

# Performance and Communication Modeling for Exascale Proxy Architecture in Aspen

Mariam Umar  
Virginia Tech  
Blacksburg, VA  
mariam.umar@vt.edu

Jeffrey S. Vetter  
Oak Ridge National Laboratory  
Oak Ridge, TN  
vetter@ornl.gov

Kirk W. Cameron  
Virginia Tech  
Blacksburg, VA  
cameron@vt.edu@vt.edu

## ABSTRACT

The performance capability of current high-performance computing cores are already at their threshold, leading to an unprecedented increase in number and complexity of exascale computational processors. Anticipating the performance and communication modeling required for millions of exascale cores necessitates profiling and modeling of proxy exascale applications on current supercomputers to help us pin-point to the application's hardware requirements to perform corresponding what-if analysis. This has to be done while considering both applications and hardware, leading to efficient co-design for exascale proxy architectures. In this paper, we study and profile the computation and communication of one of the exascale proxy applications – Co-design for Molecular Dynamics (CoMD). We model one of the proposed exascale homogeneous architecture in the Aspen machine modeling language and develop the model into a refined proxy architecture. We model the CoMD's computational and communication requirements on Exascale proxy architecture in the Aspen application modeling language to study the scalability analysis. We also present an analysis about communication profiling for CoMD in Aspen, which guides the efficient communication mapping of the application on the proxy architecture.

## KEYWORDS

XXXXX,XXXXX,XXXXX

### ACM Reference Format:

Mariam Umar, Jeffrey S. Vetter, and Kirk W. Cameron. 2018. Performance and Communication Modeling for Exascale Proxy Architecture in Aspen. In *Proceedings of Workshop on Open Source Supercomputing (OpenSuCo'18)*. ACM, New York, NY, USA, 9 pages. [https://doi.org/XX.XXX/XXX\\_X](https://doi.org/XX.XXX/XXX_X)

## 1 INTRODUCTION

Current high-performance computing systems are at the peak of their computational performance and therefore the research community is driving efforts towards deployment of exascale systems. Application and hardware design has to go hand-in-hand for the evolution of any new system. The complexity of studying and

analyzing applications and hardware of exascale is much more challenging as the hardware does not exist today. DOE has proposed a set of four theoretical abstract architectures with recommended system configurations. The development of an application suite has progressed to provide a suite of 13 proxy applications that are representative for exascale systems. Any research involving the exploration of exascale systems has to first devise a technique to transform the theoretical abstract architecture to proxy architecture and validate it to see whether it provides the same peak theoretical performance limits as expected by DOE. Later, this proxy architecture can be used to explore the patterns of execution of proxy applications and to study various performance, scalability and communication aspects of the system. Moreover, the predictive modeling for communication and computation at the scale of exaFlops requires complex processing on application type, its input size, hardware configuration, and systems conditions.

We deal with two issues in this paper, the realization of exascale architecture in terms of proxy machine model deployment and testing it for communicational and computational constraints. There has been some research done to propose and test various prototype exascale hardware, that represents the scalability of proposed exascale machines [? ? ], however, the resources required to generate prototype hardware are limited. Simulators, have a long prevalence to model various kinds and scales of architecture [? ? ]. ExaSAT [? ] is a scalable simulator that is developed specifically for varying and testing the scale of exascale architectures. Emulators have also been used to test and reconfigure various architectural features [? ]. Machine learning and statistical modeling techniques have been used to study the effect of application performance on current Giga and Peta scale performing systems, and those results are used to propose and extrapolate towards exascale architectures. Though all of these techniques have worked for the specific research area, these are not applicable to a wider problem area. None of the modeling and performance prediction techniques reflect the abstract architectures proposed by DOE [? ]. Even if we were to ignore the expected scale of future architectures, communication parameters and execution time prediction is itself a challenging problem, which takes into account the number of nodes, processes per node, data distribution, data placement and so forth. A number of communication prediction models have been proposed to optimize applications in terms of reducing message size [? ], reducing communication overhead [? ], hybrid MPI/OpenMP modeling [? ], and automatic mapping to heterogeneous resources [? ].

Our contributions in this domain are two-fold: We have implemented a homogeneous abstract machine model and present it as parameterized proxy architectures. We use Aspen DSL – a fast, accurate, portable and composable domain specific language to develop

---

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

OpenSuCo'18, November 2018, Dallas, Texas

© 2018 Association for Computing Machinery.

ACM ISBN XXX-XXXX-XX-XXXX/XX/XX...\$15.00

[https://doi.org/XX.XXX/XXX\\_X](https://doi.org/XX.XXX/XXX_X)

the proxy homogeneous architecture design. We test the theoretical peak performance of the proxy homogeneous architecture in Aspen versus that is predicted by the exascale computing project. We test the Co-design of Molecular Dynamics (CoMD) Proxy application on the proxy abstract architecture in Aspen. As the main experimental consideration on the extent of exascale architecture is the span of computational nodes, it is expected that communication design and mapping will be a critical modeling issue.

We develop and test a communication mapping framework in Aspen that uses communication and computation modeling of the application in order to map the application on the hardware. Moreover, it shows near-optimal communication mapping parameters e.g., optimal number of ranks, memory type to use in a multiple type of memory hierarchy, etc.

Our contributions in this direction are:

- (1) Implementation of a homogeneous, exascale abstract machine model [?] using Aspen and validation of the model for theoretical peak performance
- (2) Validation of the homogeneous exascale architecture using the CoMD proxy application
- (3) Design and implementation of a communication mapping framework in Aspen that maps the profiled application requirements with hardware requirements and provides a summary of the near-optimal communication mapping choices at runtime. We develop the framework and validate the results using the supercomputing facilities of Cori at NERSC.

## 2 DESIGN

In this section, we explain the theoretical constraints and requirements of the homogeneous exascale abstract architecture and how we implemented and validated them in Aspen. Later, we propose our analytical communication and computation model that maps the proxy application to the candidate proxy architecture in Aspen.

### 2.1 Abstract homogeneous exascale architecture

In the top ten exascale challenges [?], the implementation of an abstract machine model and an abstract application model are defined as two important aspects of exploration of co-design required by exascale systems. Ang et al. [?] define four different abstract machine models, (homogeneous, non-integrated heterogeneous, integrated heterogeneous and tightly integrated heterogeneous). In each of the abstract models, nodes can comprise of multiple units of multi-cores, along with the accelerator (in case of heterogeneous architectures). The individual processors of the exascale architecture are currently unknown, and therefore the current processor technology is expected to be used as the basic building blocks of the system. All the abstract models define an on-chip and off-chip memory and an on-chip network required to process the communication between multiple processing units.

While this paper deals with the implementation and testing of the homogeneous exascale architecture, and it is anticipated that accelerators will play a major role in the architecture development of the exascale machines. The aim of this paper is to understand the importance of abstract representation of the exascale architectures,

and we use the example of homogeneous architecture to test the accuracy of the abstract machine model representation in Aspen.

Figure 1 shows the block diagram of a homogeneous CPU based architecture (source: [?]). In this paper, we implement Intel's Haswell processor as the basic processor technology in the architecture since it is proposed that existing compute units will remain the basic building blocks of the exascale system. A conceptual view of Intel's Haswell processor is shown in Figure 2.

Table 1 lists the configurations of the abstract homogeneous architecture implemented in this paper, as proposed by Ang et al. [?]. Each processor is a 64-128 bit, dual-core threaded, 8-16 issue and out of order execution micro-processor. We use three levels of cache hierarchies for each microprocessor, where L1 cache is in the range of 8KB – 128KB, L2 cache ranges between 256KB – 2MB, and L3 cache contains 64MB – 128MB. As mentioned in Figure 1, two types of memories are mentioned in the system, where the in-package memory is expected to be HBM (high-bandwidth) memory [?], and is marked having a capacity of 64-192 GB. The off-chip memory is proposed to be using DRAM technology and provides a much larger capacity i.e., 512GB-2TB. An off-chip NVRAM (2TB-16TB) is also expected to be included to provide asymmetric access latency. A summary of this information is provided in Table 1 [?]. Figure 3 shows the high-level representation of individual nodes in an overall view of system.

The interconnect technology to be used for exascale system is of high-importance. The latency incurred by the interconnect may cause a serious performance issue. A number of high bandwidth and low latency interconnects are considered for exascale architecture by Ang et al. [?]. In this paper, we implement and explore the Dragon Fly interconnect for homogeneous exascale architectures that provides the benefit of reduced optical interconnect and enables high-bisection bandwidth.

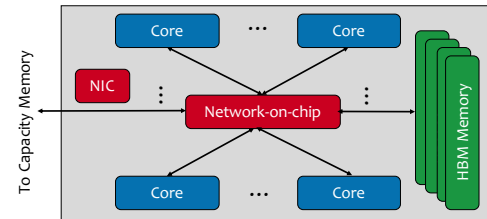


**Figure 1: Homogeneous Abstract Machine Model (Source [?])**

### 2.2 Validation criteria for exascale proxy architecture

We implemented the proxy machine model in Aspen that represents the theoretical homogeneous abstract machine model presented by Ang et al. [?]. Previously we used the proxy architecture for testing applications. The proxy architecture needs to be validated to show that whether the peak theoretical performance values are representative of what was expected by the exascale computing project by Ang et al. [?]. We have developed a set of micro-kernel benchmarks that saturates the machine for the required performance metric under observation in Aspen. The list of a set of micro kernel benchmarks and their brief explanation is as follows.

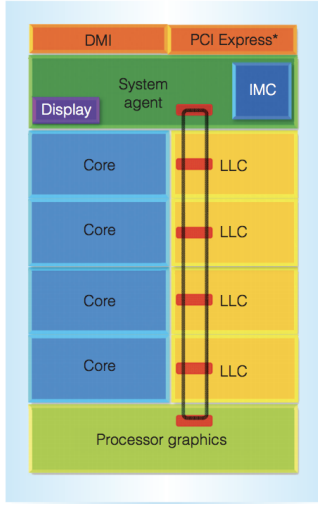


**Figure 2: Conceptual diagram of Intel's Haswell processor**  
(Source [? ])

**Table 1: Theoretical abstract machine model configurations [? ]**

| Parameter                              | Range         |
|----------------------------------------|---------------|
| Number of nodes                        | 33K-125K      |
| Number of cores (per chip)             | 64-256        |
| Processor cores/node                   | 256           |
| NUMA region/node                       | 64            |
| Gflops/per proc core                   | 64            |
| Processor SIMD vectors (units X width) | 8X16          |
| Off-chip memory B.W<br>Chip-memory     | 60-100 GB/s   |
| On-Chip memory B.W<br>Chip-memory      | 600-1200 GB/s |
| Network-on-chip B.W                    | 8-64 GB/s     |
| Per Core                               |               |
| In-package memory capacity             | 32-64 GB      |
| Off-chip memory capacity               | ~512 GB-2TB   |

- *Theoretical maximum single-precision Flops*  
Provides the maximum single-precision Flops produced by the overall system composed of ~ 125K nodes.
- *Theoretical maximum double-precision Flops*  
Provides the maximum double-precision Flops produced by overall system composed of ~ 125K nodes.
- *Theoretical maximum single-precision Flops per Joules*  
The energy consumed by the system when working on theoretical maximum single-precision floating-point operations.

- *Theoretical maximum double-precision Flops per Joules*  
The energy consumed by the system when working on theoretical maximum double-precision floating-point operations.
- *Theoretical maximum Memory bandwidth*  
The achievable maximum memory bandwidth provided by the system.
- *Theoretical maximum Memory bandwidth per Joules*  
An estimate of the energy consumed while providing maximum memory bandwidth.
- *Roofline analysis*  
A basic Roofline model showing computational and memory bounds of the system.

We tested our proxy homogeneous machine model in Aspen with the micro-kernel benchmarks for verification before deploying the system for the execution of the proxy application. The details of the verification results are shown in Section 3.

### 2.3 Proxy application for exascale architecture

We tested CoMD on the homogeneous proxy architecture in Aspen DSL environment to explore the execution pattern as well as communication and computation requirements of the application. First we will explain computation and communication models and later we will explain the mapping framework that builds on top of these analytical models.

**2.3.1 CoMD application characteristics.** CoMD is the co-design of molecular dynamics application code. It is a simulation code that focuses on the potential calculation among the elements. It performs two computation intensive operations i.e., discovery of pairs of atoms within a distance and calculation of force between those atoms. Data (in terms of atoms) in CoMD is represented in three dimensions (Nx, Ny, Nz). A few of the optimization strategies are defined to improve the performance of CoMD, including the division of the problem area into multiple small areas, where the number of areas can be decided such as it is equal to the number of MPI tasks. Individual atoms are grouped based on their physical location, and this requires computations related to force, velocity and position calculations within its circle and around the vicinity of the atom.

### 2.4 Abstract application model of CoMD in Aspen

The main emphasis in CoMD is at the spatial decomposition of atoms [? ] into small boxes. The CoMD code provided by ExMatEX and implemented in Aspen in abstract form requires data to be decomposed into dimensions equal to the number of processors, which helps to avoid load imbalance issues. In Aspen abstract application model, we distinguish the input elements as atoms within the domain of the processor and the neighboring atoms, in a similar way that ExMatEX implements CoMD. Each processor calculates the atom's velocity, position and distance with other neighboring elements. All the information related to the atoms are stored in the form of a structure of arrays, consisting of atom's bond, interaction with neighboring atoms, angle, etc. The abstract application model follows a number of steps discussed in detail in the next section.

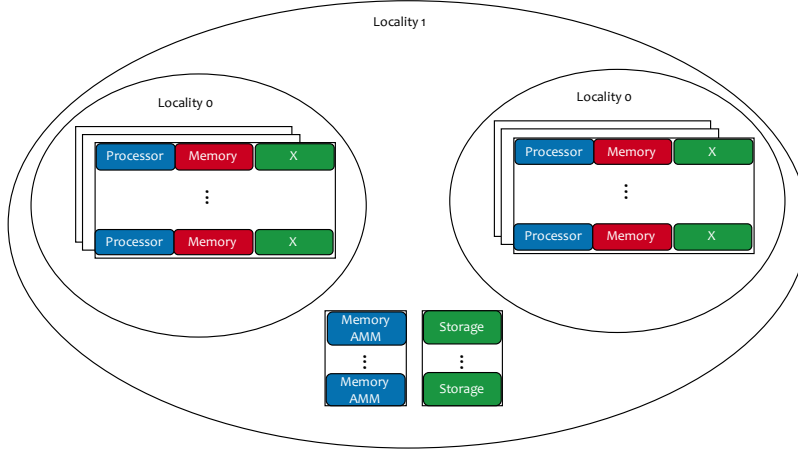


Figure 3: System abstract machine model

## 2.5 Communication & computation modeling and mapping framework in Aspen

Figure 4 shows the block diagram for all the steps required to calculate the computation and communication constraints of an application and to execute the mapping of abstract application model on the proxy homogeneous exascale architecture on Aspen. The main purpose of this framework is to map communication and computation requirements of the application with the machine model. The mapping framework consists of the following steps:

- (1) Abstract application and machine model generation in Aspen.
- (2) Extract computational and communication parameters from the Aspen abstract application model.
- (3) Compute computation and communication models that predict the patterns of execution on single and multiple nodes.
- (4) Generate a mapping framework that maps the abstract application with machine model on the exascale architecture and proposes the optimal number of ranks and other communication optimizations that can improve application execution.

A description of each step is given below:

**2.5.1 Step#1: Abstract application and machine model.** The application and machine model need to be represented as an abstraction [?] before being tested for the mapping framework. An example of steps implemented in the abstract application model for CoMD is shown in Algorithm-1.

**2.5.2 Step#2: Parameter extraction for computation and communication model.** In this step, we extract the application's parameters that are required to implement computational and communicational modeling. We extract Flops of the application, amount of total data exchanged during application execution, memory requirements, input size and the complexity of the application to use in the computation model. We also extract some of the machine characteristics i.e., number of processors used and the bandwidth of the memory – that will play a key role in the communication of the application on multiple processing elements.

**Input:** Atoms per cell, # of processors

**Result:** position, velocity, clustering

**while all processors do**

Calculate position;  
 Communicate with neighbors;  
 Calculate bonded & un-bonded forces;  
 Calculate velocity;  
 Calculate position;  
 Communicate if necessary;

**end**

**Algorithm 1:** CoMD abstract application model in Aspen

**2.5.3 Step#3: Computation and communication modeling.** This section describes the computation and communication model based on the extracted applications and machine characteristics. A common method for accurate performance modeling is to monitor performance counters. However, since the hardware for exascale does not exist, we implement a system level performance model. Our model takes into account computation and communication aspects of the application. Our model uses applications complexity, number of memory accesses, and time taken by the computation to populate the model. We consider computation and communication as separate entities to find the total time taken by the application to execute.  $T_{total}$  defines the time taken by the application to execute on proxy homogeneous architecture.  $T_{comp}$  &  $T_{comm}$  are the time taken during computation and communication phases. We calculate the time taken by the application as the *sum* of computation and communication.

$$T_{total} = T_{comp} + T_{comm} \quad (1)$$

**Computation model.** We partition the time taken by computation into time to execute the instructions (Flops) and time spent in accessing the memory. Inspired by Czechowski et al. [?], we calculate the flops and memory costs in terms of  $T_{flops}$  and  $T_{mem}$ . We use the *max* function between flops and memory, in a similar way that Aspen calculates the time taken by the application i.e., maximum of the time spent using processor or memory.

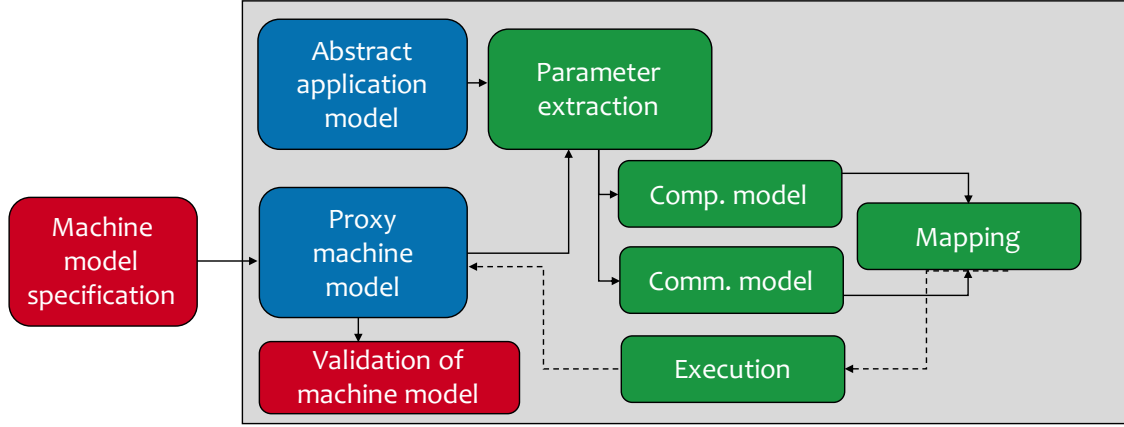


Figure 4: Mapping framework in Aspen

$$T_{comp} = \max(T_{flops} + T_{mem}) \quad (2)$$

Each floating point operation in CoMD operates in three dimensions, so the input data is represented in three-dimensions i.e., ( $N_x$ ,  $N_y$  and  $N_z$ ). This three-dimensional data ( $N_x$ ,  $N_y$ ,  $N_z$ ) is distributed and operated by each processing element ( $P$ ), which represents the first fraction of the equation 3 on the amount of work performed by each processing element( $P$ ).

The second fraction of the equation 3 defines the flops executed by each processor in terms of an exponential function of the total input size i.e., *AtomsPerCell*. We divide the total input by the maximum theoretical flops provided by each processor ( $P_{peakFlops}$ ).

Since we are targeting a homogeneous CPU based exascale architecture, we assume that the peak theoretical Flops provided by each processor is the same. In the case of heterogeneous architectures (e.g., GPUs, FPGAs), multiple flops cost estimations must be used, corresponding to the peak theoretical flops provided by each processing unit.

$$T_{flops} = \frac{N_x * N_y * N_z}{P} * \frac{\log(AtomsPerCell)}{P_{peakFlops}} \quad (3)$$

In case of cost of the memory operations, we partition total input size i.e., (*AtomsPerCell*) into three dimensions ( $N_x$ ,  $N_y$ ,  $N_z$ ) distributed to each processor ( $P$ ), which represents the first fraction of equation 4.

The second fraction in equation 4 represents memory operations performed by each processor on total input size i.e., (*AtomsPerCell*). This factor is dependent upon the peak theoretical bisection bandwidth of the memory  $\beta_{mem}$ .

$$T_{mem} = \frac{N_x * N_y * N_z}{P} * \frac{\log(AtomsPerCell)}{\beta_{mem}} \quad (4)$$

**Communication model.** We implement an analytical model for communication between various domains of the proxy exascale architecture. We calculate the time taken during communication ( $T_{comm}$ ) as the time taken for communication between the nodes ( $T_{intranode}$ ) and within the node ( $T_{internode}$ ).

$$T_{comm} = T_{intranode} + T_{internode} \quad (5)$$

In case of communication within the node, the amount of data transferred is in terms of total atoms i.e., ( $N$ ). CoMD does not encounter any load imbalance as the amount of data, (*AtomsPerCell*) is always the same that is transferred to each processing element. We take the product of the data that is communicated ( $2^{1/N} * \log(AtomsPerCell, 10)$ ) with the time ( $t$ ) taken to transfer it in order to calculate the communication within the node ( $T_{intranode}$ ). We divide the data communicated by the theoretical bandwidth, where  $\beta_{mem}$  is the bisection bandwidth of the memory. Since all the nodes have the same theoretical peak bandwidth, the same  $\beta_{mem}$  is used to compute the time taken during the intra node communication.

$$T_{intranode} = \frac{2^{1/N} * \log(AtomsPerCell, 10)}{\beta_{mem}} * t \quad (6)$$

The time taken for communication between the nodes is represented as  $T_{internode}$ . It is the combination of network latency in micro/nano seconds and time taken during intra node communication ( $t$ ). The latency of the interconnect is represented as  $\ell$ , and its value is provided by Aspen as the theoretical limit of the interconnect.

$$T_{internode} = \ell + T_{intranode} \quad (7)$$

**2.5.4 Step#4: Mapping framework.** After generating communication and computation models for CoMD, we test the application with varying number of processors in order to find the number of processors that provides the optimal execution in terms of communication and computation. While executing the application in Aspen, we also monitor the dominant communication calls, and provide the recommendations regarding optimizing the dominant communication calls, considering the scale of the future architectures.

### 3 RESULTS AND ANALYSIS

The main strength of the analytical modeling and mapping framework is to model the communication and computation patterns of the application on the proxy architecture. It also helps to identify the near optimal number of ranks that can be used in an exascale

**Table 2: Theoretical peak validations of exascale proxy architecture provided by Aspen**

| Parameters              | Aspen                 |
|-------------------------|-----------------------|
| SP Flops                | ~1.47 ExaFlops        |
| DP Flops                | ~0.736 ExaFlops       |
| SP/Joules               | ~10833.3 GFlops/Joule |
| DP/Joules               | ~5416 GFlops/Joule    |
| Memory Bandwidth        | ~592 GB               |
| Memory Bandwidth/Joules | ~1.1636 GB/Joule      |

homogeneous architecture consisting of millions of compute processors. To show the effectiveness of the modeling and mapping framework, we present two types of results in this section. We start with presenting the theoretical limits of the exascale homogeneous architecture that matches with the values proposed by the exascale computing project. After validating the theoretical configurations of the architecture, we use it to test our modeling framework on the CoMD proxy application for strong and weak scaling results. We also study the effect of dominant MPI communication calls and their behavior on a scale of millions of nodes.

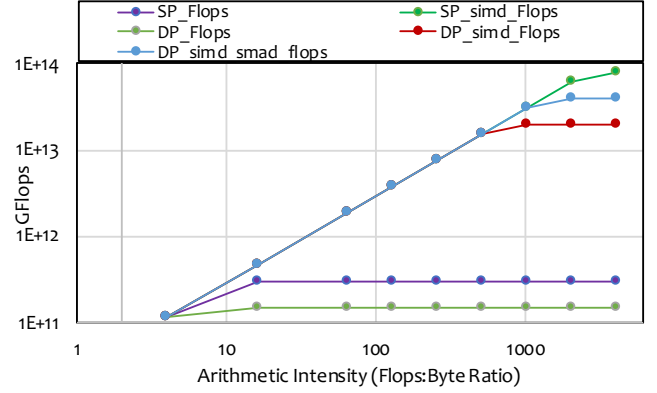
### 3.1 Evaluation of DOE's exascale homogeneous architecture

We first test the theoretical peak performance of the proxy exascale architecture in order to see whether it matches with those proposed by the Exascale Computing Project [?]. Table 2 shows the values obtained by micro-benchmarks provided by Aspen and the extensions that we implemented in the domain specific language. DOE proposes the theoretical Flops of the exascale machine to be ~1-ExaFlops. Our proxy architecture shows the single-precision and double precision flops to be 1.47 & 0.736 ExaFlops respectively. While implementing real hardware, a number of factors will decide the true configuration of the system, and therefore system configurations (i.e., number of cores, number of sockets) might have different configurations when implementing at exascale. We also calculate the energy consumed to produce one single and double precision ExaFlops. The maximum bandwidth provided by memory is ~592GB as proposed by the Exascale Computing Project. As can be seen from the Table 2, the theoretical peak performance matches the one proposed by the Exascale Computing Project, therefore we can test it for the execution of the application.

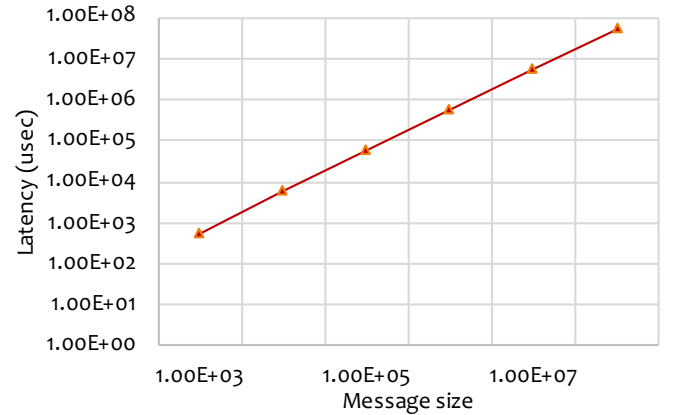
One might argue that the future exascale architectures will likely be heterogeneous containing multiple types of accelerators while our analysis in this paper is focused on homogeneous architectures. Though our study focuses on homogeneous exascale architecture we wanted to convey two observations. First, regardless of the scale, homogeneous CPU based systems will play a key role in the exascale system and their evaluation using Aspen shows they provide the capabilities proposed by the Exascale Computing Project. Secondly, we wanted to underscore Aspen's potential in terms of implementing and testing future architectures.

Figure 5 shows the Roofline analysis of the proxy homogeneous architecture implemented in Aspen. The Roofline model defines the upper limit on kernel performance  $P_k$  with the equation

$$P_k = \min(P_f, \beta A_i) \quad (8)$$

**Figure 5: Roofline analysis of proxy homogeneous exascale architecture**

where  $P_f$  is the maximum floating point operation provided by the system,  $\beta$  is the maximum theoretical bandwidth provided by the system, and  $A_i$  is the theoretical Flops to Byte ratio of the system. Identifying the Flops to Byte ratio and achievable GFlops of an application and comparing it with the Roofline plot helps to find limits of the system and whether any application will be compute or memory bound. Our analysis of CoMD using the Roofline model shows that CoMD behaves as a compute intensive application on the abstract homogeneous exascale architecture.

**Figure 6: Effect on latency with increasing message size**

We also studied the effect of increasing the number of messages with respect to the runtime. Figure 6 shows the effect of increasing message sizes on the overall latency of the system. This information is particularly useful as it will play a key role in deciding the bottlenecks of the machine and predicting the communication time of the application.

### 3.2 Communication and Computation modeling of CoMD

We executed CoMD on the proxy homogeneous exascale architecture to study the communication and computation effect on the overall application execution. We calculated the Flops per Byte

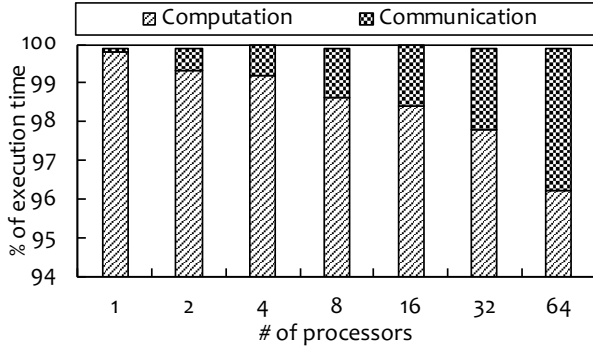


**Figure 7: Profiling of strong scaling of CoMD using Intel's Trace Analyzer and Collector**

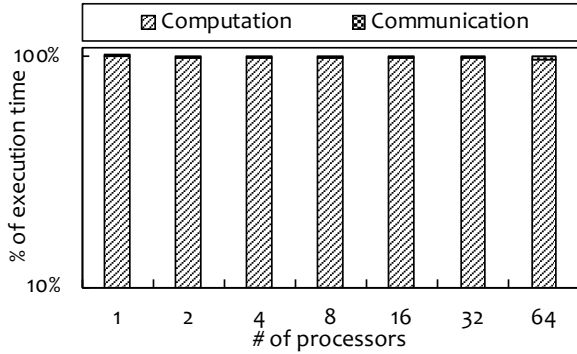


**Figure 8: Profiling of weak scaling of CoMD using Intel's Trace Analyzer and Collector**

ratio and theoretical Flops of CoMD; we compared this information with the Roofline of the system (Figure 5). Our analysis shows that CoMD is a compute intensive application for the processes within the node, while communication overhead starts to emerge for executions between a number of nodes. In order to verify this insight, we profiled CoMD on NERSC's Cori using Intel's Trace Analyzer and Collector (ITAC) for strong and weak scaling. Figure 7 and Figure 8 show the distribution between computation and communication on strong and weak scaling of CoMD using Intel's Trace Analyzer and Collector. The computation plays a major part in the overall application execution time for strong scaling, while the communication effects are not very dominant in the case of weak scaling.

We also profiled the CoMD communication patterns in Aspen in order to find out the dominant communication routines using Intel's Trace Analyzer and Collector. *MPI\_Barrier*, *MPI\_Bcast* and *MPI\_Allreduce* are the communication routines that take a significant amount of time during the communication phase of CoMD. While *MPI\_Bcast* is used to communicate the position and velocity of the atom at runtime, and it is supported by *MPI\_Barrier* in order to ensure a reliable communication. The assurance of reliability is not an issue on a small cluster, but considering the scale of future architectures, some optimization strategies need to be applied at runtime before issuing *MPI\_Bcast*. Similar situations exist for *MPI\_Allreduce* where the calculations from individual processes

are gathered to produce a converged result. Since *MPI\_Allreduce* is targeting millions of processing elements for future architectures, some optimization to break down the *MPI\_Allreduce* function into phases can help to avoid the overhead at exascale. One possible option is to introduce a virtual hierarchy in the cluster to parallelize the barrier function, it will also aid in recovering from failure to synchronize at larger scale. A failure to synchronize at larger scale will be more detrimental as compared to a small subset of the system.

Figures 9 and 10 show the strong and weak scaling for 8-Million processes. In the case of strong scaling, execution time continues to decrease with increasing number of processes until the point where we experience intra-node communication overhead which causes an increase in execution time. In terms of weak scaling, execution time continues to increase with increasing processes as the amount of work along with communication costs increase with increasing processes.

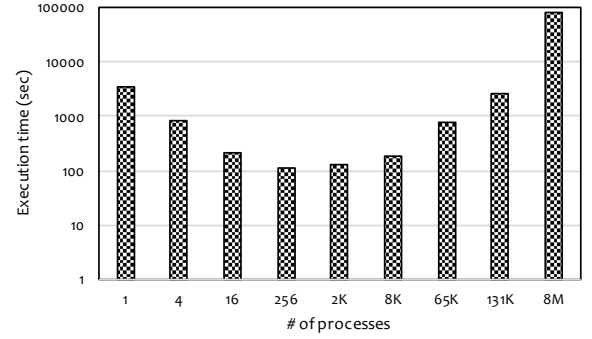


**Figure 9: Strong Scaling on 8 Million processes**

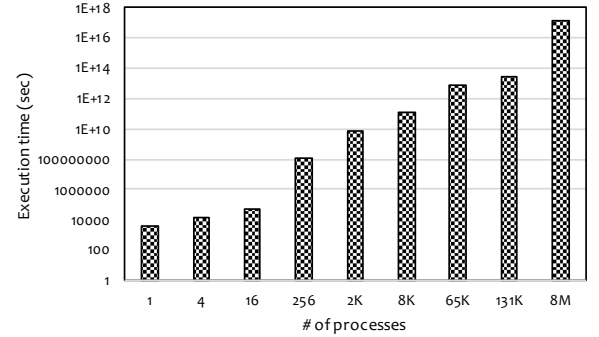


**Figure 10: Weak Scaling on 8 Million processes**

Figure 11 shows the comparison between measured and predicted values of CoMD for the strong scaling case for an input size of 64000 atoms. Measured values were obtained from NERSC's Cori leadership computing facility. The two most time consuming components of CoMD are *position calculation* kernel and the *communication* between processes. Our models under predict as they do not take into account the system noise in the background. As CoMD's execution time improves with increasing number of processes, increased processor allocation is one technique to improve the application execution. This holds until the point where

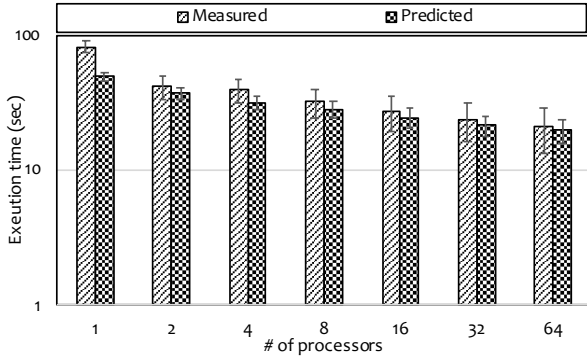


**Figure 11: Comparison between measured and predicted values for strong scaling of CoMD**

communication overhead is greater than the improvement achieved with increasing processes. In the case of exascale homogeneous architecture, it is important to consider the number of processes with respect to input size, as allocating more processes than required will result in over-subscription, and provides no benefit.

Our communication and computation model tests the application with increasing number of processes to see the optimal number of ranks for a given execution environment. Figure 12 shows the comparison between the values measured at NERSC Cori and the predicted values on the proxy exascale architecture implemented in Aspen. Since the number of messages transmitted during each iteration increases with increasing processes in weak scaling, communication overhead increases resulting in increased runtime and we do not observe any performance improvement in weak scaling. The effect of weak scaling in measured values tracks closely with the predicted values in Aspen.

Programming languages and models can also be used to take advantage of the data locality in complex memory architectures expected in exascale systems. Some notable directive based programming models(e.g. Lee et al. [? ]) mention the available but yet to be explored ventures of programming to improve memory bandwidth performance and to improve performance of CoMD like nearest-neighbor applications for weak scaling.

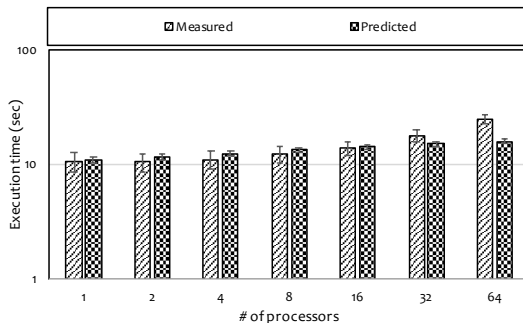


**Figure 12: Comparison between measured and predicted values for weak scaling of CoMD**

## 4 RELATED WORK

Future exascale architectures are expected to be extremely scalar and complex as compared to current high-performance computing system. Therefore, either we have to shape the existing performance and energy models to provide the scale needed for extreme-scale architectures or we design new modeling techniques for communication and computation that scales very well with the minimum overhead. Some communication modeling and mapping frameworks have been introduced e.g., Qilin. Qilin [?] proposes an automatic mapping of computation to a heterogeneous CPU-GPU architecture. Qilin mapping algorithms have been designed to cautiously distribute the work between the devices in order to achieve the improvement in time and energy consumption. ScalaTrace [?] reduces the cost of calculating the communication behavior of large-scale systems. They compress intra and inter node communication of MPI events that contains the information of an application's communication behavior. They also include the capability to replay the traces generated by the applications communication behavior. Liu et al. [?] proposes power and time efficient communication and computation modeling techniques by applying DVFS on CPU and GPU. Ahmed et al. [?] presents a communication model for predicting performance of extreme scale applications on Cray's Gemini 3-D torus interconnect network. They integrated their analytical model with MPI that provides a granularity of packet level on the interconnect.

In terms of modeling communication on a large scale system, it is important to consider both intra and inter node communication. [?] studies the detailed analysis of intra and inter node communication of parallel wavefront stencil algorithm. [?] studies the analysis of expected latency and bandwidth requirements for exascale architectures using LogGP modeling of fast fourier transform. Faraj et al. [?] studies the effects of MPI collective algorithms at a scale of 16K nodes on IBM BG/P system. Some tracing tools also capture the communication events in large-scale systems e.g., DUMPI [?], as an integral component of Structural Simulation Toolkit and collect the communication traces of mini-applications. Similar to DUMPI is an extension in CODES simulator called TRACER, that predicts the network performance and interconnect behavior by simulating the messages on ROSS. A compiler based solution to identify communication behavior is proposed in [?], that uses a matching mechanism of an application compilers traces with the already characterized communication pattern. Their framework also distinguishes between static and dynamic communication patterns.

Simulators are also employed to study and model the communication patterns on large scale systems. Durango [?] combines Aspen benefits with ROSS compiler to generate the scalable workload, with added features for the communication on extreme scale, using the real applications. A number of other simulators are also deployed to study the communication behaviors e.g., Omnet++ [?], SST [?], Dimemas [?], Bigsim [?], Booksim [?].

## 5 CONCLUSIONS AND FUTURE WORK

The exascale architecture implementation is still in its infancy, but a group of abstract machine models are proposed to represent the prospective architectures of the future. In this paper, we implement, validate and use one of the homogeneous architectures proposed

by the Exascale Computing Project in Aspen DSL. We evaluate the proxy architecture to test for theoretical peaks. We implement a communication and computation mapping model in Aspen to test it for exascale proxy application and architecture. We also study the strong and weak scaling effects of the application characteristics of CoMD on communication and computation at the scale of future architectures.

## REFERENCES

- [1] Kishwar Ahmed, Mohammad Obaida, Jason Liu, Stephan Eidenbenz, Nandakishore Santhi, and Guillaume Chapuis. 2016. An integrated interconnection network model for large-scale performance prediction. In *Proceedings of the 2016 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. ACM, 177–187.
- [2] JA Ang and Richard F Barrett. 2014. Abstract machine models and proxy architectures for exascale computing. In *Hardware-Software Co-Design for High Performance Computing (Co-HPC)*, 2014.
- [3] Claude L Bertin and Erik L Hedberg. 1999. High performance, high bandwidth memory bus architecture utilizing SDRAMs. US Patent 5,870,350.
- [4] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. 2011. The Gem5 Simulator. *ACM SIGARCH* (2011).
- [5] Christopher D Carothers, Jeremy S Meredith, Mark P Blanco, Jeffrey S Vetter, Misbah Mubarak, Justin LaPre, and Shirley 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*. ACM, 97–108.
- [6] Anthony Chan, Pavan Balaji, William Gropp, and Rajeev Thakur. 2008. Communication analysis of parallel 3D FFT for flat cartesian meshes on large Blue Gene systems. In *International Conference on High-Performance Computing*. Springer, 350–364.
- [7] Kenneth Czechowski, Casey Battaglini, Chris McClanahan, Kartik Iyer, P-K Yeung, and Richard Vuduc. 2012. On the communication complexity of 3D FFTs and its implications for exascale. In *Proceedings of the 26th ACM international conference on Supercomputing*. ACM, 205–214.
- [8] James Demmel, Mark Hoemmen, Marghoob Mohiyuddin, and Katherine Yelick. 2008. Avoiding communication in sparse matrix computations. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*. IEEE, 1–12.
- [9] Ahmad Faraj, Sameer Kumar, Brian Smith, Amith Mamidala, and John Gunnel. 2009. MPI collective communications on the Blue Gene/P supercomputer: Algorithms and optimizations. In *High Performance Interconnects, 2009. HOTI 2009. 17th IEEE Symposium on*. IEEE, 63–72.
- [10] Rosa Filgueira, David E Singh, Jesús Carretero, Alejandro Calderón, and Félix García. 2011. Adaptive-CoMPI: Enhancing MPI-based applications' performance and scalability by using adaptive compression. *The International Journal of High Performance Computing Applications* 25, 1 (2011), 93–114.
- [11] Hormozd Gahvari and William Gropp. 2010. An introductory exascale feasibility study for FFTs and multigrid. In *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*. IEEE, 1–9.
- [12] Georgios Goumas, Aristidis Sotiropoulos, and Nectarios Koziris. 2001. Minimizing completion time for loop tiling with computation and communication overlapping. In *Parallel and Distributed Processing Symposium., Proceedings 15th International*. IEEE, 10–pp.
- [13] Per Hammarlund, Alberto J Martinez, Atiq A Bajwa, David L Hill, Erik Hallnor, Hong Jiang, Martin Dixon, Michael Derr, Mikal Hunsaker, Rajesh Kumar, et al. 2014. Haswell: The fourth-generation intel core processor. *IEEE Micro* 34, 2 (2014), 6–20.
- [14] Curtis L Janssen, Helgi Adalsteinsson, and Joseph P Kenny. 2011. Using simulation to design extremescale applications and architectures: programming model exploration. *ACM SIGMETRICS Performance Evaluation Review* 38, 4 (2011), 4–8.
- [15] Nan Jiang, George Michelogiannakis, Daniel Becker, Brian Towles, and William J Dally. 2010. Booksim 2.0 user's guide. *Stanford University* (2010).
- [16] Sameer Kumar, Yogish Sabharwal, Rahul Garg, and Philip Heidelberger. 2008. Optimization of all-to-all communication on the blue gene/l supercomputer. In *Parallel Processing, 2008. ICPP'08. 37th International Conference on*. IEEE, 320–329.
- [17] Seyong Lee and Jeffrey S. Vetter. 2014. OpenARC: Open Accelerator Research Compiler for Directive-based, Efficient Heterogeneous Computing. In *23rd International Symposium on High-performance Parallel and Distributed Computing*.
- [18] Cong Liu, Jian Li, Wei Huang, Juan Rubio, Evan Speight, and Xiaozhu Lin. 2012. Power-efficient time-sensitive mapping in heterogeneous systems. In *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*. ACM, 23–32.
- [19] Chi-Keung Luk, Sunpyo Hong, and Hyesoon Kim. 2009. Qilin: exploiting parallelism on heterogeneous multiprocessors with adaptive mapping. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 45–55.
- [20] Matthias Maiterth, Martin Schulz, Dieter Kranzlmüller, and Barry Rountree. 2016. Power Balancing in an Emulated Exascale Environment. In *IEEE IPDPS*.
- [21] Paul Messina. 2017. The Exascale Computing Project. *Computing in Science & Engineering* 19, 3 (2017), 63–67.
- [22] Michael Noeth, Prasun Ratn, Frank Mueller, Martin Schulz, and Bronis R de Supinski. 2009. ScalaTrace: Scalable compression and replay of communication traces for high-performance computing. *J. Parallel and Distrib. Comput.* 69, 8 (2009), 696–710.
- [23] Steve Plimpton, Roy Pollock, and Mark Stevens. 1997. Particle-Mesh Ewald and rRESPA for Parallel Molecular Dynamics Simulations. In *PPSC*. Citeseer.
- [24] Arun F Rodrigues, K Scott Hemmert, Brian W Barrett, Chad Kersey, Ron Oldfield, Marlo Weston, R Risen, Jeanine Cook, Paul Rosenfeld, E CooperBalls, et al. 2011. The Structural Simulation Toolkit. *ACM SIGMETRICS Performance Evaluation Review* (2011).
- [25] German Rodriguez, Rosa M Badia, and Jesús Labarta. 2004. Generation of simple analytical models for message passing applications. In *European Conference on Parallel Processing*. Springer, 183–188.
- [26] Shuyi Shao, Alex K Jones, and Rami Melhem. 2006. A compiler-based communication analysis approach for multiprocessor systems. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*. IEEE, 10–pp.
- [27] Kyle L. Spafford and Jeffrey S. Vetter. 2012. Aspen: A Domain Specific Language for Performance Modeling. In *International Conference on High Performance Computing, Networking, Storage and Analysis (Supercomputing '12)*. IEEE Computer Society Press, 11. <http://dl.acm.org/citation.cfm?id=2388996.2389110>
- [28] Didem Unat, Cy Chan, Weiqun Zhang, Samuel Williams, John Bachan, John Bell, and John Shalf. 2015. ExaSAT: An Exascale Co-Design Tool for Performance Modeling. *International Journal of High Performance Computing Applications* (2015).
- [29] András Varga and Rudolf Hornig. 2008. An overview of the OMNeT++ simulation environment. In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 60.
- [30] Thiruvengadam Vijayaraghavany, Yasuko Eckert, Gabriel H Loh, Michael J Schulte, Mike Ignatowski, Bradford M Beckmann, William C Brantley, Joseph L Greathouse, Wei Huang, Arun Karunanithi, et al. 2017. Design and Analysis of an APU for Exascale Computing. In *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*. IEEE, 85–96.
- [31] Oreste Villa, Daniel R Johnson, Mike Oconnor, Evgeny Bolotin, David Nellans, Justin Luitjens, Nikolai Sakharov, Peng Wang, Paulius Micikevicius, Anthony Scudiero, et al. 2014. Scaling the power wall: a path to exascale. In *High Performance Computing, Networking, Storage and Analysis, SC14: International Conference for*. IEEE, 830–841.
- [32] Gengbin Zheng, Gunavardhan Kakulapati, and Laxmikant V Kalé. 2004. Bigsim: A parallel simulator for performance prediction of extremely large parallel machines. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*. IEEE, 78.