Proceedings of the 2016 Winter Simulation Conference
T. M. K. Roeder, P. I. Frazier, R. Szechtman, E. Zhou, T. Huschka, and S. E. Chick, eds.

PREDICTING PERFORMANCE OF SMOOTHED PARTICLE HYDRODYNAMICS CODES AT LARGE SCALES

Guillaume Chapuis
David Nicholaeff
Stephan Eidenbenz
Robert S. Pavel

Los Alamos National Laboratory
Bikini Atoll Rd., SM 30
Los Alamos, NM 87545, USA

ABSTRACT

We present performance prediction studies and trade-offs of Smoothed Particle Hydrodynamics (SPH) codes that rely on a Hashed OctTree data structure to efficiently respond to neighborhood queries. We use the Performance Prediction Toolkit (PPT) to (i) build a loop-structure model (SPHSim) of an SPH code, where parameters capture the specific physics of the problem and method controls that SPH offers, (ii) validate SPHSim against SPH runs on mid-range clusters, (iii) show strong- and weak-scaling results for SPHSim, which test the underlying discrete simulation engine, and (iv) use SPHSim to run design parameter scans showing trade-offs of interconnect latency and physics computation costs across a wide range of values for physics, method and hardware parameters. SPHSim is intended to be a computational physicist tool to quickly predict the performance of novel algorithmic ideas on novel exascale-style hardware such as GPUs with a focus on extreme parallelism.

1 INTRODUCTION

As we move towards exascale computing, predicting performance of computational physics methods and codes becomes both more crucial and more challenging: more crucial because the life cycle of a super computer machine with its specific architecture is limited to about four years, thus any code refactoring or other preparation necessary to achieve good performance needs to be completed ideally before the end of the first full year of operation making early stage performance prediction and assessment of code variations indispensable; more challenging because traditional scaling laws will continue to collapse demanding every more complex code adaptations to emerging accelerator-type hardware and extreme parallelism on instruction and thread-level coupled with advanced hardware features, such as pipe-lining, non-uniform-memory access, adaptive clock-speeds, and cache prefetching.

Computational co-design – the principle of designing software to exploit hardware features and (ideally) vice-versa – is a cornerstone of the US Department of Energy's (DOE) path towards exascale (Reed and Dongarra 2015). Early performance prediction of DOE-relevant,co-designed code on emerging, but not yet existing exascale architectures is a key tool for exascale preparedness as it will allow quick testing of ideas, early dismissal of algorithmic ideas that do not scale, and optimized hardware software pairings. Los Alamos National Laboratory's Performance Prediction Toolkit (PPT) is a co-design tool for exascale performance prediction that mimics the hardware and software ecosphere through modeling the loop structure of the *application software, the abstracted functionality of* middleware tools such as the OS and the programming model (e.g., MPI, Charm++, Legion, Chapel), and parametric hardware models (clock speed, memory size, etc.). At its core, PPT relies on a scalable parallel discrete event simulation engine Simian (Santhi, Eidenbenz, and Liu 2015). PPT allows a (computational physics) application developer to

978-1-5090-4486-3/16/$31.00 ©2016 IEEE

quickly assess algorithmic ideas on emerging software and hardware stacks. PPT models are implemented in either Python or Lua to allow for very fast prototyping. The PPT concept has been used to identify 3x (Mniszewski, Junghans, Voter, Perez, and Eidenbenz 2015) and 100x speed-ups (Zamora, Uberuaga, Perez, and Voter 2016) in algorithmic variations of Accelerated Molecular Dynamics (AMD) methods in molecular dynamics. We present a more thorough overview of PPT in Section 2.

In this paper, we present a PPT application model of Smoothed Particle Hydrodynamics (SPH) codes that we call SPHSim. Particularly when coupled to a Hashed Oct-Tree data structure for efficient neighborhood identification, SPH codes stand out among computational physics codes for their ability to hide latency. The SPH method (Fryer, Rockefeller, and Warren 2006) has been developed for astro-physics simulation and has been a work horse in this field, having been used in some of the largest physics computations ever, namely the Dark Sky Simulations using the 2HOT code (Warren 2013). We give a more detailed overview of SPH in Section 3.1.

SPHSim models the loop structure of SPH codes by mimicking the two main CPU cycle consumers of SPH, namely OctTree accesses and actual physics interaction kernel computations (such as gravity between two cells), by simply advancing simulated time by an appropriate amount as opposed to actually computing data. SPHSim implements SPH's key features such as the number of tree accesses, the number of interactions to compute, the distribution of particles over MPI ranks, by drawing random numbers from stochastic distributions that we obtained from a mix of code instrumentation and mathematical analysis of SPH. The SPHSim code is a brief 300-line Lua program that is easy to comprehend, particularly when compared to the original highly optimized SNSPH code of 100,000 lines of C. SPHSim is controlled by input parameters that encode the physics instance (e.g., number of particles, average interactions per particle), a few SPH method parameters (e.g., number of MPI ranks), and hardware parameters (e.g., clock speed, interconnect latency). We present SPHSim in more detail in Section 3.2, give validation results of SPHSim against actual SPH runs in Section 4, and discuss the weak and strong scaling properties of SPHSim in Section 5.

Our main SPHSim results (Section 6) are presented in the form of parameter scans with a key focus on studying SPH's latency-hiding abilities. By varying the interconnect latency, the interaction compute time, and software parameters, we identify software trade-offs and hardware requirements.

2 OVERVIEW OF THE PERFORMANCE PREDICTION TOOLKIT (PPT)

The Performance Prediction Toolkit (PPT) is a library of models of computational physics application codes, middleware, and hardware that allows users to predict runtime and energy consumption by running stylized pseudo-code implementations of physics applications.

PPT relies on the scalable Simian Parallel Discrete Event Simulation engine in Python or Lua using Simian's process concept. Each computing unit (such as a host, compute node, or core) is a Simian entity. An application is modeled as a computing process in Simian or thread associated with an entity. Processes can communicate with each other by exchanging messages; processes can be active, sleep, be woken up, begin and end. Application processes can advance simulated time to advance by calling a compute-function with a task list as input parameter, where a task list consists of a set of commands to be executed by the hardware. The compute-function advances simulated time by the amount it would take to execute the task list. An example of a task list could be

$$500 ALU operations, 350 memory accesses, etc.$$

.

A PPT application is a stylized version of the actual application that looks like pseudo-code written in Python or Lua capturing the loop structure of the code. PPT models typically do not predict numerical accuracy of an application algorithm but rather predict time and energy consumption. This design choice allows for much simpler implementation reproducing most loop structures of computational physics codes

well with their predictable for-loops, but it does require explicit values for more complex loop exit conditions as they sometimes occur in convergence criteria.

PPT middleware models, such as MPI, offer an API to PPT application models and implement the underlying functionality in the loop structure modeling spirit of of PPT application models.

PPT hardware models exist for interconnects and for compute nodes. They are highly parameter-oriented, such as clock-speed, cache-level access times, memory bandwidth, etc. and mimic lower level hardware processes, such as caching strategies, prefetching and pipe-lining through functions learned from regression or other machine learning models trained on hardware counter input data.

PPT models are highly parameterized for applications, middleware, and hardware models, allowing parameter scans to optimize parameter values for hardware-middleware-software pairings. Because PPT often only need to execute a few of the innermost loops of large physics codes, PPT models are comparable in speed to real code or faster, but allow of course almost unlimited instrumentation and bottleneck analysis.

The PPT hardware library contains – among others – models for AMD Opterons, Haswell, Knights-Landing processors among others as well as the nvidia GPUs K20X, K40, K6000, M2090, and Pascal. We have interconnect models for Gemini 3D torus and butterfly networks. These building blocks can be used to build cluster models for LANL's Cielo machine and ORNL's Titan machine. PPT has a very detailed MPI middleware model, an OpenMP model is under development. PPT applications include benchmarks such as the Polybench suite and physics application code models for radiation transport (the SNAP/PARTISN code), accelerated molecular dynamics, and – as presented in this paper – smoother particle hydrodynamics.

3 MODELING SMOOTHED PARTICLE HYDRODYNAMICS CODES: SPHSIM

3.1 Introduction to Smoothed Particle Hydrodynamics

Smoothed-Particle Hydrodynamics (SPH) is a numerical method for solving the equations of motion governing material bodies in their frame of reference, i.e. the Lagrangian frame, when the distortions are large. Unlike other standard numerical methods, there is no underlying "mesh" structure which discretizes the problem domain's geometry. Whereas a computational mesh defines the locations of idealized mathematical discretization points (or edges, areas, and volumes) to which field values, such as mass, momentum, pressure, or internal energy, can be prescribed, implying some kind of regular geometrical order, the points in a SPH simulation are initially placed according to the mass distribution and thereafter move in response to the prescribed physics.

Derivatives in the governing equations are approximated by weighted sums of kernels. Neighbor sets, which effectively correspond to "stencils" on meshes, are determined by the overlap of the kernels, instead of by drawing fixed lines between points. The method is particularly attractive for the dynamics of fluids and solids when the forces are very large, such as those caused by explosion and impact. Acting as an essentially free-Lagrangian simulation for all timesteps, the method suffers no ill effects caused by the approximations normally required to mix different-phase materials in a computational mesh cell.

3.2 Statistical Analysis

As mentioned above, SPH allows the simulation to essentially place the idealized discretization points at random within the physical domain. This makes computing neighbor sets nontrivial. Further, a naïve algorithm would parse through the entire container of particles to find the relevant neighbors thereby emitting $O(n^2)$ time complexity where n is the number of particles. The solution adopted by the community has been to use tree-based algorithms to order the particles hierarchically, reducing the time complexity down to $O(n \log n)$. To account for the distributed nature of high performance compute clusters, HOT and 2HOT(Warren and Salmon 1995, Warren 2013) make use of a hashed OctTree in order to provide a global address space which is optimized to keep spatially local particles, as well as their parent cells in the OctTree, contiguous in memory.

Using a tree, calculating long range forces of particles on each other becomes a sum of terms, for which acceptable nodes in the tree are approximated as multipole expansions. Walking the tree is coupled with a multipole acceptance criterion (MAC), thereby constraining the necessary number of particles needed in a neighbor set as dictated by the physical phenomenon being modeled and the desired order of accuracy. Further, an asynchronous MPI facility becomes a natural extension of walking the tree, since if necessary neighbor particles for interactions are not present, then the interaction kernel computation can be deferred until the data arrives while further interactions are computed.

While this facility has proven its robustness, running a 1 trillion particle simulation on 256K cores (Skillman, Warren, Turk, Wechsler, Holz, and Sutter 2014), it makes a deterministic model simulation effectively intractable. From the initial conditions of the particle layout to the physics kernels simulated (and subsequently the MAC), every aspect of the simulation, thought of as parameters, will influence runtime in a manner highly susceptible to perturbations in their value. Consequently, to model an SPH code making use of an asymmetric hierarchical structure, we identify five random variables to be informed by past experimental data from large particle simulations.

In 2HOT, walking the tree and computing the physics kernels are the dominant consumers of runtime. As such, and in consideration of the topology of the hashed OctTree, we stochastically model the tree with the following random variables:

1. #particles/rank: Because work load balancing automatically weights particles based on the number of interactions per particle from a previous iteration, the number of particles per rank can fluctuate leading to asymmetric MPI communication from a volume of data perspective.
2. #interactions/particle: This is strongly dependent on the physics in both kernels computed as well as initial and boundary conditions, e.g. the clustering of particles can increase rapidly for phenomenon such as shocks.
3. #accesses/interaction: As with the last random variable, the geometry of the problem specification and clustering of particles can have a drastic impact on runtime. Higher clustering correlates with deeper traversals in the tree.
4. #linkedListJumps/access: 2HOT's hash table representation of the OctTree can have collisions due to overlapping keys when the least significant bits are taken. As such, a traversal is necessary to find the relevant body or cell in a particular "hash bucket", albeit this data is still laid out contiguously in memory.
5. #requests/rank: This is correlated with the type of access, namely whether particles are local, deferred, or remote. This is the source of the MPI asynchronicity. Further, by identifying this random variable, there is an implicit assumption about 2HOT's ability to maximally keep the work and memory pipelines saturated concurrently.

3.3 Sedov Test Problem

For the validation problem presented below, we ran a weak scaling study of a Sedov problem, i.e. a spherical shock wave. The standard Monaghan kernel(Monaghan 1992) was used with diffusion turned off, artificial heat conduction turned on, and artificial viscosity and energy dissipation both dependent on runtime values. We ran at 50k particles per MPI rank across 32, 64, 128, and 256 ranks.

The random variables, in consideration of the particular problem, were set as follows:

1. #particles/rank: Perturbing around the #particles/#MPI ranks using a small value on the order of $\log n$, where n is the number of particles, is sufficient for the Sedov problem given the symmetry of the problem and subsequent uniform partitioning of particles across ranks.
2. #interactions/particle: In the Sedov problem, there is a large amount of clustering of particles near the shock, but a high degree of sparsity in coarser resolved regions outside of the shock. We take

a Gaussian distribution around $\frac{1}{2}\log n$ (the depth of the tree divided by 2) with standard deviation $\frac{1}{4}\log n$.

3. #accesses/interaction: A Poisson distribution with $\lambda = 3$ was chosen given the low number of collisions and the regular symmetry of our test problem. Namely, it weights well for finding a particle within an oct of the same size, smaller size, and one size smaller still.
4. #linkedListJumps/access: This variable was turned off because 2HOT moves recently visited bodies to the front of the queue, and given the relatively untaxing test problem of 100k particles per rank (i.e. a lower number of collision in the hash table), the impact on runtime is small compared to the #accesses per interaction.
5. #requests/rank: Again appealing to the symmetry of the problem and assuming an ideal surface area to volume ratio, we take a value of $6n^{2/3}$. The cube root of the number of particles corresponds to an edge length, square that to get a surface area, and then six faces for a cube in 3-D space.

The goal of our simulator is to simulate the real application either faster than running the real code or using less machines, while allowing users to change software and hardware parameters more easily than editing the application or having it run on a different machine. We use the above described random variables to simulate tree traversals on each rank and communication between ranks. The simulator uses the Simian conservative parallel discrete event simulation engine (Santhi, Eidenbenz, and Liu 2015).

Simulated traversals consist of simulated tree accesses and interactions. The time taken by a tree access or the computation of an interaction is set by parameters or can be represented as tasklists for which the hardware model returns a predicted runtime cost. We stop traversals after a certain number of simulated tree accesses represented by the chunk size parameter. This limit is present in the real application to allow each rank to check for communication at regular time intervals. If a traversal requires a tree cell owned by another rank, a request is added to a local list and the state of the traversal is recorded to be resumed at a later point. At the end of a chunk of traversals, the simulated rank sends its requests for cells from other simulated ranks and satisfies requests from other ranks received in the mean simulated time. We do not actually send data as we do not represent the tree but we simulate these communications and attempt to capture when such requests would be fulfilled in the real application. Each simulated rank is represented as an entity that performs three types of events:

- Progress on current traversals - the simulated rank performs available traversals up to $chunk_size$ tree accesses and then communicates with other simulated ranks;
- Receive requests - the simulated rank adds the request to a list so it can be answered at the next communication checkpoint;
- Receive cells - the simulated receives a requested cell from another rank and adds dependent traversals to a list of resumable traversals.

While this simulation still represents a lot of computations, it avoids representing and accessing the actual tree and doing physics computations, the two heaviest operations in the application in terms of runtime and memory usage.

4 VALIDATING SPHSIM

In order to validate our model, we compare its predicted results to the performance of the real code in a strong-scaling experiment. For this strong-scaling experiment, the problem size is fixed at around 100k particles and the number of ranks ranges from 1 to 1024. Both the simulation and the actual runs were computed on a cluster of 1500 nodes each equipped with a 12-core hyper-threaded Opteron 6176 @ 2.3GHz processor and connected with an Infiniband QDR interconnect. For this small problem size, the simulations only used a single core.

Figure 1 shows the results of this strong-scaling validation experiment. Both actual and simulated runtimes are given on the left chart. A – more traditional for strong-scaling – speedup representation of the same results is given on the right chart. For this problem size, scaling plateaus after 64 cores; this behavior is well captured by our simulation.

5 TESTING THE SCALABILITY OF SPHSIM

Unlike in section 4, we intend here to assess the scalability of our simulator and its underlying PDES engine. We first test our application simulator and our hardware model by running a weak-scaling experiment, in which the problem size, in terms of simulated ranks and number of particles, increases with the number of processors used. We use the same cluster as in section 4 for these experiments.

5.1 Weak-scaling experiment

The aim of this experiment is to assess the parallel capabilities of our simulation and the feasibility of simulating extreme-scale runs, such as those described in (Warren 2013). The problem size starts at around 2 billion particles and 48 simulated ranks on 24 physical ranks (a single node of the cluster). The problem size and number of physical ranks are then incrementally increased to reach 250 billion particles and 6144 simulated ranks on 3072 physical ranks (128 cluster nodes).

Figure 2 shows the results of the weak-scaling experiment in terms of runtime and maximum memory usage per node. Both curves diverge slowly from an ideal flat line but this experiment shows that the hero runs of 2HOT can be simulated using much less ranks, time and memory than running the actual code. Furthermore, changing hardware and software parameters of the simulation can be done in seconds, while doing so in the actual application requires careful tuning and extensive editing of the code.

5.2 Strong-scaling experiment

With this experiment, we assess the parallel capabilities of our simulator for average size runs. The instance size is set at 300 million particles and 3072 simulated ranks, representing an average of 100k particles per rank. This is typically the type of settings used for the real application. This simulation is run on a number of physical ranks ranging from 24 (128 simulated ranks per physical rank) to 3072 (1 simulated rank per physical rank).

Figure 3 shows the results of this strong-scaling experiment. Substantial speedups are obtained up to 192 ranks and diminishing returns up to 768. The use of additional physical ranks increases the runtime for this instance. This lack of scalability beyond 768 ranks can be explained by the nature of our PDES

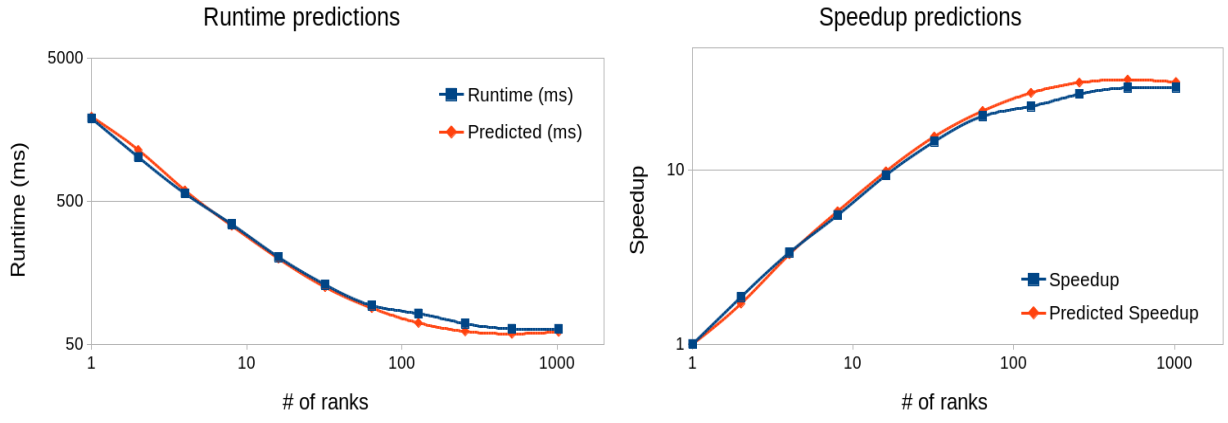

Figure 1: Strong-scaling validation result of our simulation.

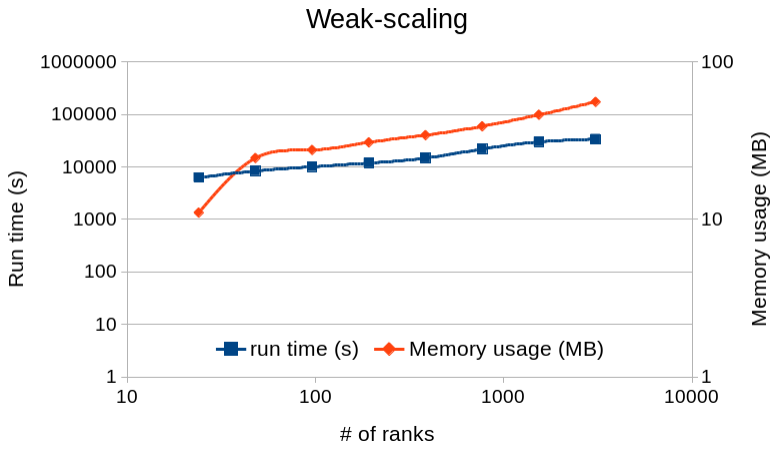

Figure 2: Weak-scaling results of our simulator. Runtimes (resp. maximum memory usage) are to be read on the left (resp. right) y-axis.

engine. In a conservative PDES engine, such as Simian, parallelism is drawn from computing multiple events occurring in the same time window or epoch. As the number of physical ranks increases, the number of entity – in our case simulated ranks – per physical rank, or LP, diminishes and so does the likelihood for an LP to have an event to handle at a given epoch, thus reducing parallel efficiency. Our weak-scaling experiment however showed that larger simulations can benefit from additional physical ranks.

6 RESULTS: USING SPHSIM

The point of simulating the behavior of an application such as SPH is to be able to explore parameter and hardware variations and determine optimal values at a lesser cost than running the actual code. By lesser cost, we mean either reduced runtime, reduced hardware needs (less cluster nodes, less memory etc) or reduced man-hours to edit the code. In order to test the ability of our simulator to perform such an exploration, we vary parameter and hardware values that are known to have an impact on the runtime of SPH. Namely, SPH's performance relies heavily on network communication and floating point computations on

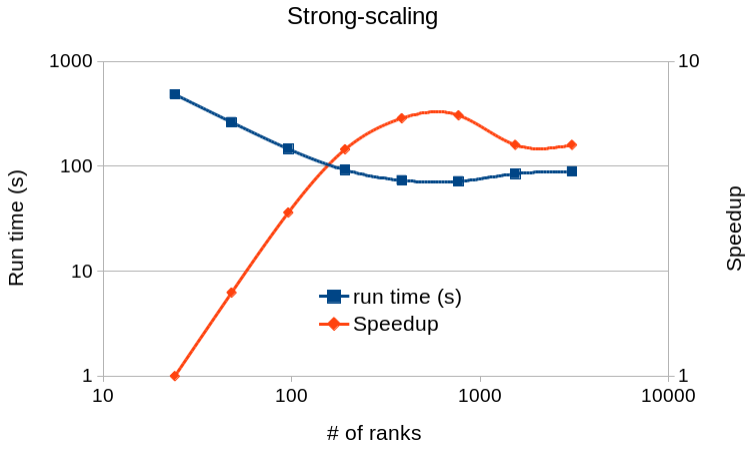

Figure 3: Strong-scaling results of our simulator. Run-times (resp. speedup) are to be read on the left (resp. right) y-axis.

the hardware side and on how often each rank performs communication with other ranks on the software side. In this next experiment, we vary the simulated values for these three characteristics and assess whether their simulated impact confirms what SPH users have reported or assumed. We model varying network performances by changing the simulated latency of communication and the floating point computation capabilities by changing the time taken to compute an interaction. We change the value of the chunk parameter to capture how often each simulated rank checks for communication.

In this experiment, we model a fixed instance of the problem consisting of 256 simulated ranks and 1k particles per rank. We first predict runtimes with values for the chunk parameter ranging from 2 to 100k; meaning that simulated ranks check for communication every 2 to 100k accesses to their local tree. For these simulations, the time to compute an interaction is set at 800 nanoseconds, while the network latency is set at .05 milliseconds.

Figure 4 shows the result of our experiment concerning the chunk size parameter. For this instance, the optimal chunk size is around 1000, meaning that ranks should check for communication every 1000 accesses to their local tree. Such a low value induces extra work but reduces the probability of ranks to be idle, waiting for data from other ranks to arrive. Depending on the nature of the problem, this parameter could even have a greater impact. For instance, in a problem where some ranks have initially little local work, such ranks would be idle for a substantial amount of time before they receive cells from other ranks with more local work.

We then predict runtimes with values for the interaction time parameter ranging from a very lightweight 50 nanoseconds to heavy computations taking 50 ms. For these runs, the chunk size is set at 1000 and the network latency at .05 milliseconds. Changing the time taken to compute an interaction in the real code is not something one can easily do. This time depends on the operations needed to be performed and thus on the type of physics being computed. These simulations aim at determining whether time should be invested in optimizing the interaction kernel or whether buying faster CPUs would be beneficial.

Figure 5 shows the results of our experiment concerning the interaction time. A slight performance gain can be obtained by reducing the interaction time from 800 nanoseconds to 400 nanoseconds. Reducing it further does not show any benefit and the whole computation becomes dominated by memory accesses to the tree. When the interaction time increases, however, we see a large impact on the overall performance. For physics with heavy interaction kernels, using dedicated hardware such as GPUs to compute interactions can therefore become beneficial.

Finally, we fix the chunk size at 1000 and the interaction time at 800 nanoseconds and vary the network latency. As SPH relies on frequent and fast communication between ranks, the assumption is that the latency

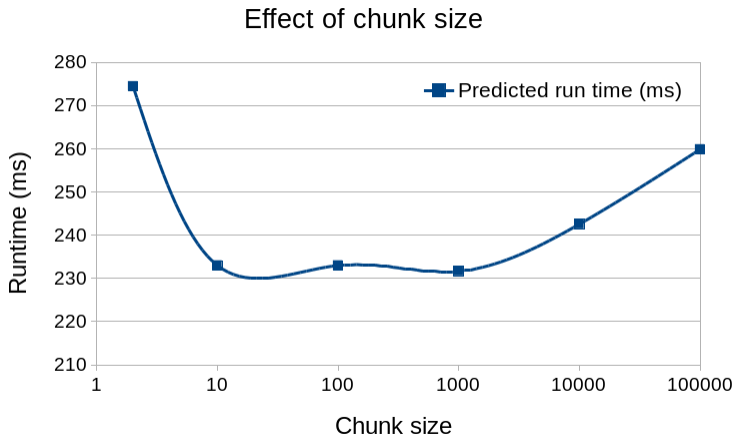

Figure 4: Effect of the chunk parameter on the predicted runtime of a single simulated step.

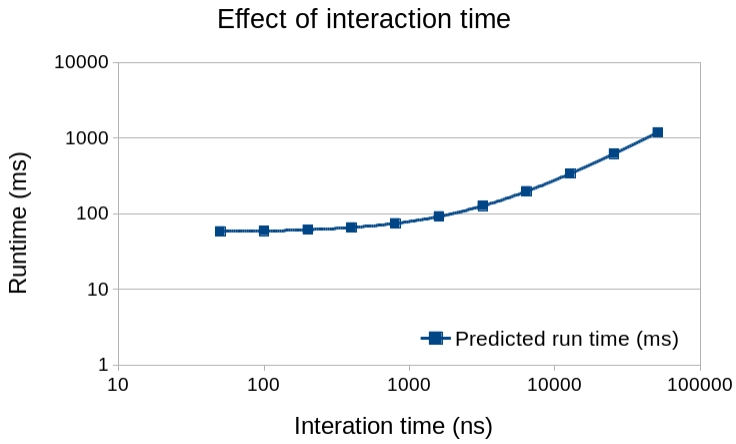

Figure 5: Effect of the interaction time on the predicted runtime of a single simulated step.

of network communications should have an impact on the overall performance. We use values for the latency ranging from 10 nanoseconds, more than an order of magnitude faster than existing infrastructures, to 100 milliseconds, the performance of a worldwide grid connected via the internet.

Figure 6 shows the results of our experiment on the network latency. Counter-intuitively, improving the network latency of a modern cluster has no impact on the performance of the application in the studied range. The results also show that a large scale run over a worldwide grid would incur a severe performance penalty. Nevertheless, decreasing the performance of a modern cluster by an order of magnitude does not bear a significant impact on the performance. These simulations tend to show that users should not invest on a faster network infrastructure, as the latency is well hidden by the application and negligible as long as ranks check for communication often enough.

7 CONCLUSION

We present performance prediction studies and trade-offs of Smoothed Particle Hydrodynamics (SPH) codes that rely on a Hashed OctTree data structure to efficiently respond to neighborhood queries. SPH codes are among the best scalable computational physics codes due to their ability to hide data movement

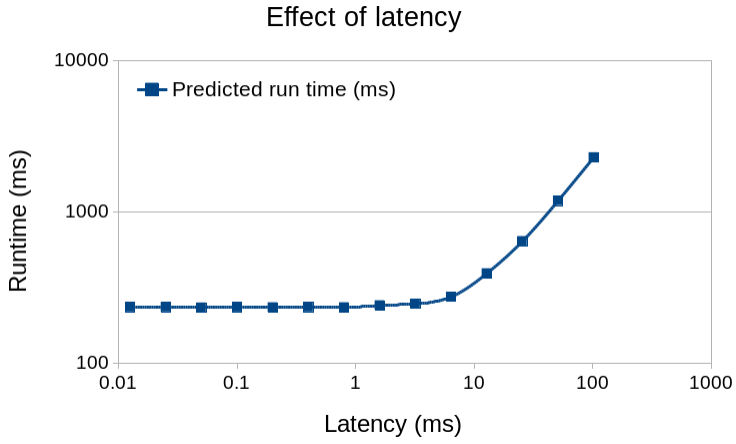

Figure 6: Effect of the network latency on the predicted runtime of a single simulated step.

latency. 2HOT (Warren 2013), the code we use as a model for this simulation, was recently scaled to run computations on a billion particles using 256k processors. We describe a statistical model of 2HOT, which allows us to avoid the time-consuming operations of creating and accessing a large data-structure and of computing physics kernels. By avoiding these time-consuming operations, we can simulate the behavior of the application faster and/or using less resources. Additionally, this simulation allows users to quickly test the performance of hardware or software variations without needing to get access to different hardware or edit the real application. This application simulator uses the PPT framework to model hardware aspects of the simulation. We perform a parameter scan of both hardware and software parameters to identify potential trade-offs in the real application. Our results show that despite increasing the amount of work, processors should communicate often to avoid being idle. We also show that despite the fact that the application relies heavily on communication between processors, improving the network infrastructure would not have a significant impact on the overall performance.

ACKNOWLEDGEMENTS

We would like to thank Andrew Nelson for his help with the manuscript and his valuable advice.

REFERENCES

Fryer, C. L., G. Rockefeller, and M. S. Warren. 2006. "SNSPH: a Parallel Three-Dimensional Smoothed Particle Radiation Hydrodynamics Code". *The Astrophysical Journal* 643 (1): 292.

Mniszewski, S. M., C. Junghans, A. F. Voter, D. Perez, and S. J. Eidenbenz. 2015. "TADSim: Discrete Event-Based Performance Prediction for Temperature-Accelerated Dynamics". *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 25 (3): 15.

Monaghan, J. J. 1992. "Smoothed Particle Hydrodynamics". *Annual Review of Astronomy and Astrophysics* 30:543–574.

Reed, D. A., and J. Dongarra. 2015, June. "Exascale Computing and Big Data". *Communications of the Association for Computing Machinery* 58 (7): 56–68.

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 Y. Levent, C. Wai Kin, Victor, M. Il-Chul, and R. Theresa, M. K., 3013–3024. Piscataway, New Jersey: IEEE Press: Institute of Electrical and Electronics Engineers, Inc.

Skillman, S. W., M. S. Warren, M. J. Turk, R. H. Wechsler, D. E. Holz, and P. Sutter. 2014. "Dark Sky Simulations: Early Data Release". *arXiv preprint arXiv:1407.2600*.

Warren, M. S. 2013. "2HOT: an Improved Parallel Hashed Oct-Tree n-Body Algorithm for Cosmological Simulation". In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 72. ACM.

Warren, M. S., and J. K. Salmon. 1995. "A Portable Parallel Particle Program". *Computer Physics Communications* 87 (1): 266–290.

Zamora, R. J., B. P. Uberuaga, D. Perez, and A. F. Voter. 2016. "The Modern Temperature-Accelerated Dynamics Approach". *Annual Review of Chemical and Biomolecular Engineering* 7 (1): 87–110.

AUTHOR BIOGRAPHIES

GUILLAUME CHAPUIS is a post-doctoral Research Associate with the Information Sciences Group (CCS3) at Los Alamos National Laboratory, New Mexico, United States of America. He holds a PhD degree in computer science from ENS Cachan (France) and a computer engineering degree from INSA Rennes (France). His research interests include Parallel Discrete Event Simulation, graph theory, High Performance Computing, General-Purpose Graphics Processing Units and bioinformatics. His email address is gchapuis@lanl.gov.

DAVID NICHOLAEFF is a computational physicist in the XCP division at Los Alamos National Laboratory. He studied mathematics and physics at UCLA for his undergraduate work; his graduate work at the University of Oxford explored quantum contextuality using tools from applied topology. David continues to research parallel and concurrent models and systems on both classical and quantum machines using projective geometry and algebraic topology.

STEPHAN EIDENBENZ is a computer scientist at Los Alamos National Laboratory. He leads research projects in cyber-security, computational co-design, performance prediction of super computers, and process modeling spanning a range of government and commercial sponsors as well as internal research grants. Stephan obtained his PhD from the Swiss Federal Institute of Technology, Zurich (ETHZ) in Computer Science. He has made research contributions in many areas of computer science, including cyber-security, computational co-design, communication networks, scalable modeling and simulation, and theoretical computer science. His email address is eidenben@lanl.gov.

ROBERT S. PAVEL is a Nicholas C. Metropolis Fellow Postdoc in the Applied Computer Science group at Los Alamos National Laboratory . His research interests are applying software engineering and techniques and tools derived from novel computer science research to improve scientific computing and applications. In particular he is focusing on ways to facilitate the use of asynchronous task based runtimes in production codes. His email address is rspavel@lanl.gov.