Proceedings of the 2018 Winter Simulation Conference
M. Rabe, A. A. Juan, N. Mustafee, A. Skoogh, S. Jain, and B. Johansson, eds.

IMCSIM: PARAMETERIZED PERFORMANCE PREDICTION FOR IMPLICIT MONTE CARLO CODES

Gopinath Chennupathi
Stephan Eidenbenz
Alex Long
Olena Tkachenko
Joseph Zerr

Los Alamos National Laboratory
Los Alamos, NM, USA

Jason Liu

Florida International University
Miami, FL, USA

ABSTRACT

Monte Carlo techniques to radiation transport play a significant role in modeling complex astrophysical phenomena. In this paper, we design an application model (IMCSim) of an Implicit Monte Carlo (IMC) particle code using the Performance Prediction Toolkit (PPT), a discrete-event simulation-based modeling framework for predicting code performance on a large range of parallel platforms. We present validation results for IMCSim. We then use the fast parameter scanning that such a high-level loop-structure model of a complex code enables to predict optimal IMC parameter settings for interconnect latency hiding. We find that variations in interconnect bandwidth have a significant effect on optimal parameter values. Our results suggest potential value using IMCSim as a pre-step to substantial IMC runs to quickly identify optimal parameter values for the specific hardware platform on which IMC runs.

1 INTRODUCTION

The increasing algorithmic complexity of computational physics methods and the rapid emergence of novel hardware architectures with a focus on parallelism have turned predicting runtime performance of computational physics codes on these novel architectures an essential albeit, difficult issue to solve on our path to next generation computing. Performance prediction is among the standard requirements for next generation physics codes. While performance prediction efforts, such as the Structural Simulation Toolkit (SST) (Rodrigues et al. 2011) tend to put their focus on modeling aspects of the hardware stack, relatively simple-to-use performance prediction tools for domain application developers are largely lacking. In order to fill this gap, we have developed the Python-based Performance Prediction Toolkit (PPT) (Chennupati et al. 2017c) that enables a domain application developer to quickly test and predict runtime performance of algorithmic variations of their methods on novel architectures used in top-tier supercomputers.

In this paper, we present an application model for the Implicit Monte Carlo (IMC) code built in the PPT framework. The Implicit Monte Carlo method solves the thermal radiative transport equations by following individual photons; unlike merely spatial domain replicated Monte Carlo methods, we model an IMC code that processes photons in parallel and decomposes the physical domain into a (usually structured) mesh, where a cluster of neighboring mesh cells are assigned to an MPI (Message Passing Interface) rank or computational process and individual photons are passed off to other ranks through MPI messages if the photon crosses a mesh boundary of two mesh cells that are owned by different MPI ranks. We describe IMC and a relevant, associated mini-app "*Branson*" in Section 3.

An application model for IMC, which we call IMCSim (in the usual PPT naming convention), mimics the loop structure of IMC in a pseudo code style. It advances mimicked simulation time whenever the actual

code performs some computational kernel, by a number of seconds that depends on the hardware modeled, such as memory hierarchy latencies (Chennupati et al. 2017b), clock speeds, pipelining factors and so on. We arrive at these mimicked compute times through a mix of first-principle insights, low-level validation, and data fitting. IMCSim mimics MPI traffic on a per packet-level and arrives at a predicted runtime of the IMC code run on a specific input scenario on a specific hardware architecture as its main result. By design and desire for simplicity, IMCSim does not compute any physics or numerical results, but rather takes convergence parameters as input. IMCSim is parameterized with a large set of parameters that we group as follows: i) method parameters such as batch size and buffer size that trade-off the communication and computation mechanism through MPI ranks and MPI optimization parameters, ii) interconnect parameters such as topology, interconnect bandwidth and latencies, and iii) compute node parameters, such as number of cores per node, memory hierarchy access cycles, pipeline depths, etc. Exploring and optimizing within this parameter space is the main use case of IMCSim (described in Section 4).

We conduct baseline validation studies of IMCSim against a production IMC code, Branson, where we focus in particular on weak and strong scaling studies for up to 1024 MPI ranks on a 16 core per compute node (results are in Section 4.3). Interestingly, for strong scaling, IMCSim predicts IMC performance very well for core counts above 16, whereas smaller runs are slightly under-predicted with IMCSim, which we conjecture is due to a minor misconfiguration in the thread synchronization on the actual hardware platform for IMC. However, such a restriction is tolerable as we are mostly interested in large-scale runs.

We use IMCSim across a large set of method parameters in a largely factorial design. Specifically, we i) predict IMC runtimes between 1 and 1024 parallel ranks, ii) across four different interconnect topologies, including a more abstract interconnect where we study in detail the effect of (iii) communication latency and (iv) bandwidth on the overall IMC performance. Our focus is to study the two communication control parameters of *batchsize* and *buffersize*. Batchsize can be thought of as the number of protons to be processed by a rank before checking whether photons have been received from neighboring ranks. Buffersize is the number of photons that a rank sends to a neighbor at a time. We find that the optimum batchsize and buffersize value pairs depends on the instance size (number of photons) and the interconnect throughput.

2 PERFORMANCE PREDICTION

We describe the performance prediction toolkit (PPT) and the related work on performance modeling.

2.1 The Performance Prediction Toolkit (PPT)

IMCSim is an application model in the Performance Prediction Toolkit (PPT) (Chennupati et al. 2017c). PPT has been built as part of a computational co-design effort at Los Alamos. PPT has been at the heart of previously reported work (Ahmed et al. 2016b; Ahmed et al. 2016a; Chapuis et al. 2016; Zamora et al. 2016), see Eidenbenz and Zerr (2017) for an overview. PPT is a library of hardware, interconnect, middleware (such as MPI and OpenMP), and application models that can be interchanged and are typically parameterized. PPT allows for fast parameter scans that identify optimum algorithmic variation and hardware pairs. In particular, such parameter scans allow to find performance bottlenecks in terms of software and/or hardware resources. For example, PPT enables to characterize regimes or parameter spaces where a code is communication or computation bound. PPT is a lean model, very simple by design and intended to be used by the application developer. PPT relies on the Simian Parallel Discrete Simulation Engine (Santhi et al. 2015). For more computationally intense performance simulations, a Lua-based implementation is available as well, which achieves C-level speeds (or better) when combined with Lua's just-in-time compilation.

More precisely, PPT advances simulated wall clock time (e.g., the time it would have taken to execute the real application) when the application model uses hardware resources in a similar sense as a real CPU executes an instruction; however, PPT instructions are usually grouped into larger blocks of instructions that are executed repeatedly (but will need to be computed only once).

A PPT application model contains the loop structure of the original code as faithfully as possible. PPT calculates the time it would take to execute each body of the loop structure and then calculates the overall execution time taking into account thread-level, distributed memory, as well as instruction-level parallelism. This modeling approach works well for applications with a predictable, stencil-like, loop structure. Most computational physics codes fall into this category. For codes without near perfectly predictable structure, we build PPT application models that take distributions of loop iterations as input (obtained through testing of the real code or through first-principle domain expert knowledge) and result in more stochastic behavior. The key example from computational physics are convergence criteria, which the PPT model typically does not compute as it does not do actual numerical calculations; convergence rates or behavior is typically known theoretically or can be measured on small problem sizes.

An application domain expert should be able to write a PPT application model within a few days and then spend additional time implementing algorithmic variations. PPT application models typically consist of up to factor of 10 fewer lines of code than the mini-app version of the full-fledged physics code, with at least another factor of 10 between the mini-app and the full code. PPT achieves this efficiency through the use of Python as coding language with *Lua* and *JavaScript* as available alternatives and through simplified inputs focused on only parameters that affect the computational performance of select kernels. When combined with Just-in-time compilation, PPT achieves C-level speeds (Santhi et al. 2015).

2.2 Related Work

Performance modeling is an established field mostly within the hardware or systems community. Well-known efforts include the Intel x87 - oriented MARSS simulator (Patel et al. 2011) and the integrating framework of the Scalable Simulation Toolkit (SST) (Rodrigues et al. 2011). In similar vein, emulation and simulation prediction approaches exist, GEM5 (Binkert et al. 2011) emulates the CPU behavior in order to predict the runtimes. BigSim (Zheng et al. 2004) offers a trace driven simulation functionality in predicting the performance of large scale applications. Aspen (Spafford and Vetter 2012) is a domain-specific language for performance modeling that annotates the existing code. CODES (Cope et al. 2011) is another codesign simulator that enables multi-layered exascale storage architectures while exploring the design space of the exascale storage systems. xSim (Böhm and Engelmann 2011) is an parallel application performance prediction tool that runs an application in a controlled environment. Recently, Durango (Carothers et al. 2017) models the performance of extreme-scale applications on next generation supercomputers with the help of annotations similar to Aspen. PPT differs with the prediction attempts in the literature, while most of the above approaches don not scale with the applications whereas PPT scales better. Moreover, the main difference of PPT is the focus on application models, ease of use, leanness, and the use of Python and other interpreted languages together with JIT compilation considering the performance of the prediction tool.

3 BASICS OF IMPLICIT MONTE CARLO

Implicit Monte Carlo (IMC) is a standard method for solving the thermal radiative transfer equations (Fleck and Cummings 1971; Wollaber 2016). IMC simulations are computationally expensive relative to other physics codes in coupled multi-physics, so its performance is integral to fast turnaround in applicable simulations. IMC models heat transfer between matter and photon radiation with particles that represent groups of photons. The IMC equations without physical scattering for radiation and material energy are:

$$\frac{1}{c}\frac{\partial I(x,\nu,\Omega,t)}{\partial t} + \Omega\cdot\nabla I(x,\nu,\Omega,t) + \sigma_a^n(x,\nu,T)I(x,\nu,\Omega,t)$$
$$= \chi c\sigma_{a,p} f U_r^n + \chi(\nu)(1-f)\left(\int_0^\infty\int_0^{4\pi}\sigma_a^n(x,\nu,T)I(x,\nu,\Omega,t)\;d\Omega d\nu + S_m(x,t)\right), \quad (1)$$

and

$$\frac{U_m^{n+1}(x,T)-U_m^n(x,T)}{\Delta t} = \frac{1}{\Delta t}\int_0^{\infty}\int_0^{\infty}\int_0^{4\pi} f\sigma_a^n(x,\nu,T)I(x,\nu,\Omega,t)\; d\Omega d\nu dt$$

$$- fc\sigma_{a,p}(T)U_r^n + f\overline{S_m}(x,t). \tag{2}$$

Eq. (1) is solved with the Monte Carlo method and used to update the material energy via Eq. (2. The particles representing energy in the radiation field have probabilistic interactions with matter. These particles travel at the speed of light and deposit energy in the matter. The material is represented as a mesh with discrete temperatures and interaction probabilities in each mesh cell. The properties of particles are sampled from probability density functions in space, angle, birth time, and frequency. These stochastic properties, along with spatially varying temperature and interaction probability, present a computational pattern that is also stochastic—the data required for computing a particle history is not known *a priori*.

Stochastic particle histories also influence parallel algorithms for IMC. One instance of this occurs in parallel IMC simulations when the spatial mesh and physical data are too large to fit on one computational resource element and must be decomposed. This is generally handled by passing particles between spatial sub-domains via MPI messaging as they cross sub-domain boundaries. Because these crossing events are based on stochastic properties and interactions, it is not known beforehand how many parallel messages will be required and when in the simulation they will occur. This gives rise to two parameters for the parallel algorithm: how often to check for incoming particles from other sub-domains, called the *message check period* or *batch size*, and how many particles to send in a parallel message, called the *maximum buffer size*. Optimal settings for these parameters, along with other subtleties of the particle passing method, have been examined for several physical problems (Brunner et al. 2006; Brunner and Brantley 2009). A simple, grey IMC code Branson (Long 2017) is used to study parallel methods and parameters for IMC and here serves as the application used to validate the performance prediction tools. Figure 1 shows the flow of an IMC program within a time-step. The IMC method has many branching events that make predicting the number of events and the cost of simulating a particle difficult.

4 CREATING THE IMPLICIT MONTE CARLO SIMULATOR

We describe the two implementations of an IMC simulator: the former (IMCSim) mimics the exact functionality of IMC, while latter (IMC_Warp) extrapolates the knowledge of a single pass of MPI simulation.

4.1 Faithfully: IMCSim

IMCSim uses PPT's MPI model to implement each MPI rank, where Figure 1 succinctly describes the main function of a rank. IMCSim updates all state information in all the stages and determines the taken branches. IMCSim – being a PPT simulator – advances the simulated (i.e., predicted) wall clock time – at the central block *"run particle"* and the MPI call at the *"send/receive particles"* block.

The time advance of "run particle" is determined by a call to the detailed parameterized hardware model `time_compute(tasklist)` through a task list of base operations that include the number of floating point operations per particle, number of memory accesses, and number of divisions. As we are interested in both actual existing hardware but also want to study conceptual bottleneck resources, we allow IMCSim to run alternatively with having set the "run particle"-time as a parameter that the user can set. We note that we do not advance simulated time at the "process event"-stage even though the real IMC code does spend a few cycles in that stage (and even different amounts depending on what branch gets taken), however this time is typically dwarfed by the "run particle"-time.

The MPI calls in the "send/receive particles"-stage implement the corresponding MPI calls down to individual MPI data packets, leveraging the detailed interconnect models of PPT. While these simulation results are highly accurate, unfortunately, PPT/Simian even in parallel mode is computationally expensive.

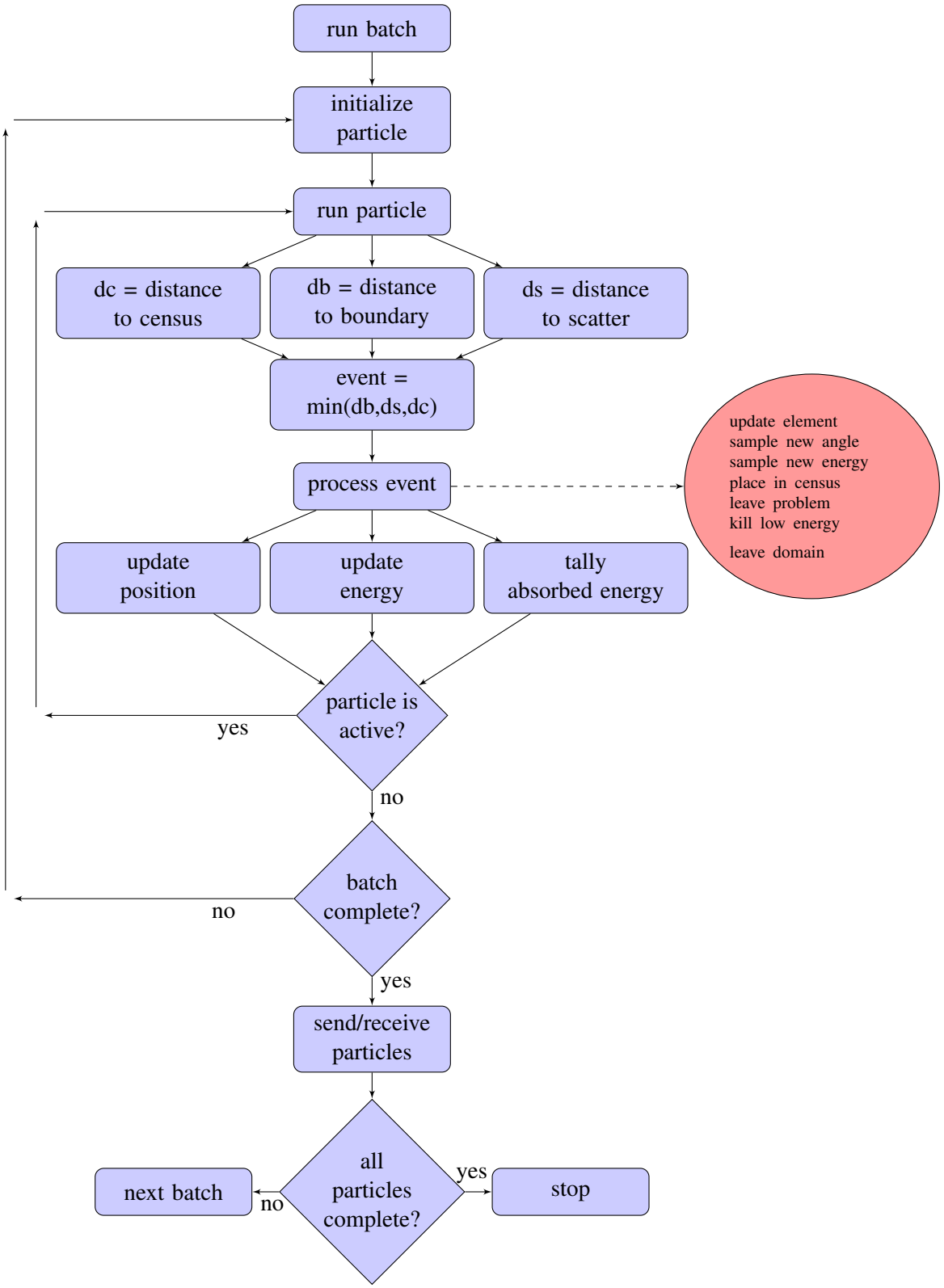

Figure 1: The main loop over IMC particles.

4.2 Fast: IMCSim_Warp

We used our detailed knowledge of the actual IMC method to reduce the runtime of IMCSim dramatically without sacrificing accuracy; we call the resulting model as IMSim_Warp. As IMC progresses particles can stop being tracked as they die or move for a time that corresponds with the end of a time step. Therefore while all particles may be tracked for much of a time step, a reduction occurs in the amount of work necessary over the time step. The key idea of reducing IMCSim runtime is to run for a single, but – importantly – complete pass of simulated MPI calls across all ranks, extrapolate from that in a time warp fashion to the stage just before the tail behavior of IMC (where batches and buffers are transmitted before they are full) begins, and then again execute the tail behavior faithfully. We collect statistics on the state variables (such as number of particles processed and batchsize) and the simulated wall clock time lapse statistics during the first phase. Based on the evolution of the state parameters, we know how many rounds of full buffers will be exchanged, and we know how long this will take. We thus warp ahead in time with a single operation (implemented as a single simulated MPI collective call) to the stage just before the tail behavior begins. The details of the tail behavior are not fully captured in Figure 1. Essentially, each MPI rank has to make sure that it does not wait too long to fill up a buffer with particles to be sent out when there simply are no longer enough active particles in the entire system. Detecting this state is not trivial and there are several algorithmic variations of how this is done. They each avoid the deadlock scenario where every rank is waiting for other ranks to send particles so it can fill up its own out-buffers. This tail behavior results in quite a few more MPI interactions, but at much smaller data volume, which significantly reduces the work for the PPT engine.

4.3 Validating IMCSim

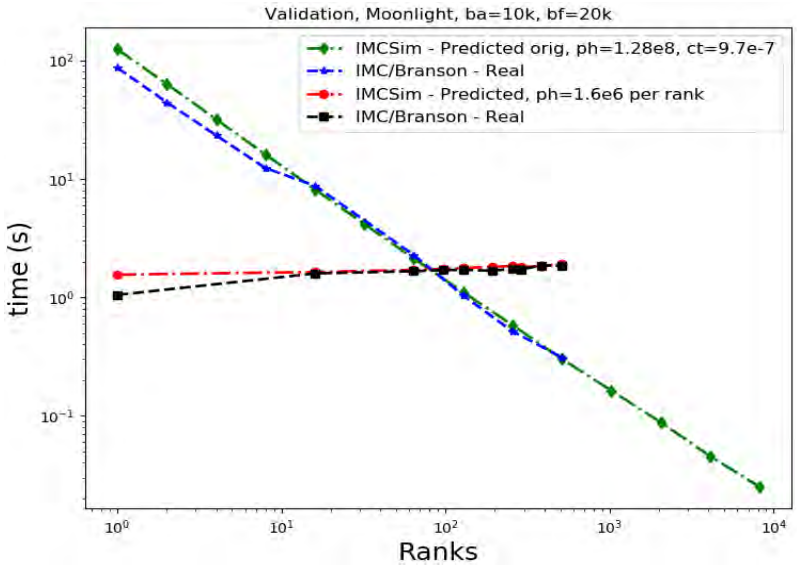

Figure 2: IMCSim Validation with Weak and Strong Scaling.

We validate IMCSim and IMCSim_Warp against actual Branson runs. We used Los Alamos' Moonlight cluster (now retired). Moonlight's 308 compute nodes comprised of 2×Eight-Core Intel Xeon E5-2670 sockets with 2.6 GHz cores. Our principle validation experiment involved a problem that used a buffersize of ten thousand particles and batchsize of twenty thousand particles. We used the "Hot Box" problem with 40x40x40 spatial cells with default MPI parameters. A strong scaling study simply dividing particles was performed with 128,000,000 photons. A weak scaling experiment was also performed with 1,600,000 photons per MPI process.

Figure 2 shows the validation results. We ran Branson in a strong-scaling and a weak scaling fashion for rank counts of 1, 2, 4, 8, 16, 32, 64, 128, 256, and 512 (we skipped a few smaller rank counts for weak scaling). In the weak scaling case, we doubled the number of particles (or photons) for each doubling of the rank count resulting in 1.6M photons per MPI rank. In case of strong scaling, we kept the number of photons constant at 128M and only increased rank counts. The results indicate that the runtimes predicted by IMCSim (blue and black) match the actual Branson runtimes (green and red) quite well, in particular at higher rank counts. Note, we only move off-node on the Moonlight-cluster at a rank count of 16, thus the MPI operations change quite drastically at rank count 16. We are interested in large-rank count behavior, which IMCSim captures very well, whereas smaller inconsistencies with Branson appear at low rank counts.

For the purposes of rapid scoping for broad trends, we find these results very satisfactory. IMCSim matches the scaling trends well. The inaccurate predictions at low rank counts do not represent typical use case for the codes of interest, that is, nodes are not under-subscribed. The inaccuracy certainly warrants further investigation, currently underway. The strong-scaling study was not extended to the point where we typically expect significant deviation from ideal, but additional studies should be conducted to investigate the predictive capacity of PPT.

5 COMPUTATIONAL EXPERIMENTAL RESULTS

Further investigations are conducted to exercise IMCSim for validation purposes and design space explorations. We investigate the effect of the batchsize and buffersize, the impact of network topology, and the effects of interconnect characteristics on latency and bandwidth.

5.1 Predicting the Effects of *Batchsize* and *Buffersize*

We study the ability of IMCsim to predict the effects and interplay of batchsize and buffersize parameters on Moonlight cluster. As described earlier, the batchsize parameter describes how many particles are processed before an MPI_receive is called and the buffersize parameter indicates how many processed particles need to be in the out-buffer before the buffer is purged and the particles are sent to the neighboring MPI ranks.

Figure 3 shows the validation results of batchsize (x-axis) versus buffersize at different values (500, 1000, 5000, 10000) for both the parameters. These results are measured for various simulated MPI ranks that range from 4, 8, 16, 32, 64, 128, 256 and 512 ranks. The color scheme indicates fast runtimes for dark colors and slower runtimes for brighter colors. We see that IMCSim predicts the effects of the two parameters relatively accurately, albeit it gets a few tiles off.

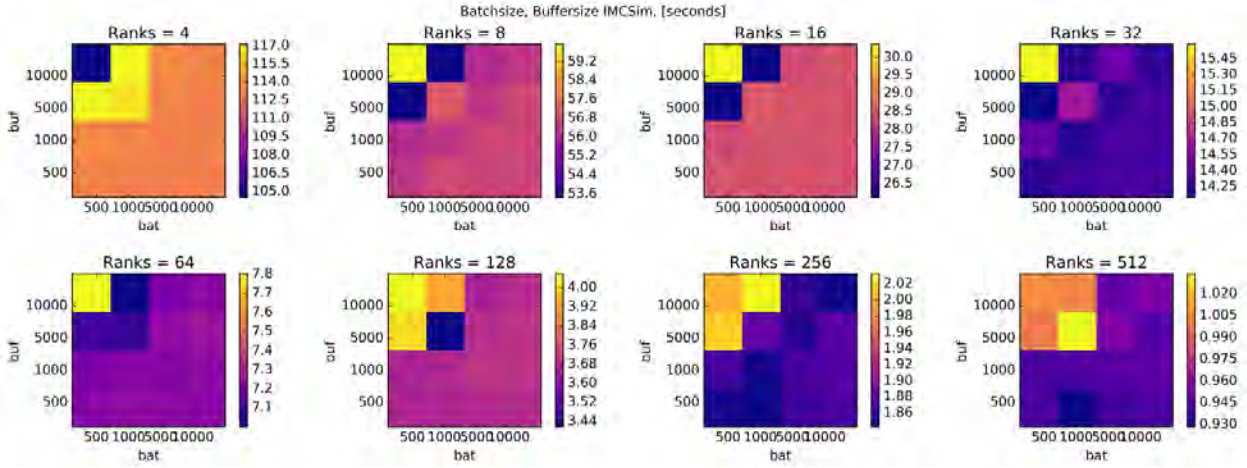

Figure 3: IMCSim batchsize versus buffersize at various MPI ranks that range from 4 to 512.

IMCSim allows quick exploration of batchsize and buffersize parameters across different rank counts, which can indicate to developers and users of Branson appropriate defaults are preferred and the values are tuned. In this particular experiment, we find the predicted effect to be relatively small, $\sim$20%.

5.2 Changing the Kernel Compute Time

We consider the "*what-if scenario*" whereby some architectural and/or software changes—that is, greater instruction-level parallelism on novel architectures, perhaps coupled with algorithmic kernel improvements—result in reduced compute times. With IMCSim of PPT, we can quickly evaluate the relative impact of these changes and how they would impact the scaling performance. We perform another strong scaling study for Moonlight and adjust the kernel compute time. Evident in Figure 4, at low rank counts where computation dominates, kernel compute time has significant impact. However, all curves flatten as expected when low kernel compute times are dwarfed by communication costs. Therefore, this helps in understanding how one can employ IMCSim kind of techniques in evaluating a given problem in improving the kernel time.

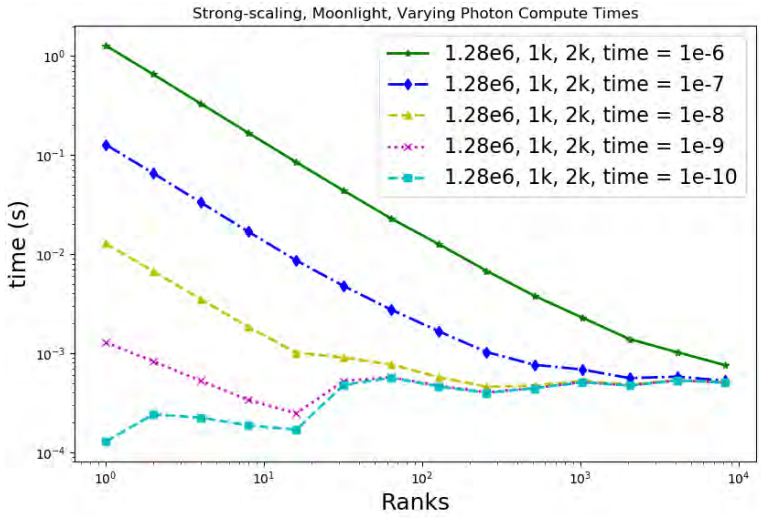

Figure 4: Varying the kernel compute time.

5.3 Predicting the Performance on Various Interconnects

We evaluate the effects of interconnect topology and communication parameters. We employ different simulated interconnect models on a given compute node (and kernel computational time consequently) in analyzing the communication patterns of different interconnect topologies. Figure 5 shows, the Titan model exhibits significant deviation from other topologies starting at 64 ranks, an issue with our model currently under investigation. However, other platforms compare well with each other even at 512 ranks, all within 20% of each other. On the other hand, Figure 6 communication time, where we set the compute time specifically to 0.0 seconds in order to observe the effects of communication time. We find that some systems, those that are fat tree in design, tend to lose performance at 32 ranks, but recover at higher counts. Other systems tend to degrade at higher ranks. Note, the communication pattern of IMCSim demonstrates the ability of our tool to consider the impact different topologies will have on the actual simulations.

5.3.1 Interconnect Bandwidth and Latency

We investigate the impact of different message passing latency times and interconnect bandwidths in an idealized crossbar topology. For this problem, from Figure 7, it is clear that message latency does not

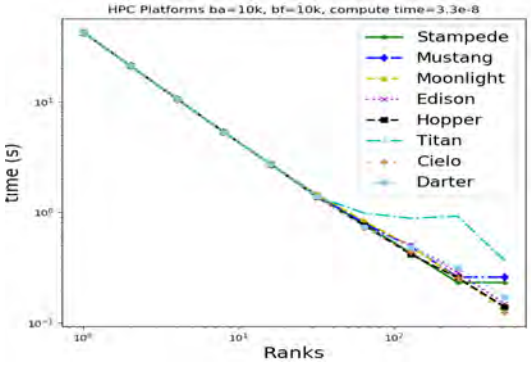

Figure 5: Shared compute node type.

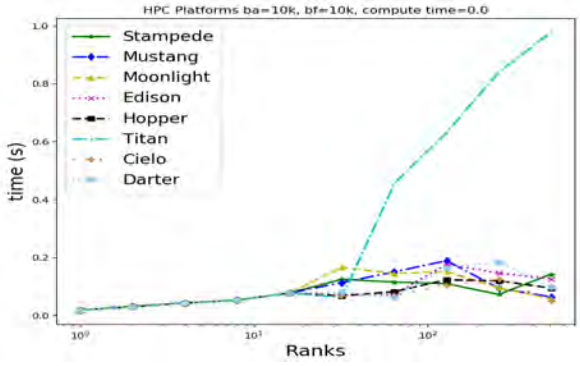

Figure 6: Communication time only.

heavily impact the predicted simulation time. However, increasing interconnect bandwidth shows notable reductions in expected runtime. We note a realistic value for bandwidth is on the order of tens of gigabytes (GB) per seconds. Other problems that are more latency-sensitive would likely demonstrate different characteristics. Another advantage with IMCSim is that we are able to quickly explore the effects of different problem settings on existing architectures before launching large calculations on real machines.

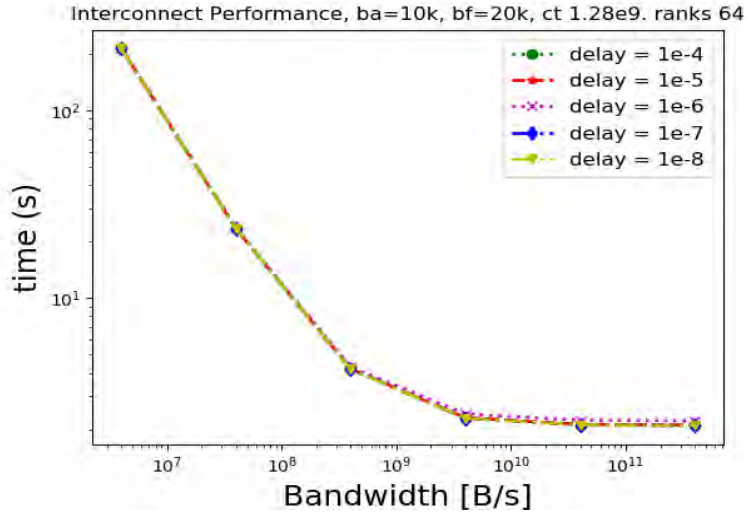

Figure 7: The influence of latencies and bandwidths on predicted runtimes.

5.4 Scaling of Parallel Discrete Event Simulation Engine

Along with the above experiments, we are interested in studying the scalability of IMCSim and PPT. While the simulator will allow one to predict the performance of a code on a very large allocation of computing resources, for which one often must wait, using a much smaller set of resources for the simulator itself may require some time to compute this prediction. Figure 8 shows the strong scaling trend of IMCSim with our discrete event simulation engine, Simian (using a Python variant). We simulated Moonlight-like ranks for several different counts decomposing a 3D spatial mesh. The simulation was then run using an increasing number of actual ranks (2, 4, 8, 16, 32, and so on up to 4096) in a strong-scaling study. Insufficient memory for the number of entities accounts for the two largest simulated jobs. Local minima reveal the best balance of computation by the simulator with inherent communication. Flat behavior is seen at higher physical rank counts, because the PPT model reaches one entity per physical rank. Overall the results

are expected, but they do indicate instances of some desire to improve the performance of IMCSim itself, not just its accuracy. Beyond the software simplifications of IMCSim_Warp (described in Section 4.2), alternative variants of Simian can improve simulator turnaround time.

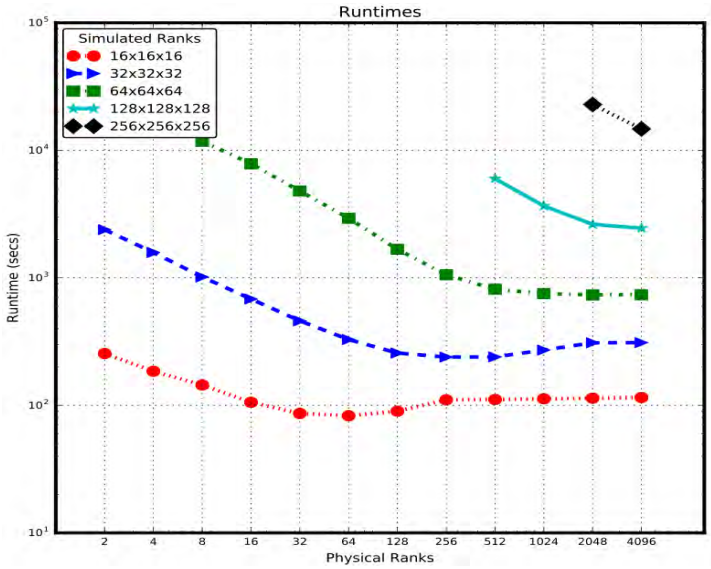

Figure 8: Scaling of parallel discrete event simulation engine, SimianPie (a Python variation of Simian).

6 CONCLUSION

We have presented a new prediction technique for implicit monte carlo methods that use domain and particle decomposition schemes for parallelism. IMCSim has been incorporated into the Performance Prediction Toolkit, which uses integrated, abstract hardware, middleware, and software performance prediction models. Ease of use is a key design criteria for our techniques. And scalability is afforded through just-in-time compilation engines, memory models (Chennupati et al. 2017a) and branch predictions (Kalla et al. 2017).

We demonstrated the flexibility and utility of IMCSim to evaluate performance under conditions, ranging from input parameters to hardware configurations. We witness expected results for broad trends and interesting insights for finer details on our codes perform under various conditions.

Future work includes continued refinement and validation of our models, seeking better understanding of differences between actual and predicted performance. Moreover, uncertainty quantification and variance estimates can be applied to our results to better indicate if the differences are within the ranges of conceivable statistical overlap. We can implement more interesting algorithmic methods, including mesh-passing to improve load balance. We can improve the MPI model, similar to Obaida et al. (2018). Finally, we consider additional node architectures, most notably GPGPUs.

REFERENCES

Ahmed, K., J. Liu, S. Eidenbenz, and J. Zerr. 2016a. "Scalable Interconnection Network Models for Rapid Performance Prediction of HPC Applications". In *Proceedings of the 18th International Conference on High Performance Computing and Communications*, edited by J. Chen and L. T. Yang, 1069–1078. Exeter, UK: IEEE.

Ahmed, K., M. Obaida, J. Liu, S. Eidenbenz, N. Santhi, and G. Chapuis. 2016b. "An Integrated Interconnection Network Model for Large-scale Performance Prediction". In *Proceedings of the 2016 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, edited by R. Fujimoto et al., 177–187. Banff, Alberta, Canada: ACM.

Binkert, N., B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. 2011. "The Gem5 Simulator". *SIGARCH Comput. Archit. News* 39(2):1–7.

Böhm, S., and C. Engelmann. 2011. "xSim: The Extreme-Scale Simulator". In *Proceedings of the 2011 International Conference on High Performance Computing and Simulation*, edited by W. W. Smari and J. P. McIntire, 280–286. Istanbul, Turkey: IEEE.

Brunner, T. A., and P. S. Brantley. 2009. "An Efficient, Robust, Domain-Decomposition Algorithm for Particle Monte Carlo". *Journal of Computational Physics* 228(10):3882–3890.

Brunner, T. A., T. J. Urbatsch, T. M. Evans, and N. A. Gentile. 2006. "Comparison of Four Parallel Algorithms for Domain Decomposed Implicit Monte Carlo". *Journal of Computational Physics* 212(2):527–539.

Carothers, C. D., J. S. Meredith, M. P. Blanco, J. S. Vetter, M. Mubarak, J. LaPre, and S. Moore. 2017. "Durango: Scalable Synthetic Workload Generation for Extreme-Scale Application Performance Modeling and Simulation". In *Proceedings of the 2017 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, edited by W. Cai et al., 97–108. NTU, Singapore: ACM.

Chapuis, G., S. Eidenbenz, and N. Santhi. 2016. "GPU Performance Prediction Through Parallel Discrete Event Simulation and Common Sense". In *Proceedings of the 9th EAI International Conference on Performance Evaluation Methodologies and Tools*, edited by W. Knottenbelt and K. Wolter, 204–211. Berlin, Germany: ACM.

Chennupati, G., N. Santhi, R. Bird, S. Thulasidasan, A. A. Badawy, S. Misra, and S. Eidenbenz. 2017a. "A Scalable Analytical Memory Model for CPU Performance Prediction". In *Proceedings of the 8th International Workshop on High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation, PMBS*, edited by S. Jarvis et al., 114–135. Denver, CO, USA.

Chennupati, G., N. Santhi, S. Eidenbenz, and S. Thulasidasan. 2017b. "An Analytical Memory Hierarchy Model for Performance Prediction". In *Proceedings of the 2017 Winter Simulation Conference*, edited by W. K. V. Chan et al., 908–919. Piscataway, New Jersey: IEEE.

Chennupati, G., N. Santhi, S. Eidenbenz, R. J. Zerr, M. Rosa, R. J. Zamora, E. J. Park, B. T. Nadiga, J. Liu, K. Ahmed, and M. A. Obaida. 2017c. *Performance Prediction Toolkit (PPT)*. Los Alamos National Laboratory (LANL). https://github.com/lanl/PPT.

Cope, J., N. Liu, S. Lang, P. Carns, C. Carothers, and R. Ross. 2011. "Codes: Enabling Co-design of Multilayer Exascale Storage Architectures". In *Proceedings of the Workshop on Emerging Supercomputing Technologies*, edited by B. C. Lee and M. Själander, Volume 2011. Tucson, AZ, USA: ACM.

Eidenbenz, S. J., and R. J. Zerr. 2017. "Codesign Performance Prediction for Computational Physics 3rd Year Review Overview talk". Technical report, LANL. https://permalink.lanl.gov/object/tr?what=info:lanl-repo/lareport/LA-UR-17-20796.

Fleck, J. A., and J. D. Cummings. 1971. "An Implicit Monte Carlo scheme for calculating time and frequency dependent nonlinear radiation transport". *Journal of Computational Physics* 8:313–342.

Kalla, B., N. Santhi, A. H. A. Badawy, G. Chennupati, and S. Eidenbenz. 2017. "Probabilistic Monte Carlo simulations for static branch prediction". In *Proceedings of the 36th International Performance Computing and Communications Conference (IPCCC)*, edited by M. Wang et al., 1–4. San Diego, CA, USA: IEEE.

Long, A. R. 2017. "Domain Decomposed Parallel Implicit Monte Carlo with the Data Server Model". In *Proceedings of ANS M&C 2017*. Jeju, South Korea.

Obaida, M., J. Liu, G. Chennupati, N. Santhi, and S. Eidenbenz. 2018. "Parallel Application Performance Prediction Using Analysis Based Models and HPC Simulations". In *Proceedings of the Annual Conference on SIGSIM Principles of Advanced Discrete Simulation*, edited by F. Quaglia et al., 49–59. Rome, Italy: ACM.

Patel, A., F. Afram, S. Chen, and K. Ghose. 2011. "MARSS: A Full System Simulator for Multicore x86 CPUs". In *Proceedings of the 48th Design Automation Conference*, edited by L. Stok et., 1050–1055. San Diego, CA, USA: ACM.

Rodrigues, A. F., K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. CooperBalls et al. 2011. "The Structural Simulation Toolkit". *ACM SIGMETRICS Performance Evaluation Review* 38(4):37–42.

Santhi, N., S. Eidenbenz, and J. Liu. 2015. "The Simian Concept: Parallel Discrete Event Simulation with Interpreted Languages and Just-in-time Compilation". In *Proceedings of the 2015 Winter Simulation Conference*, edited by C. Macal et al., 3013–3024. Piscataway, New Jersey: IEEE.

Spafford, K. L., and J. S. Vetter. 2012. "Aspen: A Domain Specific Language for Performance Modeling". In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, edited by J. K. Hollingsworth, 1–11. Salt Lake City, UT, USA: IEEE.

Wollaber, A. B. 2016. "Four Decades of Implicit Monte Carlo". *Journal of Computational and Theoretical Transport* 45(1–2):1–70.

Zamora, R. J., A. F. Voter, D. Perez, N. Santhi, S. M. Mniszewski, S. Thulasidasan, and S. J. Eidenbenz. 2016. "Discrete Event Performance Prediction of Speculatively Parallel Temperature-accelerated Dynamics". *Simulation* 92(12):1065–1086.

Zheng, G., G. Kakulapati, and L. V. Kale. 2004. "BigSim: A Parallel Simulator for Performance Prediction of Extremely Large Parallel Machines". In *Proceedings of the 18th International Parallel and Distributed Processing Symposium*, edited by B. Monein, 78–87. Santa Fe, NM, USA: IEEE.

AUTHOR BIOGRAPHIES

GOPINATH CHENNUPATI is a Post-doctoral Research Associate with the Information Sciences Group (CCS-3) at Los Alamos National Laboratory, New Mexico, United States of America. He holds a Ph.D. in computer science from University of Limerick, Ireland. His research interests include parallel discrete event simulation, performance modeling, high performance computing, machine learning, and artificial intelligence. His email address is gchennupati@lanl.gov.

STEPHAN EIDENBENZ is the Director of the Information Science and Technology (ISTI) institute at Los Alamos National Laboratory. He obtained a PhD from the Swiss Federal Institute of Technology, Zurich (ETHZ) in Computer Science. His research interests include cyber security, computational co-design, communication networks, scalable modeling and simulation, and theoretical computer science. His email address is eidenben@lanl.gov.

ALEX LONG is a staff scientist in the Computational Physics and Methods (CCS-2) group at Los Alamos Laboratory. He specializes in parallel methods for particle transport and optimization techniques for Monte Carlo. His email address is along@lanl.gov.

Olena Tkachenko was a post-baccalaureate in CCS-3 and ISTI as well as the Ultrascale Systems Research Center at LANL. Her research interests are in performance modeling, high performance computing, computer networks and using machine learning on HPC architecture systems dealing with fault mitigation. Her e-mail address is otkac001@fiu.edu.

JOSEPH ZERR is a staff scientist in the Computational Physics and Methods (CCS-2) group in CCS-2 working in the linear, deterministic transport code team. He specializes in parallel methods for particle transport using deterministic methods applied to structured spatial grids. His email address is rzerr@lanl.gov.

JASON LIU is an Associate Professor at the School of Computing and Information Sciences, Florida International University. He received a Ph.D. degree from Dartmouth College in Computer Science. His research focuses on parallel simulation and high-performance modeling of computer systems and communication networks. His email address is liux@cis.fiu.edu.