Aspen: A Domain Specific Language for Performance Modeling

Kyle L. Spafford
Oak Ridge National Laboratory
spaffordkl@ornl.gov

Jeffrey S. Vetter
Oak Ridge National Laboratory
Georgia Institute of Technology
vetter@computer.org

***Abstract*—We present a new approach to analytical performance modeling using Aspen, a domain specific langauge. Aspen (Abstract Scalable Performance Engineering Notation) fills an important gap in existing performance modeling techniques and is designed to enable rapid exploration of new algorithms and architectures. It includes a formal specification of an application's performance behavior and an abstract machine model. We provide an overview of Aspen's features and demonstrate how it can be used to express a performance model for a three dimensional Fast Fourier Transform. We then demonstrate the composability and modularity of Aspen by importing and reusing the FFT model in a molecular dynamics model. We have also created a number of tools that allow scientists to balance application and system factors quickly and accurately.**

I. Introduction

Projections from various reports including the International Exascale Software Project (IESP) [1] and the DARPA Exascale Study [2] indicate that power will be the major constraint for an exascale computer. In order to provide the required orders-of-magnitude increase in energy efficiency, architects are turning toward system designs that include significantly more parallelism (estimated at billion-way concurrency by the IESP), reduced memory capacity and bandwidth, and, in many cases, heterogeneous processing elements.

Meanwhile, application scientists are tasked with developing algorithms that use these undefined architectures, while simultaneously informing the architects about probable algorithmic directions and requirements. This situation is exacerbated by the fact that these new applications will have to, in turn, exploit or contend with architectural characteristics that are projected to be substantially different than existing architectures.

Given the long timescale and relatively high uncertainty in application design and system architecture, scientists are using a broad spectrum of predictive design techniques. However, these techniques, discussed in §II, have technical limitations that can impede their use in these uncertain scenarios. For example, cycle-accurate simulation, one of the most effective techniques for predicting the performance of new architectures, is challenging due to the lack of a detailed specification of the architecture, software stack, and programming model. Even functional simulation becomes difficult, as source code or instruction traces for proposed algorithms may not yet be available. These limitations also apply to other methods that require a detailed specification of the target architecture, including hardware emulation. Aside from simulation, scientists often rely on analytical modeling because it provides a flexible, fast estimate without the need for exact, low-level details. Oftentimes, however, advantages come at the cost of accuracy. Equally frustrating is the fact that the community lacks a common, structured analytical modeling methodology that allows composable models to be created, refined, exchanged, reused, and maintained for complex applications and architectures being designed by large teams.

A. Aspen

In order to address these limitations, we have designed a prototype system for the analytical modeling of Exascale applications and architectures. Aspen (Abstract Scalable Performance Engineering Notation) is a modeling framework that is based on a domain specific language (DSL). Scientists write structured models of their applications and architectures using Aspen's DSL. Aspen specifies a formal grammar for describing two types of models. The first type describes an application's behaviors including available parallelism, operation counts, data structures, and control flow. The second type specifies an abstract machine model. These model descriptions are compiled (converted to an intermediate representation in memory), checked for semantic correctness, and, then, queried or simulated for a variety of modeling scenarios of interest to scientists.

Using a domain specific language for Aspen provides several advantages over traditional approaches to analytical modeling. First, Aspen's DSL constrains models to fit the formal language specification, which enforces similar concepts across models and allows for correctness checks. Second, because models are written in a language, Aspen provides formal rules for modularity such that models are easily composed, reused, and extended. Third, given this structure, Aspen promotes collaboration between application and computer scientists because they can share and refine models using a common set of concepts and tools. Finally, this structured approach allows the model to reflect the composition of applications, in that the Aspen language can express many of the same concepts that are used in software architecture.

However, Aspen is by no means a panacea for analytical modeling, and it does not aim to replace simulators or other methods. While more structured than traditional analytical models, Aspen still makes a number of limiting assumptions which will be illustrated in the following sections.

SC12, November 10-16, 2012, Salt Lake City, Utah, USA
978-1-4673-0806-9/12/$31.00

We anticipate the primary use of Aspen to be a facilitator for coarse-grained exploration of novel algorithms on new architectures. From this vantage point, Aspen will augment traditional simulation approaches until detailed architectural information and application source code is available.

Even though the majority of our discussion focuses on Aspen's language design and use, Aspen's design draws from widely-used analytical performance modeling techniques including the LogP model and its variants [3], [4] as well as the BSP model [5]. While these models traditionally capture computation and communication, Aspen's semantics also incorporate the factors of data movement and capacity, drawing largely on the characterizations of cache-oblivious algorithms [6]. Aspen's syntax for computation (or kernels) was heavily influenced by the requirement fitting approach described by Hoefler et al. [7], and its models for communication are based on algorithms used in MPI [8], [9]. In general, Aspen is a superset of these models.

B. Contributions

This paper makes the following contributions: (i) We provide an overview of the Aspen language features and characterize the advantages of a language-based approach over other, less systematic methods. (ii) We describe a suite of analysis tools which consume Aspen as input and generate results that allow users to identify application and architecture tradeoffs. (iii) We describe an analytical performance model for the three dimensional Fast Fourier Transform (3D FFT) and demonstrate the use of Aspen in exploring the relationship between the slab and pencil decompositions, the relative importance of flops, memory operations, and messages, and the effect of the interconnect topology at scale. (iv) We then demonstrate the modularity and composability of Aspen by importing this 3D FFT model into a model for molecular dynamics using the particle mesh Ewald (PME) method.

II. Background: Predictive Design Techniques

In order to make predictive design decisions, application scientists and architects must rely on a palette of modeling and simulation tools that span a range of metrics that balance accuracy, flexibility, performance, ease, and scalability. As illustrated in Table I, each technique has strengths and weaknesses, and one technique may be more appropriate at a specific time in the design lifecycle than at other times. In Table I, we define these metrics as follows. (a) *Speed* is the ability for a technique to generate a prediction, once the model, simulator, or system is built and configured. (b) *Ease* is the ability to create a framework for generating predictions in the given technique. (c) *Flexibility* is the ability to change the configuration or parameters in order to generate a new prediction. (d) *Accuracy* is the absolute and relative error of the prediction from the measurement on the same configuration of the final system (if built). Accuracy implies validation and verification. (e) *Scalability* is the ability of the prediction technique to generate predictions of increasingly larger collections of components.

Given these metrics, we believe that we can generally categorize existing techniques into three basic areas as follows: analytical modeling, simulation and emulation, and empirical evaluation. In this paper, we assume that the final system is the baseline for any prediction.

TABLE I
Ranking of Predictive Design Techniques (where 1 is best).

	Speed	Ease	Flexibility	Accuracy	Scalability
Ad-hoc Analytical Models	1	3	2	4	1
Structured Analytical Models	1	2	1	4	1
Aspen	1	1	1	4	1
Simulation – Functional	3	2	2	3	3
Simulation – Cycle Accurate	4	2	2	2	4
Hardware Emulation (FPGA)	3	3	3	2	3
Similar hardware measurement	2	1	4	2	2
Node Prototype	2	1	4	1	4
Prototype at Scale	2	1	4	1	2
Final System	-	-	-	-	-

1) Analytical Modeling: With *analytical modeling*, scientists create high-level abstractions of the application and architecture that can be evaluated easily and quickly. These models are often flexible and scalable, but they can often offer only limited accuracy. *Ad hoc analytical models* are developed using a variety of techniques, and are often created on-the-fly by scientists, to produce a prediction. These techniques include models created with back-of-the-envelope calculations, Excel spreadsheets, and in languages like Matlab and Python. In contrast, *structured analytical models* are developed using a predefined method or set of abstractions for generating a prediction. These approaches can be replicated, shared, and reused easily by multiple scientists. Furthermore, these approaches often satisfy properties of correctness that can be checked and verified independent of the specific model being developed.

Typically, these structured models use a reductive approach to hardware—reducing machine complexity to a small number of parameters (e.g. PRAM [10], LogP [3], and its variants [4], and BSP [5]). Machine parameters are then combined with asymptotic bounds on operation counts to predict performance. More broadly, this category also includes techniques like queuing theory and Markov models.

2) Simulation and Emulation: Simulation and emulation are very popular in computer architecture, examples include: SIMICS, GEMS, M5, GEM5, PTLsim, ASIM, SimOS, WWT, RSIM, FAST, SimpleScalar, MAMBO, COTSon, CACTI, Graphite, and SST [11]. The benefits of simulators are their flexibility and their low-level, high fidelity model of the target architecture. The cons of simulators and emulators are their predictive instability, infrastructure cost, and slow running time. *Functional Simulation (full-system)* usually provides an instruction-level simulation which accurately models executing individual instructions but does not provide timing behaviors that allow for low-level or application-level performance pre-

diction. *Cycle accurate simulation (full-system)* extends functional simulation behaviors to include timing information that allows for very accurate prediction of individual instructions, instruction blocks, and applications. *Hardware Emulation* provides similar capabilities of cycle-accurate simulation but with improved performance by employing hardware acceleration, often using FPGAs. Examples include RAMP, DEEP, BEE, and a host of proprietary emulators. An important consideration in our context is that the large majority of simulation and emulation work focuses on the processor or node level architectures, and not on scalable systems, such as those used in HPC. Notable exceptions include SST-Macro [12] and POEMS [13].

3) Empirical Evaluation: Finally, if prototype or similar hardware exists, researchers can port, measure, and explore their applications on these architectural testbeds to get a sense of how their applications will perform on the final system, and to verify and validate their earlier predictions. *Prototype* hardware is an early sample of the target architecture and software that allows execution of application software, even though it may have known limitations. *Prototype at Scale* is an early sample of the target architecture that includes an operational interconnect and the functionality to integrate the interconnection network with the node prototype. Typically, these early scalable prototypes are much smaller than the final system, but they provide scientists with the opportunity to explore and test the interconnection networks functionality and performance.

4) Combining methods: Furthermore, it is common for scientists to combine multiple predictive design techniques to arrive at a solution, as opposed to relying on one method exclusively. In particular, Snavely et al. [14] and Kerbyson et al. [15] have successfully blended analytical modeling, empirical measurement, and simulation to generate performance predictions.

In our taxonomy, Aspen falls between *structured analytical models* and *functional simulation*.

III. Motivating Example: 3D FFT

As a motivating example for Aspen, we select a three dimensional Fast Fourier Transform (3D FFT) because it is a reasonably complex kernel that has relevance to real applications, and we can compare and contrast our Aspen modeling approach to existing models in literature [9], [16], [17], [18], [19], [20].

As evidenced by this and other work, the 3D FFT is ubiquitous in scientific applications, and its strict requirements for a high performance interconnect make it an important algorithm for influencing exascale designs. Typically, the most studied aspects of the 3D FFT are its requirements for floating point computation and the scalability of its global, all-to-all communication. Some of the most salient questions from recent analyses include the following. 1) How does the arithmetic intensity of FFT change with problem size and machine characteristics? 2) What is the best way to decompose the problem among processors? 3) What level of performance must future interconnect hardware achieve to reach the exascale [16]? 4) What is the relative importance of floating point computation, memory bandwidth, and interconnect bandwidth? 5) How does this balance, combined with current trends in hardware, impact the design of an exascale machine [17]?

A. 3D FFT Overview

We examine the general case of the $n \times n \times n$ double complex 3D FFT on P processors, similar to the test in the NAS parallel benchmarks [21]. However, we do not constrain the model to the decomposition implemented in NAS. In general, there are two popular methods for decomposing the 3D FFT, slab and pencil. In a slab-based decomposition, each processor gets an $\frac{n}{P} \times n \times n$ subdomain, while in a pencil-based scheme, processors are organized into a 2D topology, and each processor gets an $\frac{n}{P_x} \times \frac{n}{P_y} \times n$ strip of the domain, as shown in Figure 1.

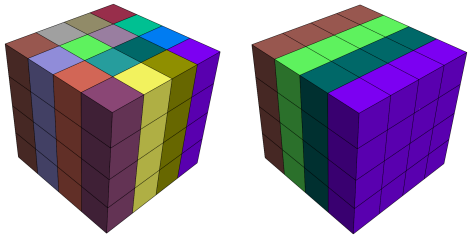

Fig. 1. Pencil (left) vs. Slab (right) decomposition of a 3D volume. Each processor's subdomain is shown in a different color.

In both decompositions, the general algorithm is to perform a 1D FFT in each of the x, y, and z dimensions. There are three main routines—a local 1D FFT, a transpose, and a global exchange of data—which proceed in rounds. In both decompositions, there are three rounds of local FFTs and transposes, corresponding to each dimension. A global exchange is only required for each of the dimensions across which a domain is split, so the pencil decomposition requires two exchanges while the slab decomposition only requires one. So, in general, while the slab decomposition requires less global communication, it allows for fewer total processors ($p \leq n$ as opposed to $p \leq n^2$) and uses more memory per processor.

Fortunately, there are known bounds for each of 3D FFT's subroutines. For any Cooley-Tukey style 1D FFT, the required number of floating point operations is bounded by

$$\mathcal{O}(5n \log_2 n) \tag{1}$$

and, the number of cache misses has been bounded for any FFT in the I/O complexity literature (on any two-level memory hierarchy which meets the tall cache assumption [6]) as $\mathcal{O}((1+(n/L)(1+\log_Z n)$, and for sufficiently large n, the number of cache misses, N_m becomes

$$N_m = n \max(\log_Z n, 1)a \tag{2}$$

where L is the cache line size in words, Z is the cache capacity, and a is a constant which transforms the bound to an

absolute count [6], [17]. The number of words transferred in each iteration of the transpose kernel is also characterized as $\mathcal{O}(2n^3)$, as the entire volume must be loaded, then stored. The same holds for the exchange operation, although the factor of two from loading and storing is removed.

Once the appropriate values are chosen for problem and cache sizes (i.e. n, Z, and L), the analytical model is almost capable of determining the arithmetic intensity (and similar ratios such as flops to messages) by dividing the appropriate bounds expressions. The remaining piece of required information is the value for a, a constant that arises from the nature of characterizing requirements by asymptotic bounds (e.g. $\mathcal{O}()$) [6]. Due to the complexity in modeling the memory hierarchy (e.g. from multi-level cache hierarchies, replacement policies, etc.) this type of constant is frequently measured using performance counters on an existing implementation of the algorithm to calibrate the model. It is a particularly common approach for characterizing memory traffic, even in the case of much simpler kernels, like matrix multiplication [7].

This type of calibration (or incorporating any parameters based on a specific architecture or implementation) tends to result in better accuracy, but introduces a limitation—the loss of generality in empirical results. Indeed, due to the rapid changes in computing hardware, most model predictions can become dated in just a few years. As such, it is desirable to publish some durable, parameterized version of a performance model to be shared and reused by the community.

To illustrate this limitation, consider the bound on the communication in the 3D FFT. A basic model for a fully connected network might predict runtime as

$$T = \alpha + \frac{n^3}{PB_{link}} \tag{3}$$

where α is link latency and B_{link} is link bandwidth. But, at large scales, a fully connected network is unlikely. A more realistic estimation of runtime must include the topology of the interconnect. Examples include

$$T = \frac{n^3}{P^{\frac{5}{6}}B_{link}} \tag{4}$$

for an ideal 3D torus [17], or

$$T = \frac{n^3}{P^{\frac{2}{3}}B_{link}} \tag{5}$$

for a 3D torus without task-aware process placement [17]. However, even these two bounds are simplifications, coming from a model concerned primarily with large scales where latency is insignificant. It becomes useful, then, to standardize models, such that any limiting assumptions are more explicit.

In other words, an artifact-driven analytical modeling framework, like Aspen, helps to mitigate the rapidly changing hardware landscape. Furthermore, it is possible for this need to arise in relatively simple analyses, as shown in the limiting factor analysis of 3D FFT.

B. Modeling 3D FFT in Aspen

While a full description of Aspen is beyond the scope of this paper, a formal definition, including an extended Backus-Naur form (EBNF) grammar, is available [22][1]. This section demonstrates the basic language concepts behind Aspen and shows how it can be used to express and extend the analysis of our motivating example, the 3D FFT.

1) Parameters and Data Objects: In order to model a 3D FFT, there are several requirements for Aspen. First, it must have the ability to describe basic quantities like n and a. This corresponds to an Aspen parameter, a named expression that can be a constant or derived value from other Aspen parameters. The parameters for 3D FFT are shown in Listing 1:

```
1 // Dimension of cubic 3D Volume
2 param n = 8192
3 param a = 6.3
4 param wordSize = 16 // Double Complex Words
5 param dataPerProc = (n^3 * wordSize) / P
6 data fftVolume [n^3 * wordSize]
```

Listing 1. Aspen statements for 3D FFT parameters

These include the symbols from the basic theoretical bounds like `n` and `a` (recently measured as approximately 6.3 [17], as well as two derived parameters, `wordSize` for converting from words to bytes and `dataPerProc`, a legible shorthand for the size of the data on each processor. Also shown in Listing 1 is the declaration of the `fftVolume` as a simple Aspen data object with an expression for its size [22].

2) Kernels: Aspen must also have some representation for a subroutine or basic grouping of work, which corresponds to the logical separation among the 1D FFT, transpose, and exchange. Kernels are Aspen's fundamental abstraction for a computation. They contain information about parallelism, operation counts, memory accesses, I/O, and communication. This information is contained in a set of *clauses*, which are statements written using expressions based on the application parameters. In most cases, the expressions used in clauses are very similar to traditional asymptoptic bounds ($\mathcal{O}()$ expressions, although constants are important for accuracy) or the requirement expressions described by Hoefler et al [7].

Each kernel must have a single *parallelism clause*, which describes the number of independent pieces of work. Each subsequent *requirement clause* details the amount of work contained per element of parallelism. The kernel description of the n^2 independent, local 1D FFTs is shown in Listing 2:

```
1 kernel localFFT {
2   exposes parallelism [n^2]
3   requires flops [5 * n * log2(n)] as dp, simd
4   requires loads [a * (n*wordSize) * max(1, log(
        n*wordSize)/log(Z))] from fftVolume
5 }
```

Listing 2. Aspen statements for the local 1D FFTs

[1]All Aspen models used in this paper as well as a more in-depth description of the implementation details of the Aspen tools are available in the Aspen Technical Report [22].

The bounds on floating point operations and memory accesses (in this case not differentiated between loads and stores for simplicity) in Equations 1 and 2 directly translate to Aspen requirement clauses. Machine dependent parameters such as Z and L can be given a default value in an application model (but are overriden when a machine model is part of a combined analysis). Note that the flops are specialized with two *traits*.

In Aspen, traits allow modelers to annotate kernel clauses (and data objects [22]) with rich semantic and performance information by appending a suffix to certain types of model elements. In this case, traits were used to characterize requirement clauses by annotating the `flops` requirement with `as dp, simd` and a memory access requirement with `from fftVolume`. These additions drastically increase the specificity of the requirement by indicating the flops are double precision and can exploit SIMD FPUs. In the context of the application model itself, these largely serve as labels (and allow querying, for instance, what percentage of an applications flops are SIMD?). However, when a machine model is introduced, their role becomes more important in performance prediction[2].

After the local 1D FFTs, data must be rearranged in memory before being exchanged with other nodes. During this second kernel, the local transpose, each processor's data is essentially streamed through memory, leading to a simple kernel shown in Listing 3.

Following the transpose, data is exchanged among processors in an all-to-all communication. This corresponds to an Aspen kernel with a requirement specifying the message volume customized with the `allToAll` trait. Aspen contains traits for all major MPI collective operations and a collection of typical interconnect topologies [22].

```
kernel transpose {
  exposes parallelism [P]
  requires loads  [dataPerProc] from fftVolume
  requires stores [dataPerProc] to fftVolume
}
kernel exchange {
  exposes parallelism [P]
  requires messages [(n^3 * wordSize) / P] as
      allToAll
}
```

Listing 3. Aspen statements for the transpose and exchange kernels

TABLE II
EXAMPLE CONTROL FLOWS

Figure	Aspen
2a	`A -> B`
2b	`iterate [2] {A}`
2c	`{A, B, C}`
2d	`map [3] {A}`
2e	`{iterate [2] {A}, B -> D, map [4] {C} -> E}`

3) Control Flow: Finally, the difference in control flow between the slab and pencil decomposition needs to be captured. This is done with the last major Aspen application construct, the control flow block. This control flow block implements

[2]An initial set of traits is defined in the Aspen Technical Report [22]

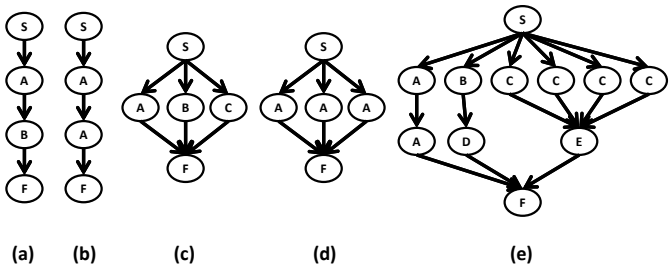

Fig. 2. Example control flow graphs, where **S** and **F** denote start and finish, and other letters correspond to kernels. One possible Aspen notation (each has several equivalent representations) for each of the graphs is given in Table II.

an augmented bulk synchronous parallel (BSP) approach to specifying *task* parallelism (as opposed to the data parallelism of the kernel requirement clauses). Similar to BSP [5], control flow is specified as a sequence of steps, with implicit synchronization between each step. In the simplest case, a step is a single kernel. In order to express more complex control flows, however, a step can be composed of any of the following (examples are shown in Table II and Fig. 2): (a) ***Dependency*** When a sequential dependency exists between two control flow atoms, a right arrow (a -> b), indicates that b depends on a. (b) ***IterateBlock*** When an atom is repeated with sequential dependence (as in timestep iteration), an expression is allowed which specifies the number of times the atom is repeated. This syntax is equivalent to a repetition of the dependency rule. (c) ***Superstep*** When one or more atoms can be executed in parallel (as in a BSP superstep), a comma delimited list of atoms is enclosed in braces. (d) ***MapBlock*** A map block is similar to the iterate block, except there is no sequential dependence. This, in effect, just produces a number of copies of an atom to execute in parallel. It is named after the similar concept from functional programming. (e) ***Control Flow*** Control flow blocks are modular and may contain references to other named control flows, shown in Figure 2 as a complex combination of steps.

The control flows for the pencil and slab decomposition are shown in Listing 4:

```
control slab {
  localFFT -> transpose // in X
  localFFT -> transpose // in Y
  exchange
  localFFT -> transpose // in Z
}
control pencil {
  localFFT -> transpose // in X
  exchange
  localFFT -> transpose // in Y
  exchange
  localFFT -> transpose // in Z
}
```

Listing 4. Aspen statements for 3D FFT control flow

4) Example Application Model Analysis: With the 3D FFT application model complete, we return to the question—what is the arithmetic intensity of FFT and how does it change as parameters are varied? It should be noted that any kernel or

application described in Aspen contains sufficient information to calculate these types of ratios. An analytical expression for the ratio can be derived by combining the requirement clauses of the same resource (e.g. flops, loads, messages) and adjusting by the appropriate factors from the control flow block.

From an inspection of the requirement clauses, we see that the important parameters for the arithmetic intensity of the `localFFT` kernel are problem size `n`, the constant, `a`, and cache capacity, `Z`. The resulting surfaces are shown in Figure 3, showing the rapid increase in intensity as problem size and cache capacity increase.

While these examples are the most interesting for 3D FFT, the application model facilitates a wide variety of analyses. For applications with more diverse data objects, for instance, the model is capable of analyzing total size, traits (and hence, frequency of access or access pattern), or size of data structures over time (through kernel temporary allocations or resize operations) [22].

C. Abstract Machine Model

While many analyses can be performed using only the application model, most performance models tend to target a particular architecture. As mentioned in Section III, Aspen aims to improve the generality of analytical models by keeping the machine model distinct from the application. This abstract machine model (AMM) is the second major component of Aspen.

For a given architecture, the machine model will describe how to map the *resources* (like flops or messages) that are required by an application to time. Similar to the application model, the machine model is composed of a hierarchy of concepts, shown in Figure 4. The next sections walk through construction of a machine model for the NSF Keeneland machine [23], a heterogeneous cluster containing Intel Westmere CPUs and NVIDIA Tesla M2090 GPUs, with a Mellanox QDR infiniband interconnection network.

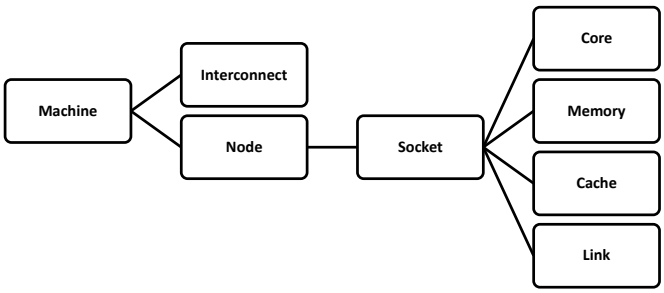

Fig. 4. The Aspen Abstract Machine Model (AMM) Concept Hierarchy

*a) **Parameters**:* Similar to the application model, a machine model can have parameters. These parameters can be constants or expressions derived from other parameters. For instance, consider GDDR5 memory (the type present in the M2090 GPU). The physical properties of this memory include clock rate, bus width, and whether or not ECC is enabled. A corresponding set of parameters showing how a high-level metric (bandwidth) is derived from basic physical properties is:

```
1 // Parameters for GDDR5
2 param gddr5Clock = 3700 * mega  // effective
3 param gddr5Width = 48
4 param eccPenalty = 0.75
5 param gddr5BW = gddr5Clock*gddr5Width*eccPenalty
```

Listing 5. Aspen Parameters for GDDR5 Memory

Similar compositions can be built for floating point computation, including core clock rate, SIMD width, and presence of the fused multiply-add instruction. Note that all models contain the relevant powers of ten (mega, giga, tera, etc.) "built-in" for increased readibility.

*b) **Machine and Interconnect**:* Machine is the top-level concept for the Aspen AMM, which represents a single computing system. A machine is defined as a collection of nodes, the type and topology of the interconnection network, and the number of connections to that network at each node (normally one or two, e.g. single-rail or dual-rail infiniband). An example for Keeneland is shown in Listing 6.

```
1 machine Keeneland {
2    [numNodes] sl390 nodes
3    connected with [numRails] of infiniband as
         fatTree
4 }
```

Listing 6. Aspen AMM Statement for Keeneland machine block

where `numNodes` and `numRails` are defined as parameters (120 and 1 in the case of Keeneland), and `sl390` and `infiniband` reference an Aspen node model and interconnect model, respectively. The Infiniband model is shown in Listing 7.

```
1 // 8 bits per byte, 8/10 encoding
2 param qdrDataRate = ((10 * giga) / 8 ) * 0.8
3 param ibWidth = 4
4 param ibLatency = 1.0 * micro
5 param ibBW = qdrDataRate * ibWidth
6 interconnect infiniband {
7    latency   [ibLatency]
8    bandwidth [ibBW]
9 }
```

Listing 7. Aspen AMM Statements for Infiniband Interconnect

*c) **Node and Node Components**:* The remaining AMM concepts describe the components of a single compute node. Each node must have one or more sockets, with each socket containing at least one core, at least one type of memory (RAM or cache), and one link. Link, in this case, refers to intra-node communication (i.e. among sockets) rather than inter-node communication, which is described by the interconnect model. For Keeneland, the node topology is the Hewlett-Packard SL390, which has two CPUs and three GPUs per node. The corresponding Aspen code is given in Listing 8 where `westmere` and `fermi` refer to socket models.

A complete list of the node-level components is given below:

- **Socket:** A socket roughly represents a single processor, which may contain multiple cores, memories, and/or

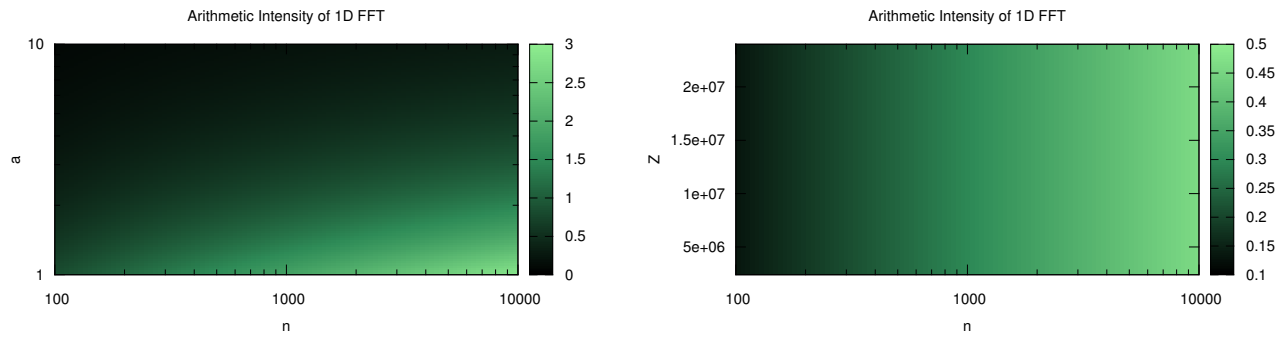

Fig. 3. Two plots showing how the arithmetic intensity (flop-to-byte ratio) of FFT varies with three parameters, problem size, n, and cache capacity, Z, and cache miss constant, a. Cache capacity, Z is varied from 2.4MB to 24MB (corresponding to the test platforms in Section III-C1) on a logarithmic scale.

caches. An example for the Intel Westmere CPU is given in Listing 9.

- **Core:** The concept for a single processing element, including its floating point computation ability and how that changes based on traits representing double precision and the presence of a fused multiply-add instruction.
- **Memory:** A model for RAM including capacity, latency, and bandwidth for multiple access patterns [22].
- **Cache:** The concept for private or shared (fast) memory attached to one or more cores. The complete set of concepts describing a Westmere socket is shown in Listing 9.
- **Link:** The link between sockets on a node, i.e. Quick Path Interconnect (QPI), HyperTransport, or PCIe (in the case of discrete GPUs).

```
node sl390 {
   [2] westmere sockets
   [3] fermi sockets }
socket westmere {
   [wmNumCores] westmere cores
   [1] banks of ddr3 memory
   [1] banks of shared wmCache cache
   linked with qpi }
```

Listing 8. Aspen AMM Statements for an HP SL390 Node and Westmere Socket

*d) **Limiting Assumptions**:* In an effort to maintain generality, the Aspen AMM currently makes some assumptions. It assumes a completely connected intra-node topology (i.e. each socket can directly communicate with every other socket on the node) that operates at the minimum link bandwidth of the pair of communicating sockets. For example, in the Keeneland model, when a Westmere and Fermi socket communicate, it occurs at eight gigabytes per second, the bandwidth of PCIe (instead of the faster QPI link). The second assumption involves resource contention, which is generally assumed to be linear. For example, if both Westmere sockets are involved in global communication, each is assumed to get fifty percent of the interconnect bandwidth. While it is known that both these assumptions represent strongly idealized behavior, NUMA and contention effects have proven difficult to accurately model in any analytical framework. Similarly, the memory hierarchy is modeled at only two levels—last level cache and physical memory. Again, this stems from striving for generality, as the detailed modeling of caches (including hierarchies, coherency protocols, prefetchers, etc.) has traditionally been prohibitively difficult for analytical models and is better studied with simulators. Future versions of Aspen may remove these limitations via additional language features.

```
// DDR3 Parameters
param ddr3Cap = 12 * giga
param ddr3MemClock = 1066 * mega
param ddr3Channels = 3
param ddr3BW  = ddr3MemClock * ddr3Channels * 8
param ddr3Lat = 10 * nano
// Westmere Details
core westmere {
   flops [wmBaseFlops]
   with sp(flops), dp(flops / 2), simd(flops *
       wmSIMD), fmad(flops) }
memory ddr3 {
   capacity  [ddr3Cap]
   latency   [ddr3Lat]
   bandwidth [ddr3BW] }
cache wmCache {
   capacity  [wmCacheCap]
   latency   [wmCacheLat]
   bandwidth [wmCacheBW] }
```

Listing 9. Aspen AMM Statements for DDR3 and Westmere Details

1) Example AMM Analysis: For any machine model, one can evaluate the upper bound on performance using the roofline model [24]. Given the arithmetic intensity of a kernel, the roofline model defines an upper limit on kernel performance, P_k, with the equation, $P_k = \min\{P_f, BA_i\}$ where P_f is the peak hardware floating point performance, B is peak bandwidth, and A_i is the arithmetic intensity (i.e. flop-to-byte ratio). For instance, the roofline models for each of Keeneland's processing elements, a dual-socket Intel Xeon X5660 CPU and an NVIDIA Tesla M2090 GPU are shown in Figure 5. Details for these processors are listed in Table III. Note that the M2090's register file is larger in capacity than the L2 cache and is therefore treated as the last level cache. While Figure 5 focuses on achievable flops, a similar construction could be made based on any resource (e.g. achievable bandwidth based on byte-to-flop ratios).

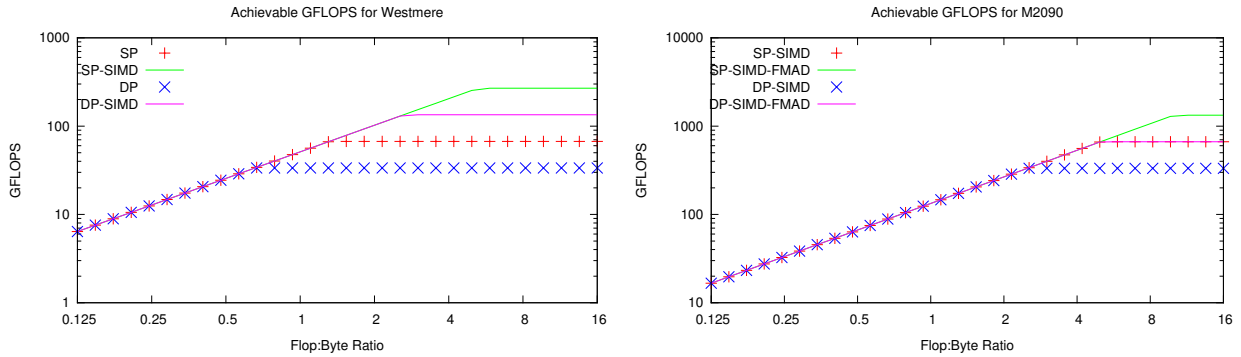

Fig. 5. Two Aspen-generated roofline charts showing the maximum achievable performance for a given type of floating point operation and arithmetic intensity for (a) Intel Xeon X5660 and (b) Tesla M2090 GPU. Note that M2090 SM width (32) is modeled as SIMD width in the style of Dongarra et al. [25], emphasizing the penalty for warp serialization.

TABLE III
HARDWARE PARAMETERS.

	Xeon X5660	Tesla M2090
Number of Cores	12	16 (512)
Peak GFLOPS SP/DP	268/134	1331/665
Memory Clock (Mhz)	1066	3700 (GDDR5 eff)
Memory Bandwidth (GB/s)	51.2	177
LL Cache Size (MB)	24	2.4

D. Combined Analyses

We now return to the analysis of 3D FFT, and provide some example results from Aspen using a combination of application and machine models.

In order to further characterize the relative importance of flops and memory traffic, we examine the predicted runtime for the `localFFT` kernel (by clause) on both types of processing elements in Keeneland. For reference, we also include the measured runtime using Intel MKL 12.1.2 for the CPU and CUFFT 4.1 for the GPU to provide insight into the tightness of the analytical bounds. Results are shown in Figure 6. Two observations are worth noting, including the high degree of observed overlap on both platforms and the comparable rate of increase (as `n` increases) in predicted cost for both clauses. These results indicate that any extrapolations to large scale shoud incorporate the requirements for both flops and memory operations.

Next, we characterize the runtime of the `exchange` kernel in a strong scaling scenario. Figure 7 shows the decrease in expected runtime as the number of processors increases (ranging from one thousand to one hundred thousand processors) for a fixed problem size of $n = 2^{16}$ for a Keeneland-like machine with varying interconnect topology. The model prediction indicates an order of magnitude difference in required `exchange` time for a hypercube and 3D torus or 3D mesh, demonstrating agreement with the projections from Gahvari and Gropp [16] and Czechowski et al [17].

Our final analysis of the 3D FFT Aspen model illustrates the differences between the slab and pencil decomposition by charting communication and computation costs in the same strong scaling scenario used for the topology comparison.

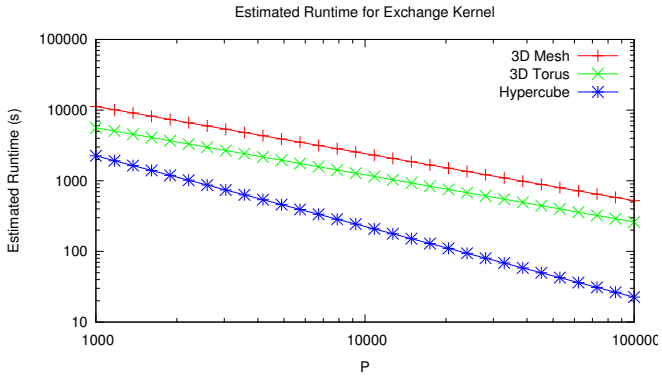

Fig. 7. Estimated runtime of the `exchange` kernel on variants of the Keeneland machine using a QDR Infiniband interconnect with a 3D Torus, 3D Mesh, or hypercube topology. This is a strong scaling scenario with a problem size of $n = 2^{16}$.

Recall from the model overview that the slab decomposition requires one less iteration of the `exchange` kernel, but cannot scale to beyond $p = n$ processors. The results from comparing the slab and pencil control flows on a machine with a variable number of Westmere processors are shown in Figure 8. For smaller processor counts, the slab-based decomposition will perform substantially faster due to reduced communication, which strongly dominates the time for computation. But, if more processors are available than the edge length of the volume, the increased parallelism in the pencil decomposition overcomes the overhead of an additional exchange.

IV. COMPOSABILITY

An additional benefit of a standardized, language-based approach to performance modeling is composability. Aspen application models support composability via an **`import`** statement [22]. In this section, we consider how the 3D FFT model might be imported into a model for molecular dynamics to describe the Particle Mesh Ewald (PME) method, a common approach for approximating long-range forces.

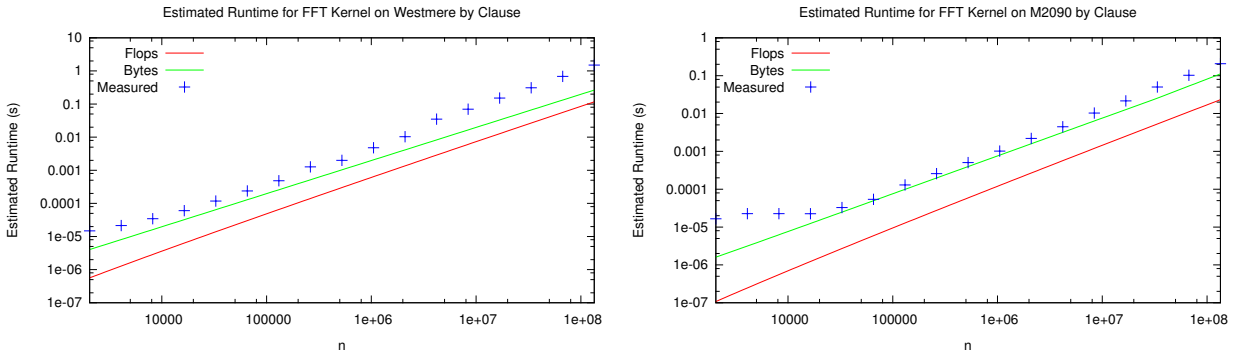

Fig. 6. Estimated runtime by clause on Xeon X5660 and Tesla M2090. The two implementations achieve a high degree of overlap between floating point and memory operations.

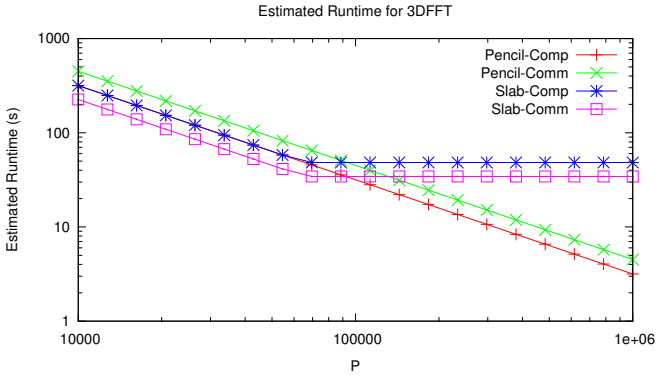

Fig. 8. Estimated runtime of computation and communication for the slab and pencil decompositions. While the slab is faster at smaller process counts due to reduced communication, the pencil decomposition is able to scale past 2^{16} processors (the length of the volume in one dimension). This is a strong scaling scenario with a problem size of $n = 2^{16}$ on Westmere processors with QDR IB in a hypercube topology.

A. MD Overview

The Aspen molecular dynamics model in this section is based on extending MiniMD [26] with long-range forces. It considers the general case of classical molecular dynamics, in which the equations of motion are integrated over time for a collection of interacting atoms spread uniformly in three dimensional space. The model uses a traditional spatial decomposition—the *box*, the region of space, is divided into *cells*, cubic regions with edge length equal to the cutoff distance plus some small factor, ϵ. For short-range forces, the magnitude of the force is inversely proportional to the distance between the two atoms, such that the contribution from distant atoms is negligible. Hence, short-range interactions can be ignored for atoms that are not in neighboring cells.

Our model uses the Lennard-Jones pairwise potential for short-range forces and the PME method for long-range forces. For a detailed description of the PME method, we refer the reader to the description from Darden et al. [27]. At a high level, the PME method involves assigning charge information from the atoms to a fixed mesh. Then, a 3D FFT is performed on the mesh to obtain a convolution in Fourier space. Next, an inverse 3D FFT brings the problem back to real space. Finally, the forces are calculated based on an interpolation from the mesh back to the atoms. This computation of long-range forces is often the bottleneck for molecular dynamics simulations due to high communication costs. Fortunately, the compute-bound short-range forces can be determined in parallel with the communication-bound long-range forces. This allows for a substantial degree of overlap, with the bottleneck varying based on the number of atoms and the resolution of the charge mesh. This parallelism can be represented in Aspen with the notation shown in Listing 10, which shows how the force computation fits into timestep integration.

```
1  // Import 3DFFT model, using charge mesh as the
       fft volume
2  import 3DFFT from "3D_FFT.aspen" with n =
       meshDim, fftVolume = chargeMesh
3  control main {
4    iterate [nTimeSteps] {
5      integrate(position, velocity)
6      integrate(velocity, accel)
7      exchange
8      buildNList
9      assignCharge
10     { iterate[2]{3DFFT.pencil}, ljForce }
11     interpolate
12     integrate(position, velocity)  }}
```

Listing 10. Aspen AMM Statements for MD Control Flow

A natural performance question arises from this parallelism—given a fixed box and charge mesh, how many atoms can be simulated before the force calculation phase becomes compute bound? To characterize this with Aspen, the number of flops in the `ljForce` kernel, shown in Listing 11, and the 3DFFT (3 rounds of `3DFFT.localFFT` kernel), is compared to the total message volume from two rounds of the `3DFFT.exchange` kernel. The resulting ratio is shown in Figure 9 for two fixed-size charge meshes (64^3 and 128^3) as the total number of atoms varies. At smaller atom counts, the ratio is dominated by the contribution of the FFT until the flops from the force kernel begin to increase dramatically (at around 100k atoms in the case of the 64^3 mesh). This is consistent with our expectation, since the number of flops in the FFT does not vary with the atom count.

```
kernel ljForce {
  exposes parallelism [nCellsX * nCellsY *
      nCellsZ]
  // Get the atoms' position
  requires loads [3 * atomsPerCell * wordSize]
      from position
  // Load neighbor IDs
  requires loads [avgNeighbors * wordSize] from
      neighborList
  // Load neighbors' positions
  requires loads [avgNeighbors * 3 * wordSize]
      from position
  // Distance calculation (11) + force
      calculation (21)
  requires flops [atomsPerCell * avgNeighbors *
      (11 + 21)] as simd, dp
  // Store forces
  requires stores [atomsPerCell * 3 * wordSize]
      to forces
  requires stores [atomsPerCell * wordSize] to
      energy
}
```

Listing 11. Aspen AMM Statements for MD LJ Force Kernel

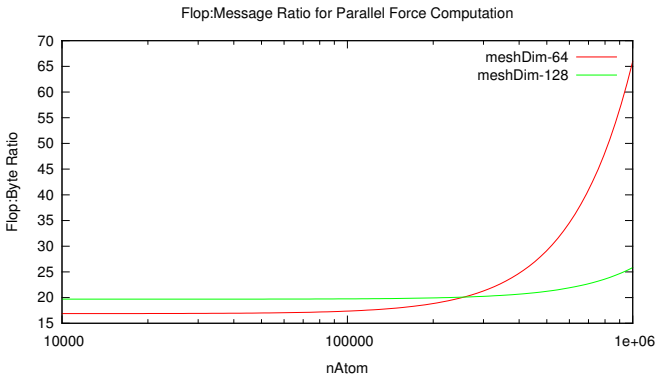

Fig. 9. Flop-to-message ratio for the parallel force computation. Contributions to flops come from the `ljForce` and `3DFFT.localFFT` kernels, while message volume is from the `3DFFT.exchange` kernel. Results are shown for two fixed-size charge meshes (64^3 and 128^3) on 32^3 cells with uniform atom distribution. For low atom counts, the ratio is dominated by the intensity of the 3D FFT (which is independent of atom count) until the contribution of the `ljForce` kernel dominates the ratio (at about 100k atoms for the 64^3 mesh).

While characterizing this ratio is important, we primarily use it to demonstrate the expressiveness of Aspen and emphasize the value of composability for performance models. Aspen facilitates this type of high-level, application-specific question and offers a significant productivity advantage compared to alternative methods (repurposing ad-hoc models, large number of simulation runs for parameter sweeps, etc.).

V. Conclusions

Using the three dimensional Fast Fourier Trasnform and a molecular dynamics application as examples, we have illustrated the advantages of using a composable, language-based approach to performance modeling. We have presented an overview of Aspen, a language designed to facilitate this type of analysis by providing increased detail and expressiveness compared to traditional analytical models while obviating the requirements for source code or an exact machine specification.

We have also demonstrated the use of a set of standard analysis tools built on Aspen including determining arithmetic intensity, predicting runtime, generating roofline charts, and evaluating performance as application-specific parameters vary. These tools have the potential to reduce the duplication of effort involved in ad hoc modeling approaches and force any limiting assumptions to be made more explicit.

In addition, we extended two recent performance models for 3D FFT [16], [17]. Our findings using Aspen agree with Gropp and Gahvari's observations on interconnect toplogy while adding valuable insight into the memory costs of the local FFTs and transpose operations. We also provide a characterization of the differences in the slab and pencil decompositions, exploring the tradeoff between communication costs and parallelism.

Finally, we showed how Aspen allows model composition, incorporating the 3D FFT model for use in the particle mesh Ewald (PME) method for calculating long-range forces in molecular dynamics simulations.

Acknowledgment

This research is sponsored by the Office of Advanced Scientific Computing Research in the U.S. Department of Energy and DARPA contract HR0011-10-9-0008. The paper has been authored by Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC under Contract DE-AC05-00OR22725 to the U.S. Government. Accordingly, the U.S. Government retains a non-exclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

REFERENCES

[1] J. Dongarra, P. Beckman *et al.*, "International exascale software roadmap," *International Journal of High Performance Computing Applications*, vol. 25, no. 1, 2011.

[2] P. Kogge, K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, J. Hiller, S. Karp, S. Keckler, D. Klein, R. Lucas, M. Richards, A. Scarpelli, S. Scott, A. Snavely, T. Sterling, R. S. Williams, and K. Yelick, "Exascale computing study: Technology challenges in achieving exascale systems," DARPA Information Processing Techniques Office, Tech. Rep., 2008.

[3] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken, "Logp: towards a realistic model of parallel computation," in *Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming*, ser. PPOPP '93, 1993, pp. 1–12.

[4] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman, "Loggp: incorporating long messages into the logp model," in *Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures*, ser. SPAA '95, 1995, pp. 95–105.

[5] L. G. Valiant, "A bridging model for parallel computation," *Communications of the ACM*, vol. 33, no. 8, pp. 103–111, 1990.

[6] M. Frigo, C. Leiserson, H. Prokop, and S. Ramachandran, "Cache-oblivious algorithms," in *40th Annual Symposium on Foundations of Computer Science*, 1999, pp. 285–297.

[7] T. Hoefler, W. Gropp, W. Kramer, and M. Snir, "Performance modeling for systematic performance tuning," in *State of the Practice Reports*, ser. SC '11, 2011, pp. 6:1–6:12.

[8] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of collective communication operations in mpich," *International Journal of High Performance Computing Applications*, vol. 19, no. 1, pp. 49–66, Spring 2005.

[9] R. Nishtala, G. Almasi, and C. Cascaval, "Performance without pain = productivity: data layout and collective communication in UPC," in *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '08, 2008, pp. 99–110.

[10] R. M. Karp, "A survey of parallel algorithms for shared-memory machines," University of California at Berkeley, Tech. Rep., 1988.

[11] A. F. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. CooperBalls, and B. Jacob, "The structural simulation toolkit," *SIGMETRICS Perform. Eval. Rev.*, vol. 38, no. 4, pp. 37–42, 2011.

[12] C. L. Janssen, H. Adalsteinsson, and J. P. Kenny, "Using simulation to design extremescale applications and architectures: programming model exploration," *SIGMETRICS Perform. Eval. Rev.*, vol. 38, no. 4, pp. 4–8, 2011.

[13] V. S. Adve, R. Bagrodia, J. C. Browne, E. Deelman, A. Dube, E. N. Houstis, J. R. Rice, R. Sakellariou, D. J. Sundaram-Stukel, P. J. Teller, and M. K. Vernon, "Poems: End-to-end performance design of large parallel adaptive computational systems," *IEEE Trans. Software Engineering*, vol. 26, no. 11, pp. 1027–1048, 2000.

[14] A. Snavely, L. Carrington, N. Wolter, J. Labarta, R. Badia, and A. Purkayastha, "A framework for performance modeling and prediction," in *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*. Baltimore, Maryland: IEEE Computer Society Press, 2002, pp. 1–17.

[15] D. J. Kerbyson, H. J. Alme, A. Hoisie, F. Petrini, H. J. Wasserman, and M. Gittings, "Predictive performance and scalability modeling of a large-scale application," in *Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)*. Denver, Colorado: ACM, 2001, pp. 37–37.

[16] H. Gahvari and W. Gropp, "An introductory exascale feasibility study for FFTs and multigrid," in *Proceedings of the IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, April 2010.

[17] K. Czechowski, C. Battaglino, C. McClanahan, K. Iyer, P. Yeung, and R. Vuduc, "On the communication complexity of 3d fft and its implications for exascale," in *Proceedings of the 2012 International Conference on Supercomputing (ICS 2012)*, June 2012.

[18] J. Doi and Y. Negishi, "Overlapping methods of all-to-all communication and FFT algorithms for torus-connected massively parallel supercomputers," in *High Performance Computing, Networking, Storage and Analysis (SC), 2010 International Conference for*, November 2010, pp. 1 –9.

[19] C. Young, J. A. Bank, R. O. Dror, J. P. Grossman, J. K. Salmon, and D. E. Shaw, "A 32x32x32, spatially distributed 3d fft in four microseconds on anton," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, ser. SC '09, 2009, pp. 23:1–23:11.

[20] A. Nukada and S. Matsuoka, "Auto-tuning 3D FFT library for CUDA GPUs," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, ser. SC '09, 2009, pp. 30:1–30:10.

[21] D. Bailey, T. Harris, W. Saphir, R. V. D. Wijngaart, A. Woo, and M. Yarrow, "The NAS parallel benchmarks 2.0," *NASA Technical Report NAS-95-020*, 1995.

[22] K. Spafford and J. Vetter, "Performance modeling with aspen," *Oak Ridge National Laboratory Technical Report 2012-01. Available http://kylespafford.com/aspen.html*, 2012.

[23] J. Vetter, R. Glassbrook, J. Dongarra, K. Schwan, B. Loftis, S. McNally, J. Meredith, J. Rogers, P. Roth, K. Spafford, and S. Yalamanchili, "Keeneland: Bringing heterogeneous GPU computing to the computational science community," *IEEE Computing in Science and Engineering*, vol. 13, no. 5, 2011.

[24] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, Apr. 2009.

[25] R. Nath, S. Tomov, and J. Dongarra, "An improved magma GEMM for fermi graphics processing units," *International Journal of High Performance Computing Applications*, vol. 24, no. 4, pp. 511–515, 2010.

[26] M. H. et al., "Improving performance via mini-applications," *Sandia National Laboratories Technical Report SAND2009-5574*, 2009.

[27] T. Darden, D. York, and L. Pedersen, "Particle mesh ewald: An $n \log(n)$ method for ewald sums in large systems," *The Journal of Chemical Physics*, vol. 98, no. 12, pp. 10 089–10 092, 1993.