Performance modeling of a hierarchcial N-body algorithm for arbitrary particle distribution (Unrefereed Workshop Manuscript)

Keisuke Fukuda[1,a)] Naoya Maruyama[2,3] Jeremy S. Meredith[4] Jeffrey S. Vetter[4]
Satoshi Matsuoka[1]

Abstract: Hierarchical algorithms are considered to be important in next-generation large scale scientific computing. Such algorithms are typically compute-intensive and have higher communication locality that are beneficial on future supercomputers with much less B/F ratio. However, one of the big challenges of such algorithms is that the data structures and computation/communication patterns are irregular and it is difficult to analyze and predict the performance. In this paper, we introduce a performance modeling method for Fast Multipole Method, a typical example of hierarchical algorithms for N-body problems, using a domain specific performance modeling language Apsen. We show that our modeling scheme can adapt to various particle distributions parameters and provides useful information to application researchers to optimize algorithmic parameters.

1. Introduction

Many studies predict the next generation supercomputers will be more flops rich and bandwidth poor and the B/F ratio will be even smaller than those of the present supercomputers. In addition, the cost of data movement will continue increasing and global communications will be inefficient. One of the big issues of extremely large scale scientific applications is to reduce global communications and increase compute intensiveness.

The Fast Multipole Method (FMM) is an approximating algorithm for N-body applications and other scientific problems. It is a hierarchical algorithm, which is based on recursive decomposition and tree data structures. It is considered promising for extremely large-scale simulations thanks to its reduced global communications and higher compute intensiveness and locality. Yokota et al. have shown that FMM is more efficient than a conventional FFT-based spectral method with more than 4000 GPUs[1]. FMM was developed as an N-body algorithm, but now it is known to be useful for other applications including preconditioner for the boundary element method.

Despite the benefits of FMM, there are several big challenges in FMM. One of the challenges is difficulty of performance analysis and prediction caused by the irregularity of data structures and computations. FMM is based on recursive decomposition of the simulation space and uses an octree as a main data structure, where computation, memory access and communication patterns are more complex than many other algorithms that are based on matrices or regular grids. FMM is an adaptive algorithm, which means that the shape of the tree data structure highly depends on particle distribution of input data and not predictable before runtime. In addition, FMM can support any computation kernel if appropriate multiple and local expansions are given. They have different spatial and computational complexities[2].

Performance modeling is an effective approach to analyze and predict the performance of an application. Performance modeling efforts for FMM, however, has been limited. This is mainly because the irregularity and data dependency. Mathematical models cannot express the "shape" of a tree. Choi et al[3] have build a detailed analytical performance model which considers memory and cache hierarchy for KIFMM: one of the major implementations of FMM. Their models are limited to uniform particle distribution which generates a perfectly balanced tree and assumes deep understanding of the behavior of the application.

In this paper, we propose a performance modeling method using a domain specific modeling language Aspen. In Aspen, applications and platform hardware are modeled in a domain specific language (they are called "application modes" and "machine models"). Number of flops, memory reads and writes, communication, processor flops and bandwidth and many other characteristics are expressed. Aspen runtime combines an application model and a machine model to predict performance. Using Aspen delivers several benefits. First, the tree data structures and flow of computations can be fully expressed in Aspen grammar, which means any particle distribution can be supported. Second, the models are composable: since not only applications but ma-

[1] Tokyo Institute of Technology, Meguro-ku, Tokyo, 152–8550 Japan
[2] RIKEN Advanced Institute of Computational Sciences, Kobe-shi, Hyogo, 650–0047, Japan
[3] JST CREST
[4] Oak Ridge National Laboratory, 1 Bethel Valley Road, Oak Ridge, TN, U.S.A.
[a)] fukuda@matsulab.is.titech.ac.jp

chines are modeled, performance models can be combined and applied to a new hardware. Many supercomputers nowadays are accelerated with NVIDIA GPUs or Intel MIC processors and new generations of the accelerators or even a new device might be supported. There are also a wide variety of interconnects. Many GPU-accelerated supercomputers have fat-tree infiniband network, while IBM Blue Gene series and the K computer have their own interconnects. If the application and hardware performance models are integrated, it is not realistic to support all platforms or rebuild the model for new hardware.

2. N-body algorithms and the Fast Multipole Method

2.1 Family of N-body algorihtms

N-body problems appear in many scientific applications including astrophysics, molecular dynamics, acoustic analysis, and electromagnetics. The computation complexity of the native method is $O(N^2)$ and several approaches have been proposed to reduce the complexity. The most accepted algorithms are Barnes-Hut tree algorithm[4] and the Fast Multipole Method(FMM)[5]. They are approximate algorithms and reduce the computational complexity to $O(NlogN)$ and $O(N)$ respectively. We focus on FMM in this paper.

2.2 Fast Multipole Method

FMM was first proposed by Greengard[5]. In this paper, the algorithm is described only as long as necessary and details and mathematical aspects of the algorithm are not given. For more details, Greengard[5] and Yokota wrote a very good survey[2].

FMM achieves $O(N)$ complexity by aggregating far particle information becase the force from other particles decays rapidly, while force from near particles is still calculated directly in the same manner of the naive N-body computation. The aggregated data of particles is called "multipole" and "local" from the mathematical techniques.

Within a single timestep, the computational domain is split recursively into an octree of which each leaf cell has at most N_{crit} particles. N_{crit} is a user-defined parameter that determines balance of direct computation and approximate computation. The generated octree is higher and there is more approximate computations and less direct computations. N_{crit} is typically between 16 and 256 for CPUs and more than 1000 for GPUs and massively parallel many core processors. After building an octree, actual force calculation is done in an evaluation step. There are six phases in the evaluation step: P2P, P2M, M2M, M2L, L2L. P2P phase evaluates the forces between near particles and the rest of the phases are for approximate computation. These six phases are common between FMM implementations, although there is great difference in their execution order and control flows.

In P2P phase, direct N-body calculation is done with a limited set of leaf cells. In a typical implementation and configuration, P2P happens only between a pair of neighboring cells. Each leaf cell has at most N_{crit} particles and number of P2P interactions is proportional to the number of leaf cells: $O(N/N_{crit})$. Thus the complexity of total amount of P2P computations is $O(NN_{crit})$.

Approximate computation begins with P2M(Particle to Multipole) phase. First, particles in a leaf cell are aggregated into an vector called "multipole" using a mathematical technique called "multipole expansion." Next, M2M (Multipole to Multipole) phase walk the tree upward from the leaves to the root to aggregate children's multipole vectors into the parent's single multipole vector recursively. M2M is followed by M2L (Multipole to Local) phase that computes interactions between "far" nodes and converts the multipole vector of a source node to a vector called "local" of the target node. This means information from far particles is transmitted to receiver particles. A local vector of a node is transferred to the child nodes using "local" expansion. This phase is called L2L (Local to Local). After L2L interactions reach the leaf nodes and local vectors are ready, the particles receive information in L2P (Local to Particle) phase.

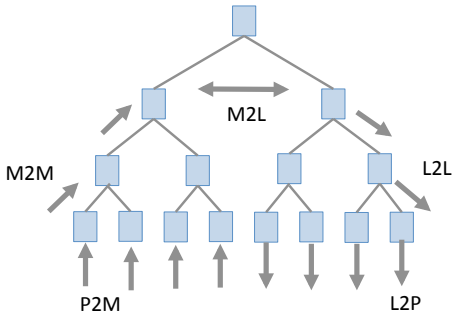

Fig. 1 Picture of P2M, M2M, M2L, L2L, L2P phases

Another parameter P is used to specify accuracy of evaluation. Multipole expansion converts the force kernel into a series. Approximation is done by ignoring the terms after the P-th term. P is typically between 4 and 12 depending on applications.

2.3 ExaFMM

ExaFMM is an implementation of FMM developed by Yokota et al.[6] It is one of fastest implementations as of 2013[2]. It is a collection of C++ header files and designed to be an integration framework for many N-body applications.

In this paper we focus on a single-node, multithreaded version of ExaFMM. ExaFMM is parallelized using MassiveThreads[7] by Taura[8].

2.3.1 Tree construction

ExaFMM first constructs an octree using recursive decomposition of the space. As described above, the space is decomposed into 8 children recursively until all of the leaf nodes have at most N_{crit} particles. The tree can be adaptive, which means that levels of a leaf cells differ, depending how many particles are within a certain area. After the decomposition, the octree is converted into a linked list of nodes.

2.3.2 Evaluation

In ExaFMM, there are three major steps in the force evaluation phase: upwardPass, dualTreeTraversal, and downwardPass. It is a unique feature of ExaFMM that the three steps are implemented in a recursive and task parallel way. They start from the root cell and traverse the tree spawning tasks if necessary. The efficient implementation of underlying MassiveThreads runtime enables tens of thousands of tasks run efficiently on a multicore

CPU. upwardPass calculates P2M kernel from particles to multipole within each leaf cell and M2M kernel from children's multipole to parent's multipole. dualTreeTraversal computes M2L kernel or P2P kernel on every pair of tree nodes, depending on if they are "near" or "far". Finally, downwardPass is an inversed process of upwardPass, where L2L and L2P kernels are calculated. Below are shown the pseudo code of the three steps (partially cited from Taura[8]).

```
// upwardPass
def upwardPass(cell) {
    for ch in children of cell {
        // spawn child tasks if necessary
        upwardPass(ch)
    }
    if(cell is leaf?) {
        P2M(cell)
    } else {
        M2M(cell)
    }
}

// dualTreeTraversal
def daulTreeTraversal (C1, C2) {
    if (C1 and C2 are far enough) {
        M2L(C1, C2)
    }
    else if (C1 is leaf and C2 is leaf) {
        P2P(C1, C2)
    }
    else if (C1 is leaf) {
        for cc in C2's children {
            traverse(C1, cc)
        }
    }
    else if (C2 is leaf) {
        for cc in C1's children {
            traverse(cc, C2)
        }
    }
    else if (C1 == C2) {
        for a,b in C1' children where a < b {
            traverse(a,b)
        }
    }
    else if (radius(C1) > radius(C2)) {
        // If C1 is a larger node
        for c in C1's children {
            traverse(c, C2)
        }
    }
    else {
        // If C2 is a large node
        for c in C2's children {
            traverse(C1, c)
        }
    }
}

// downwardPass
def downwardPass(cell) {
    // calculate L2L from the parent to cell
    L2L(cells)
    if(cell is leaf?) {
        L2P(cell)
    } else {
        for ch in cells' children {
            downwardPass(ch)
        }
    }
}
```

3. Aspen

In this section, we give a brief overview of a performance modeling domain specific language Apsen. We don't describe the grammar or detailed information how Aspen works here. For more details, refer Spafford et al.[9].

In Aspen language, calculation, memory access and communication of an application are explicitly written. It also has hardware specifications, called machine models, including floating point calculation units, accelerators, memory devices, and network devices and topologies. The Aspen runtime combines the application model and machine model to estimate runtime, energy consumption, scalability and other aspects of application execution.

Aspen was originally proposed as a high-level modeling system to enhance co-design and cooperation between application researcher and computer scientists and enables efficient future supercomputer designs, but it is also useful for our purposes of modeling irregular applications. Modeling such applications with conventional mathematical models is highly challenging. Aspen allows application models to be written using data structures and control flows, which are enough expressive to model tree data structures and traversal operations. Another benefits of adopting Aspen is that it provides modular and composable components of application and machine models so that the force kernels of an application and ExaFMM framework can be separated in a sophisticated way.

4. Modeling

We build a performance model of ExaFMM in two steps: modeling of the 6 find-grained kernels (P2P, P2M, M2M, M2L, L2L, L2P) and combining them in the control flow of upwardPass, DualTreeTraversal, and downwardPass phases.

4.1 Modeling 6 Kernels

In this section, we build models of 6 fine-grained kernels (P2P, P2M, M2M, M2L, L2L, L2P). As described above, ExaFMM is designed as a framework to support any user-provided kernels if sufficient multipole and local expansions are given. Thus these kernel models can be built by users and integrated into the control flow model. The models are self-contained: flops, memory accesses based on necessary algorithmic parameters such as P are sufficient and they don't need the information of ExaFMM.

In this paper, we focus on the Cartesian expansion kernels, which are embedded in ExaFMM distribution and are the simplest.

The actual models of the kernels are fairly straightforward. Below is an example of P2P kernels. Constant numbers embedded in the model is calculated by the model generation routine from the algorithmic parameters. Although such constants can be parameterized in the model and given in the evaluation time, we decided to embed the numbers directly in the models due to minor technical issues of Aspen runtime.

```
kernel P2P_100_125_True {
    // mutual = True, Ni=100, Nj=125, nn=12500, n2=225
    execute {
      loads [225 * 4 * wordSize] from srcBodies
      loads [225 * 4 * wordSize] from trgBodies

      flops [12500 * 3*3] as simd

      flops[12500 * 6 + 3] as simd
      flops [12500 * 1] as simd

      flops [12500 * 1] as sqrt, simd
      flops [12500 * 2] as simd


      flops [12500 * 6] as simd
      flops[12500 * 1 * 4] as simd
      stores [125 * wordSize * 4 * 1] to srcBodies
    }

    execute {
      flops [100 * 4] as simd
      stores [100 * wordSize * 4] to trgBodies
    }
  }
```

4.2 Modeling tree structure and control flows

After the models of the six kernels are ready, we can generate a whole application model of ExaFMM. In a normal situation it is necessary to port all of the tree-construction code of ExaFMM to a model generation program to model ExaFMM's behavior without actually running ExaFMM. However, it's a technical issue and the range of this paper is to evaluate how the models can estimate the behavior of ExaFMM. Thus we re-use the ExaFMM code and generate Aspen code after an octree is generated. As described above, a simple linked list of octree nodes is obtained from the tree construction phase and it is feed to model generation script. Below is a small part of the generated Aspen code.

```
model exaFMM {
  param P = 4
  param wordSize = 4
  param NCHILD = 8
  param NTERM = P*(P+1)*(P+2)/6

  data CX [3*wordSize]
  data BX [3*wordSize]
  data Xi [3 * wordSize] // Target node's X
  data Xj [3 * NCHILD * wordSize] // Parent node's X
  data Li [NTERM * wordSize] // for L
  data Lj [NTERM * wordSize] // for L
  data Mi [1 * wordSize]     // for M
  data Mj [1 * wordSize]     // for M
  data B_TRG [NBODY * 4 * wordSize] // for L2P

  kernel main {
    call upwardPass
    call dualTreeTraversal
    call downwardPass
  }
  kernel upwardPass {
    call P2M_125
    // ...
    // ...
    call P2M_125
    call M2M
  }

  kernel dualTreeTraversal {
    call P2P_125_125_True
    call P2P_125_125_True
    // ...
    // ...
  }

  kernel downwardPass {
  call L2L
  call L2P_125
  // ...
  // ...
  call L2L
  call L2P_125
}
```

4.3 A comparison target as a simple model

We setup another model as a comparison target to show how our Aspen model is useful. It is based on the computational complexity of each phase and performance fitting.

ExaFMM executes P2M and M2M together as upwardPass, M2L and P2P as dualTreeTraversal, and L2L and L2P as downwardPass. Thus we build and tune for each of the three passes and sum up them to estimate the overall evaluation time. **Table 4.3** shows the computational complexity of each of the six phases.

Thus we define model as:

$$T = aNP^2 + b\frac{N}{N_{crit}}P^4 + c\frac{N^2P^6}{N_{crit}} + dNN_{crit} + e\frac{N}{N_{crit}}P^4 + fNP^2$$

where T is the evaluation time to estimate and a, b, c, d, e, f are scalar parameters. We adjust the parameter with a relatively large case $N = 1000000, N_{crit} = 128$ with lattice distribution. Apparently, this model does not consider particle distribution, since such a mathematical model cannot take a shape of a tree into consideration.

Table 1 Computational Complexities of ExaFMM phases

Phase	Complexity	Note
P2M	NP^3	Num of leaves is $O(N/N_{crit})$, each P2M is $O(N_{crit}P^3)$
M2M	$\frac{NP^4}{N_{crit}}$	Num of M2M calls is $O(N/N_{crit})$ Each M2M is $O(P^4)$
P2P	NN_{crit}	Num of leaves is $O(N/N_{crit})$ and each P2P call is $O(N_{crit}^2)$
M2L	$\frac{N^2P^6}{N_{crit}}$	Num of M2L calls is $O(N^2/{N_{crit}}^2)$ each M2L call is $O(P^6)$ (given by Yokota et al.[2].)
L2L	$\frac{NP^4}{N_{crit}}$	Same as M2M
L2P	NP^3	Same as P2M

5. Analysis and Evaluation

5.1 Evaluation environment

All the evaluations are done on TSUBAME2.5 installed in Tokyo Institute of Technology. We use a single "thin" nodes which have two sockets of 6-core, 2.96GHz Intel Xeon x5670 and 54GB of DDR3 main memory.

The performance models written in Aspen are also evaluated on using the TSUBAME2.5 machine model. The interpretation of the models is done on "fat" nodes with 96GB main memory.

5.2 Analysis of kernels

Before evaluating the overall performance of the models, we evaluate each of the 6 kernel models (P2M, M2M, M2L, P2P, L2L, L2P) individually. Since each kernel call is very low-level and find-grained to measure with normal gettimeofday or similar system calls, we adopt RDTSC (Read timestamp counter) operation of Intel Processors. Note that this analysis is not about the total amount of P2P/M2L/L2L time, but each function call of the phases.

Here we show three figures: **Fig. 2**, **Fig. 3** and **Fig. 4**. P2P and M2L are the most significant phase in ExaFMM and we choose one from the rest of the phases.

In Fig. 2, the red dots show measured samples of P2P kernels. Ni and Nj are numbers of the two leaf cells in each interaction. The red solid line is estimated performance by Aspen. Note that the X-axis is $Ni \times Nj$. The dotted green line shows $O(Ni \times Nj)$ and $O(Ni + Nj)$ for reference. The actual dots implies that with small Ni and Nj the number of flops is small and the memory access, which is $O(Ni + Nj)$ because just particle positions potentials are read, and asymptotically becomes $O(Ni \times Nj)$. With enoughly large $Ni \times Nj$, where approximately $Ni, Nj \geq 20$, the estimated time is about $1/3$ of the actual samples.

Fig. 3 is M2L. Since the M2L kernel, not the total time of all M2L kernels, only depends on the accuracy parameter P, 1% trimmed mean of 1000 times of M2L function call. The red line is actual samples and the yellow is estimated time by Aspen. The actual samples show outlier at P=7, but the two lines have same complexity. L2L modeling in Fig. 4 is from the rest of the phases, which are less significant in ExaFMM. Similar to M2L, L2L phase only depends on the parameter P, but we show all the actual samples, not mean values, because the variance is relatively large. However, the estimated values are more accurate than P2P and M2L models.

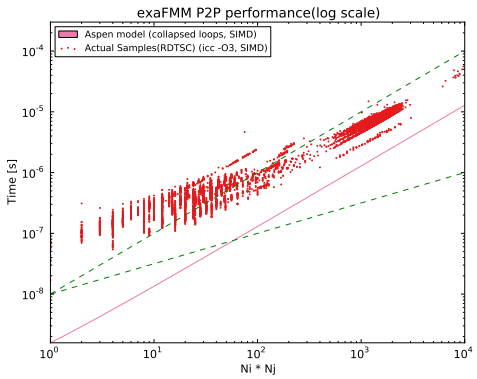

Fig. 2 Actual and Estimated Performance of P2P kernel

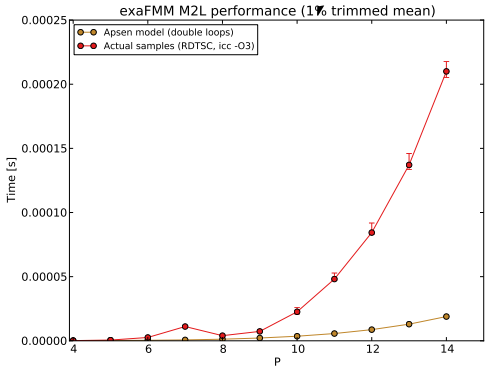

Fig. 3 Actual and Estimated Performance of M2L kernel

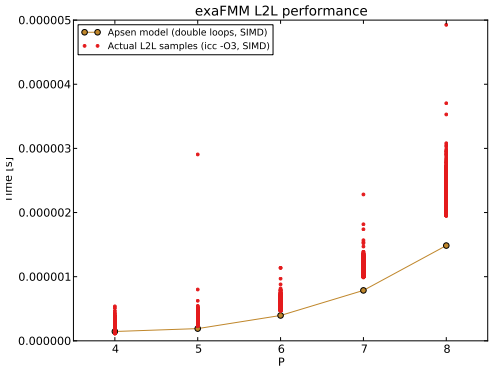

Fig. 4 Actual and Estimated Performance of L2L kernel

5.3 Overall Evaluation

Fig. 5 and **Fig. 6** show runtime of the evaluation phase of actual ExaFMM runs and estimation by the Aspen models with $P = 4, 6$ and 50000 particles. Particle distributions are lattice, plummer and sphere, and N_{crit} values are 16, 32, 64, 128, 196, 256 and 512. The estimated times by Aspen (blue bars) are normalized using the case of the leftmost lattice-16 case.

In Fig. 5, and Fig. 6, the proposed Aspen model gives more useful information about the application behavior. An important role of performance modeling is to help application researchers/users to determine the best algorithmic parameters to minimize the computation time. In this case, the most important parameter is N_{crit}, because it greatly affects the overall computation time. There's always a tradeoff of direct and approximate computation time and it is not obvious which N_{crit} values is optimal for a given condition. The Aspen model indicates 64 for all of the three distributions in this case. From the actual results, the values are the second-best for lattice and sphere and the best for plummer distribution out of 7 choices for each. On the other hand, the simple model implies 16 for all of the case, but the value is the worst for lattice and plummer, and 5th for the sphere distribution.

The error values of Aspen models are at most 3× approximately, which are corresponding to the results of per-kernel analysis shown in the section 5.2. **Table 5.3** and **Table 5.3** We observe a similar result in Fig. 6,

Table 2 Breakdown of Plummer-512 case

Step	Actual Time	Estimated Time
upwardPass	0.00380	0.00093
dualTreeTraversal	1.39707	0.39563
downwardPass	0.00334	0.00096

Table 3 Breakdown of Lattice-128 case

Step	Actual Time	Estimated Time
upwardPass	0.00355	0.00087
dualTreeTraversal	0.53717	0.15575
downwardPass	0.00311	0.00089

6. Related Work

Choi[3] and Chandrawmowlishwaran[10] have proposed a very accurate performance model of KIFMM: another efficient implementation of FMM. They consider memory hierarchy including L2 and L3 cache, but their model assumes uniform particle distribution and detailed knowledge about the implementation. Although our model also requires some knowledge about the ExaFMM implementation, the necessary information is only the abstract structure of the recursive tree traversal and not detailed behavior.

Teng et al. gave a theoretical analysis on load balancing between distributed computing nodes in a large scale simulation[11]. Althouhg the reuslt is insightful, it is not practical for real world FMM implementations.

7. Future work

Our current modeling scheme has two major limitations. First,the current model assumes single-node execution and does not consider inter-node MPI communication. Second, in current Aspen implementation the model calculation time is long and the Aspen runtime consumes large memory. The results shown in this paper is limited to 50000 particles, which is much smaller than actual simulation.

8. Conclusion

In this paper, we proposed a performance modeling method of the Fast Multipole Method for any particle distribution using a domain specific modeling language Aspen. Our model assumes less information about the implementation and gives accurate result for various particle distribution and algorithmic parameters and provides useful information to tune the algorithmic parameter for the application researchers.

References

[1] R. Yokota, L. Barba, T. Narumi, and K. Yasuoka, "Petascale turbulence simulation using a highly parallel fast multipole method on GPUs," *Computer Physics Communications*, Sep. 2012. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0010465512002974

[2] R. Yokota, "An FMM Based on Dual Tree Traversal for Many-core Architectures," Sep. 2012. [Online]. Available: http://arxiv.org/abs/1209.3516

[3] J. Choi, A. Chandramowlishwaran, K. Madduri, and R. Vuduc, "A cpu: Gpu hybrid implementation and model-driven scheduling of the fast multipole method," in *Proceedings of Workshop on General Purpose Processing Using GPUs*, ser. GPGPU-7. New York, NY, USA: ACM, 2014, pp. 64:64–64:71. [Online]. Available: http://doi.acm.org/10.1145/2576779.2576787

[4] J. E. Barnes, "A modified tree code: Don't laugh; It runs," *Journal of Computational Physics*, vol. 87, no. 1, pp. 161–170, Mar. 1990. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/002199919090232P

[5] L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations," *J. Comput. Phys.*, vol. 73, no. 2, pp. 325–348, 1987. [Online]. Available: http://portal.acm.org/citation.cfm?id=36895.36901 http://www.sciencedirect.com/science/article/B6WHY-4DD1T30-K7/2/2b3def8a3a8d71ff0d1697298ea6d2c8

[6] R. Yokota and L. a. Barba, "A tuned and scalable fast multipole method as a preeminent algorithm for exascale systems," *International Journal of High Performance Computing Applications*, vol. 26, no. 4, pp. 337–346, Jan. 2012. [Online]. Available: http://hpc.sagepub.com/cgi/doi/10.1177/1094342011429952

[7] J. Nakashima, S. Nakatani, and K. Taura, "Design and implementation of a customizable work stealing scheduler," in *Proceedings.3rd International Workshop on Runtime and Operating Systems for Supercomputers*.

[8] K. Taura, J. Nakashima, R. Yokota, and N. Maruyama, "A Task Parallelism Meets Fast Multipole Methods," in *Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA)*, no. 1, 2012.

[9] K. L. Spafford and J. S. Vetter, "Aspen: A domain specific language for performance modeling," in *2012 International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, Nov. 2012, pp. 1–11. [Online]. Available: http://dl.acm.org/citation.cfm?id=2388996.2389110 http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6468530

[10] A. Chandramowlishwaran, J. W. Choi, K. Madduri, and R. Vuduc, "Brief Announcement: Towards a communication optimal fast multipole method and its implications for exascale," in *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, no. 2, Pittsburgh, PA, USA, 2012.

[11] S.-H. Teng, "Provably Good Partitioning and Load Balancing Algorithms for Parallel Adaptive N-Body Simulation," *SIAM Journal on Scientific Computing*, vol. 19, no. 2, pp. 635–656, Mar. 1998. [Online]. Available: http://epubs.siam.org/doi/abs/10.1137/S1064827595288942

http://portal.acm.org/citation.cfm?id=289842

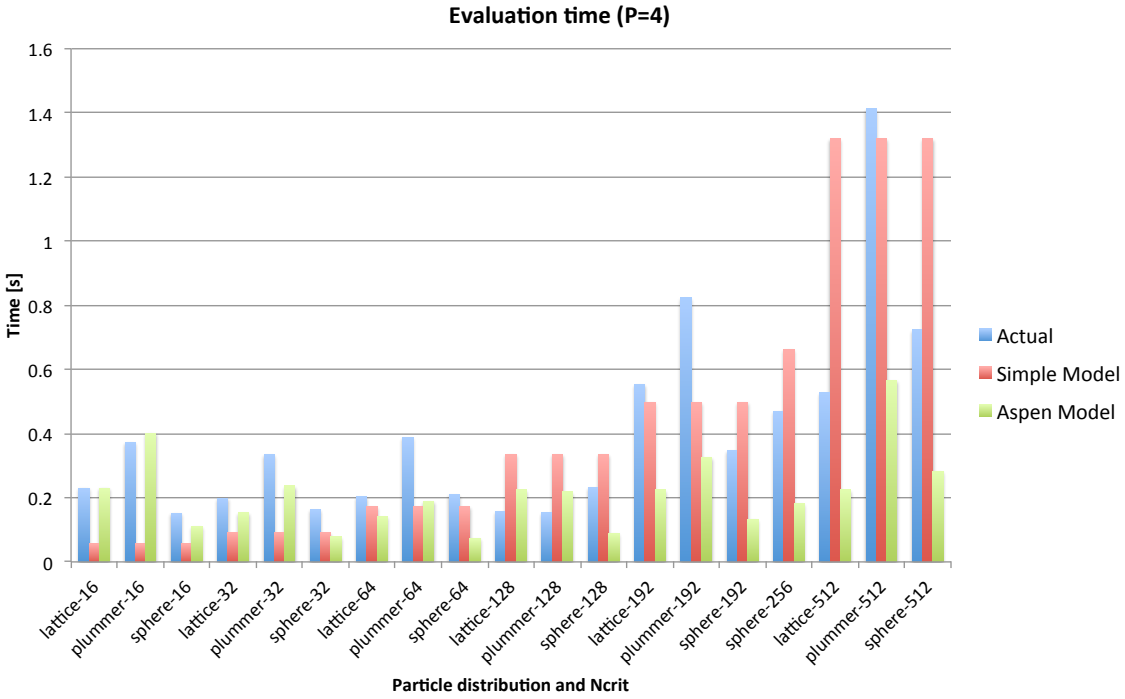

Fig. 5 Aspen Model and Actual Performance with P=4

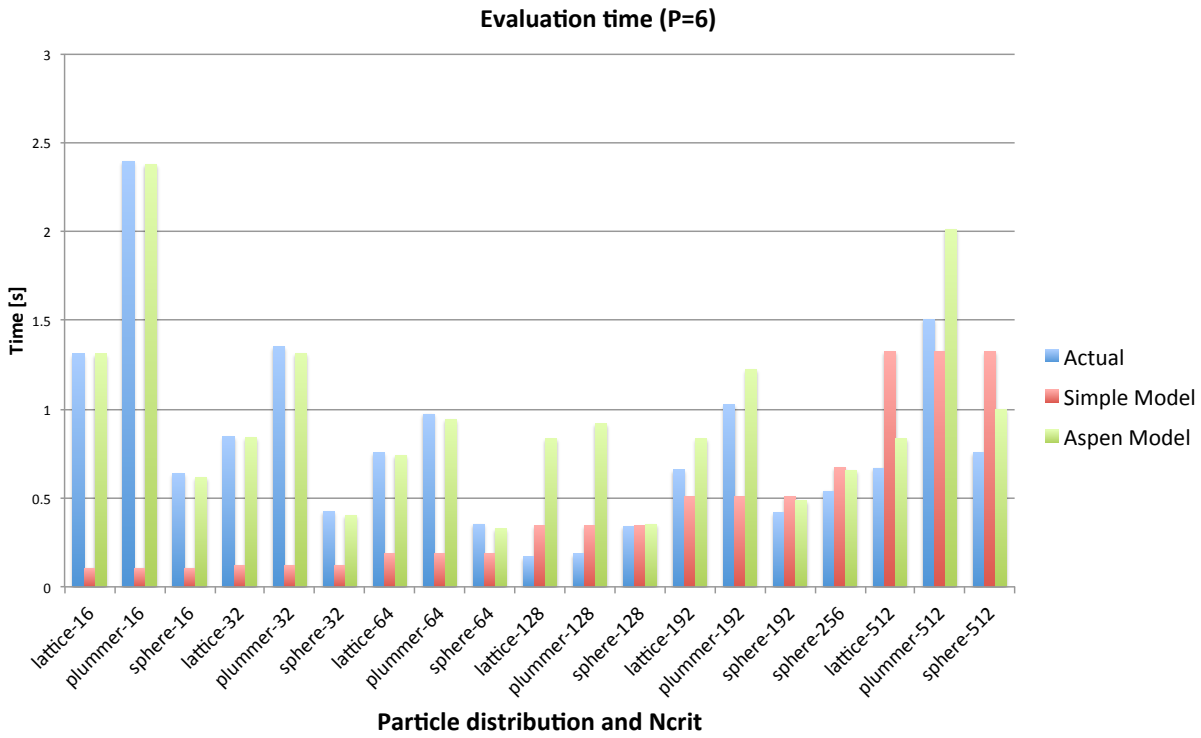

Fig. 6 Aspen Model and Actual Performance with P=6