl, R. Rex, T. Mehlan, T. Hoefler, and W. Rehm. Analysis of the Memory Registration Process in the Mellanox InfiniBand Software Stack. In *Euro-Par 2006*.

[21] C. A. Moritz and M. I. Frank. LoGPC: Modelling Network Contention in Message-Passing Programs. *IEEE Trans. on Parallel and Distributed Systems*, 12(4):404, 2001.

[22] MPI Forum. MPI: A Message-Passing Interface Standard. Version 2.2, September 4th 2009.

[23] G. Rodriguez, R. M. Badia, and J. Labarta. Generation of simple analytical models for message passing applications. In *Euro-Par 2004 Parallel Processing*, pages 183–188, 2004.

[24] D. Roweth and T. Jones. QsNetIII an Adaptively Routed Network for High Performance Computing. In *HOTI '08: 2008 16th IEEE Symp. on High Perf. Interconnects*, pages 157–164, 2008.

[25] R. Rugina and K. E. Schauser. Predicting the Running Times of Parallel Programs by Simulation. In *12th International Parallel Processing Symp.*, page 654, 1998.

[26] M. M. Tikir, M. A. Laurenzano, L. Carrington, and A. Snavely. PSINS: An Open Source Event Tracer and Execution Simulator for MPI Applications. In *Euro-Par '09*, pages 135–148, Berlin, Heidelberg, 2009. Springer-Verlag.

[27] G. Zheng, G. Kakulapati, and L. V. Kalé. Bigsim: A parallel simulator for performance prediction of extremely large parallel machines. In *18th Intl. Parallel and Distr. Proc. Symp. (IPDPS)*, page 78, April 2004.