Hindawi Publishing Corporation
Scientific Programming
Volume 2015, Article ID 157305, 10 pages
http://dx.doi.org/10.1155/2015/157305

Research Article

Automated Design Space Exploration with Aspen

Kyle L. Spafford and Jeffrey S. Vetter

Oak Ridge National Laboratory, One Bethel Valley Road, Building 5100, MS-6173 Oak Ridge, TN 37831-6173, USA

Correspondence should be addressed to Jeffrey S. Vetter; vetter@computer.org

Received 22 April 2014; Accepted 2 November 2014

Academic Editor: Roman Wyrzykowski

Architects and applications scientists often use performance models to explore a multidimensional design space of architectural characteristics, algorithm designs, and application parameters. With traditional performance modeling tools, these explorations forced users to first develop a performance model and then repeatedly evaluate and analyze the model manually. These manual investigations proved laborious and error prone. More importantly, the complexity of this traditional process often forced users to simplify their investigations. To address this challenge of design space exploration, we extend our Aspen (Abstract Scalable Performance Engineering Notation) language with three new language constructs: user-defined resources, parameter ranges, and a collection of costs in the abstract machine model. Then, we use these constructs to enable automated design space exploration via a nonlinear optimization solver. We show how four interesting classes of design space exploration scenarios can be derived from Aspen models and formulated as pure nonlinear programs. The analysis tools are demonstrated using examples based on Aspen models for a three-dimensional Fast Fourier Transform, the CoMD molecular dynamics proxy application, and the DARPA Streaming Sensor Challenge Problem. Our results show that this approach can compose and solve arbitrary performance modeling questions quickly and rigorously when compared to the traditional manual approach.

1. Introduction

The design of next generation Exascale computer architectures as well as their future applications is complex, uncertain, and intertwined. Not surprisingly, modeling and simulation play an important role during these early design stages as neither the architectures nor the applications yet exist in any substantive form. Consequently, relevant performance models need to describe a complex, multidimensional design space of algorithms, application parameters, and architectural characteristics. Traditional performance modeling tools made this process difficult and resulted in a tendency to use simpler, less accurate models.

In our earlier work, we designed Aspen (Abstract Scalable Performance Engineering Notation) [1], a domain specific language for structured analytical performance modeling, to allow scientists to construct, evaluate, verify, compose, and share models of their applications. Aspen specifies a formal language and methodology that allows modelers to quickly generate representations of their applications as well as abstract machine models. In addition, Aspen includes a suite of analysis tools that consume these models to produce a variety of estimates for computation, communication, data structure sizes, algorithm characteristics, and bounds on expected runtime. Aspen can generate all of these estimates without application source code or low-level architectural information like Register Transfer Level (RTL). This ability to cope with high levels of uncertainty distinguishes Aspen from simulators, emulators, and other trace-driven approaches.

In fact, Aspen (and analytical modeling in general) is particularly useful at an early time horizon in the codesign process where the space of possible application parameters, algorithms, and architectures is too large to search with computationally intensive methods (e.g., cycle-accurate simulation) [2]. With this much uncertainty, application developers tend to identify important ranges of application parameters, rather than discrete values. Similarly, hardware architects may have identified a range of possible computational capabilities, but the machine characteristics have not been finalized. For example, feasible clock ranges may be dictated by the feature size and known well in advance of fabrication. Finding optima within these ranges transforms a typical performance modeling projection into an optimization problem.

1.1. Key Contributions. To address this challenge of design space exploration, we have extended our Aspen language and environment with expressive semantics for characterizing flexible design spaces rather than single models. Specifically, we add three new language constructs to Aspen: user-defined resources, parameter ranges, and a collection of costs in the abstract machine model. Then, we use these constructs to enable automated design space exploration via a nonlinear optimization solver. The solver uses these ranges (along with other constraints) to evaluate the Aspen performance models and evaluate a user-defined objective function for each point in the design space. As we will show, this automated process can allow thousands of model evaluations quickly and with minor regard to the performance model complexity.

The key contributions of this paper are as follows:

(1) a description of Aspen's syntax and semantics for specifying resources, parameter ranges, and costs in the abstract machine model;

(2) a formal problem description for four types of optimization problems derived from Aspen models;

(3) a description of new Aspen analysis tools which consume Aspen models and explore the design space with a standard nonlinear optimization solver;

(4) a demonstration of these new capabilities on existing Aspen models for 3DFFT, CoMD, and the Streaming Sensor Challenge Problem [3].

1.2. Related Work. In the space of analytical models, Aspen's approach to the abstract machine model is conceptually in between pure analytical models and semiempirical power-performance models based on direct measurement. Examples of the former include BSP [4] and Log P variants [5, 6] that focus strictly on algorithmic bounds. Examples of the latter include models based on performance counters or measurements [7–12] including proposed counters such as the leading loads counter [13]. Aspen is distinguished from these works in that it is capable of modeling machines and applications in more detail than the pure analytical models while obviating the requirement of the semiempirical approaches for an instrumented execution environment. Other related approaches are trace-driven and use linear programming for power-performance exploration, especially for searching the configuration space of dynamic voltage and frequency scaling [14, 15] or making decisions under explicit hardware power bounds [16].

On the application side, our goals for the use of Aspen and the 3DFFT model are directly related to the Exascale feasibility and projection studies of Gahvari and Gropp [17], Bhatele et al. [18], and Czechowski et al. [19].

In terms of design space exploration itself, an automated approach is a well-studied topic. Hardware-focused studies are also common, although they typically focus on reconfigurable architectures [20–22], particularly in well-constrained compiler-based planning or system on a chip (SoC) designs [23–26].

Several works focus on the theoretical aspects of exploring design spaces. Peixoto and Jacome examine metrics for the high-level design of such systems [27]. There are also works focusing on the abstractions [28] and algorithms for the search [29], environments where source code is available and modifiable [30], and specialized approaches for multilevel memory hierarchies [30]. In general, these works have similar goals and overall function to DSE in Aspen, but they consider very different machine models (usually with much more certainty and detail than the Aspen AMM).

2. Aspen Overview

While a more detailed description of Aspen has been published elsewhere [1], we briefly provide an overview and illustrate its use on an example model for a 1D Fast Fourier Transform (FFT). Aspen's domain specific language (DSL) approach to analytical performance modeling provides several advantages. For instance, Aspen's `control` construct helps to fully capture control flow and preserves more algorithmic information than traditional frameworks like BSP [4] and Log P variants [5, 6]. Similarly, the abstract machine model is more expressive than frameworks that reduce machine specifications to a small set of parameters.

The formal language specification forces scientists to construct models that can be syntactically checked and consumed by analysis tools; this formal specification also facilitates collaboration between domain experts and computer scientists. Aspen has also been defined to include the concept of modularity, so that it is easy to compose, reuse, and extend performance models.

Furthermore, this specification allows scientists to include application specific parameters in their model definitions, which would otherwise be difficult to infer. With this feature, Aspen can help answering application-specific questions such as how does parallelism vary with the number of atoms? And, this type of approach also allows inverse questions to be asked, such as, given a machine, what application problem can be solved within the system constraints?

Aspen is complementary to other performance prediction techniques including simulation [31, 32], emulation, or measurement on early hardware prototypes. Compared to these techniques, Aspen's analytical model is machine-independent, has fewer prerequisites (e.g., machine descriptions, source code), and decreased computational requirements. This positions Aspen as an especially useful tool during the early phases in the modeling lifecycle, with continuing use as a high-level tool to guide detailed studies with simulators. Hence, the primary goal of Aspen is to facilitate algorithmic and architectural exploration early and often.

2.1. Example: FFT. The FFT is a common scientific kernel and plays an important role in the image formation phase of SSCP [3], explored further in Section 5. Fortunately, FFT is also a well-studied algorithm, and tight bounds on the number of operations in an FFT are known.

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

Listing 1: Aspen kernel for 1D FFT.

```
param n = 1 .. 100 // Basic Syntax
param n = 100 in 10 .. 1000 // Default Value
```

Listing 2: Syntax for an Aspen range.

For an n-element Cooley-Tukey style 1D FFT [33], the required number of floating point operations is bounded by $\mathcal{O}(5n\log_2 n)$, with some implementations requiring only 80% of this upper bound [34]. The number of cache misses has also been bounded for any FFT in the I/O complexity literature (on any two-level memory hierarchy which meets the tall cache assumption [35]) as $\Theta(1 + (n/L)(1 + \log_Z n))$, where L is the cache line size in words and Z is the cache capacity in words. For sufficiently large n, the number of cache misses, N_m, approaches $N_m = An\max(\log_Z n, 1)$, where A is a constant [19, 35] which translates the upper bound to an explicit count. Using the same variable names, these bounds roughly translate to two Aspen kernel clauses, as shown in Listing 1.

The listing also highlights the use of Aspen traits to add semantic information to specialize the flops, indicating that they are double precision, complex, and amenable to execution on SIMD FP units. The trait on the second clause specifies that the memory traffic in this kernel is from the `fftVolume` data structure.

The other variable, a, is a constant that arises from the nature of characterizing requirements by asymptotic bounds (e.g., $\mathcal{O}()$) [35]. Due to the complexity in modeling the memory hierarchy (e.g., from multilevel cache hierarchies, replacement policies) 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 [36].

3. Modeling Methodology

In order to facilitate the evaluation of optimization problems, Aspen has been extended with three new language constructs to increase expressiveness.

3.1. User-Defined Resources. Prior work [1] with Aspen constrained modelers to a small set of predefined quantities of interest: flops, loads, stores, and messages. Since then, requests for modeling more exotic resources like system calls, allocation/deallocation, and more detailed modeling of system data paths (PCIe, QPI) have necessitated a more flexible system.

The first addition to Aspen is the ability for custom resources to be defined at arbitrary points in the abstract machine model (AMM) hierarchy. For instance, integer operations can be defined at the `core` level and access to a center-wide, shared filesystem could be defined at the `machine` level. Resources may also define custom traits with optional arguments. All new definitions, however, must provide an expression for how the resource maps to time and how the traits commutatively modify or replace the base expression (the mapping when no traits are present). An example of the new syntax is shown in Listing 3. Note that the new `conflict` statement describes the sets of resources that cannot overlap.

Furthermore, the AMM's assumptions of a completely connected socket topology and linear contention [1] are unchanged and apply equally to user-defined resources.

3.2. Ranges. The next construct is the range, illustrated in Listing 2. The range or interval is a familiar concept to programmers, has implementations in most modern languages, and is fairly easy to express and reason about.

More precisely, a *range* in Aspen is a closed, inclusive, connected, and optimal set of real numbers, S. A range that is *closed and inclusive* indicates that the interval contains lower and upper bounds a and b such that $a \le x \le b$, $\forall x, a, b \in S$, and $\forall S \in \mathbb{R}$. *Optimal*, in this case, means that range should be as narrow as possible. Aspen also allows for the specification of an explicit default value. This default value provides a convenient way for modelers to encode the "common case." When left unspecified, the lower bound is

```
core snbCore {

  resource flops(number) [number / snbIssue ]
    // Traits
    with dp [base * 2],
      // Optional Trait Argument
      simd(width) [base / min(width, snbSIMDWidth)],
      fmad [base / 2]
    // Per-resource, per-core dynamic power
    dynamic power [ (tdp - snbIdlePower) / snbNumCores ]

  resource intops(number) [ number / snbIssue ]
    dynamic power [ (intMaxPower - snbIdlePower) / snbNumCores ]

  resource aesops(number) [ number / snbIssue ]
    dynamic power [ (aesMaxPower - snbIdlePower) / snbNumCores ]

  conflict (flops, intops, aesops)

  // Shared static power cost
  static power [ snbIdlePower ]
}
```

Listing 3: Aspen core model with static and dynamic costs.

used (by convention) in single analyses which do not consider ranges.

3.3. *Including Costs in the Abstract Machine Model.*

The second extension to Aspen includes the incorporation of several new types of costs into the abstract machine model: rack space, die area, static power, dynamic power, and component price. Each type of cost has rules for which components of the AMM hierarchy are applicable. However, all of these costs are optional. The only required cost is the specification of the time it takes to process a given resource.

Available rack space, the simplest cost, is specified at the machine level and associated costs are defined per node in standard units.

Total available die area is provided at the socket level and area costs are listed explicitly for all core, cache, and memory components. This allows, for instance, exploration of the tradeoff between die area spent on cache and the number of cores.

Static power costs are specified by providing each component of the AMM hierarchy with an idle wattage. Dynamic power is similarly specified at each point in the hierarchy, but it is also split by resource. That is, for a given component, performing different operations may result in different dynamic power requirements. A trivial example of this difference is an AMM where the cost of a floating point operation exceeds the cost of an integer operation.

Consider the example shown in Listing 3, where an AMM model for an Intel Sandy Bridge processor distinguishes between the power costs of a standard integer operation and the execution of the new advanced encryption instruction set. While this example may seem somewhat contrived with existing hardware, its inclusion as a feature is important in future-proofing Aspen against the general trend towards more specialized instructions and fixed-function units that may vary widely in energy consumption.

These power costs also allow specifying constraints for maximum instantaneous power draw (i.e., highest wattage) and total energy consumption. Maximum power draw for an application is computed as the sum of all AMM component static costs and the largest of the sums of dynamic costs for each kernel:

$$W_{\max} = \overbrace{\sum_{i}^{\mathbb{M}} W_{i_{\text{idle}}}}^{\text{static}} + \overbrace{\max_{j}^{\mathbb{K}} \left(\sum_{k}^{\mathbb{R}_j} W_{k_{\text{dyn}}} \right)}^{\text{dynamic}}, \qquad (1)$$

where $\mathbb{M}$ is the set of all components in the AMM, W_i is the idle power draw of component i, $\mathbb{K}$ is the set of all kernels in the application model, $\mathbb{R}_j$ is the set of all resources required by kernel j, and $W_{k_{\text{dyn}}}$ is the dynamic power cost of resource k. In the absence of an application model, the maximum power draw is given by upper bound as the sum of static costs and the dynamic costs of all nonconflicting resources.

Similar to Aspen's other assumptions, these power calculations represent a simplified model which neglects several physical factors including cooling costs and transitions between component idle/peak states.

The Aspen tools already include the capability to produce bounds on predicted runtime by kernel clause [1], and the total energy cost of an application model is hence computed by the following:

$$C_{\text{energy}} = \overbrace{\left(W_{\text{idle}} \times r_{\text{total}}\right)}^{\text{static}} + \overbrace{\sum_{i}^{\mathbb{K}} \left(\text{calls}_i * \sum_{j}^{\mathbb{C}} \left(r_j \times W_j \right) \right)}^{\text{dynamic}}, \quad (2)$$

where W_{idle} is the total system idle power, r_{total} is the total runtime, $\mathbb{K}$ is the set of all application kernels, calls_i indicates the number of calls to kernel i, $\mathbb{C}$ is the set of all clauses in kernel i, r_j is the runtime bound on clause j, and W_j is the dynamic power cost of the resource associated with clause j.

4. Nonlinear Optimization Solver

Using these new ranges and costs, a variety of optimization problems can be derived from Aspen models. These optimization problems have the following form.

(i) $f(\vec{\mathbf{x}})$ is an objective function which must be maximized or minimized such as runtime, energy consumed, or problem size.

(ii) $\vec{\mathbf{x}} = x_1, x_2, \ldots, x_n$ is a vector of decision variables with upper and lower bounds, sometimes called free variables. These bounds are typically derived from a range construct. Some examples include the number of nodes, problem sizes, and clock frequencies. The number of decision variables is known as the *dimensionality* of the problem.

(iii) $h_i(\vec{\mathbf{x}}) = 0, i \in 1, \ldots, p$, is a set of p equality constraints, which are arbitrary functions of the decision variables that must be equal to zero.

(iv) $g_i(\vec{\mathbf{x}}) \leq 0, i \in 1, \ldots, m$, is a set of m inequality constraints, which are functions on the decision variables that must be less than or equal to zero.

The difficulty of these optimization problems depends on several factors. In the best case, the constraint functions and the objective function are linear, and all of the decision variables are reals. This results in a traditional linear programming problem which can be trivially solved given the relatively low number of decision variables derived from an Aspen model.

If, however, some decision variables are integers, the problem is a mixed integer-linear program and is NP-complete. Similarly, difficulty is increased if the objective function or any of the constraint functions is nonlinear (i.e., nonlinear programming). And, if the objective function is not differentiable, a large class of efficient gradient-based methods cannot be used.

The current set of Aspen optimization tools relaxes all integer variables such that the typically generated optimization problem is a completely bounded, pure nonlinear program where the objective function may not be differentiable. An example of a relaxed integer variable might be the number of nodes (which, in practice, is easy to round to the nearest integer after optimization).

Since the objective or constraints may be complex, derived expressions (e.g., projected runtime, energy costs, and operation counts), these functions may be nonlinear and nondifferentiable. Hence, all optimization problems are solved using a gradient-free improved stochastic ranking evolution strategy (ISRES) [37] algorithm from the NLopt package [38].

Because no feasible point may be known *a priori*, these are considered global (as opposed to local) optimization problems. Establishing the criteria for termination is not always straightforward. However, due to the relatively low dimensionality (ISRES scales to thousands of variables) of Aspen-generated problems, we select NLopt's time-based stopping criterion with a threshold of a few seconds.

An interesting facet of this approach is that a user can constrain any combination of the parameters, leaving the objective function to include the remaining parameters. For example, in the Machine Planner scenario, the user defines the application model and constraints, general parameters of time to solution or power, and they use the design space exploration to search for the best combination of machine parameters. In another example, the Problem Size Planner, the user defines the machine parameters, constrains the same general parameters of time to solution or power, and then maximizes the application input problem that can be solved with that configuration.

5. Design Space Exploration

Combined with the existing analysis tools, the new range and cost constructs enable the formulation of a vast number of optimization problems for design space exploration. Combinations of the number and type of Aspen models involved, the portions of those models that are fixed or free variables, the goal (maximization or minimization), objective function, and additional constraints rapidly grow out of control. To constrain this otherwise unwieldy variety, the tool interface for design space exploration is centered on four common scenarios, summarized in Table 1.

5.1. Implementation Overview. The implementation of the tools, however, enables roughly the same workflow for each of the four scenario types, as depicted in the process diagram in Figure 1. This workflow has two main phases, problem formulation and optimization.

First, depending on the scenario, one of the Aspen optimization tools is run. This tool consumes one or more Aspen model files as input and collects the relevant ranges from the model into the vector of decision variables, $\vec{\mathbf{x}}$. Additional constraints such as time, energy, space, capacity, or price are specified via command line option. Also specified via the command line are nonstandard objective functions, which may include one or more parameters, derived capabilities, or weighted combinations of parameters and capabilities.

Based on these inputs, the Aspen optimization tools generate a single C++ code file that drives NLopt's standard API. This generated code preserves the semantics of the original Aspen models such that variable names are consistent and

Table 1: Comparison of Aspen scenarios for design space exploration.

Name	Application model	AMM	Constraints	Objectives
Parameter tuner	Single, free	None	Param ranges	Minimize op counts, data sizes
Problem size planner	Single, free	Single, fixed	Param ranges, energy budget, and time limit	Maximize App params
Machine planner	Single, fixed	Single, free	Param ranges, energy budget, and time limit	Minimize AMM params
AMM architect	None	Multiple, free	Param ranges, power budget, and price	Maximize capability, minimize costs

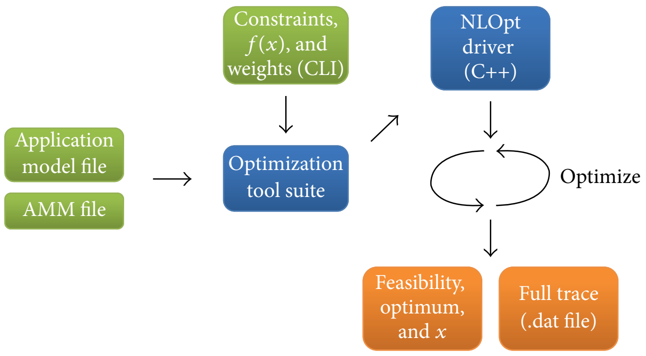

Figure 1: Process diagram of the design exploration workflow. CLI indicates inputs specified via command line options to the four problem formulation tools. $f(\vec{\mathbf{x}})$ refers to the objective function and $\vec{\mathbf{x}}$ is the vector of decision variables.

the code is amenable to inspection and modification for special use cases.

In the optimization phase, the generated C++ code is compiled and run. This code prints the value of the objective function at the optimum as well as the values of all of the decision variables. Or, in the case of unfeasible problems, it indicates that no optimum was found. It optionally generates a trace file that contains all the values of $\vec{\mathbf{x}}$ and $f(\vec{\mathbf{x}})$ for each evaluation of the objective function for postprocessing and visualization.

5.2. DSE Scenarios. In the following sections, we provide an overview of each scenario (and Aspen tool) in more detail and provide some pertinent example analyses. Note that, for these examples, we use relatively straightforward objective functions and only a handful of decision variables, but Aspen can handle problems of arbitrary complexity and dimensionality (given a reasonable solution timeframe).

5.2.1. Parameter Tuner. The first optimization tool addresses application models with tunable parameters that have a significant impact on performance. While this is generally applicable to application-specific parameters, our motivating use case is a tiling factor. This type of factor (equivalent to blocking and chunking factors for our purposes) is quite common due to data-parallel decomposition and cache-blocking techniques.

As a motivating example, we consider the DARPA UHPC Streaming Sensor Challenge Problem (SSCP) [3]. In this challenge problem, dynamic sensor data are converted to an image and pushed through a multistep, data-parallel analysis pipeline. The image is split into tiles according to a tiling factor, `tf`, which specifies how many tiles to use in each dimension. The two primary phases of the pipeline are digital spotlighting and backprojection.

The `tf` factor has a particularly interesting effect on total floating point operation count. Digital spotlighting kernels tend to require less work with smaller tiling factors (largely due to a requirement for fewer FFTs) while backprojection is more efficient at larger tiling factors. Choosing poor `tf` results in a potential for substantial unnecessary work (and, consequently, poor performance and low energy efficiency).

In order to characterize this tradeoff with the Paramater Tuner, the Aspen model for SSCP encodes the tiling factor as a range:

```
param tf = 32 in 16 .. 64
```

Combined with a command line argument for the resource of interest (e.g., flops, memory capacity), the Parameter Tuner generates a minimization problem with one bounded decision variable (`tf`) and an objective function that computes the total number of that resource required by the kernels in SSCP.

Prior to this work, Aspen had the capability to plot resource requirements in terms of one or two variables [1]. Figure 2 depicts a standard resource plot annotated with a tick for the first 250 points where the objective function was evaluated, with the minimum found at $7.009e + 13$ total flops at a `tf` of 34.

We note two observations concerning Figure 2. First, each objective function evaluation is consistent with the analytically computed total flop count, indicating consistency across different Aspen tools. Second, the linear relaxation of `tf` (an integer) introduces some minor inefficiency, as the objective function is evaluated multiple times for equivalent values.

5.2.2. Problem Size Planner. The second optimization tool is focused on the exploration of what problems are feasible to solve on a machine given a set of constraints. These

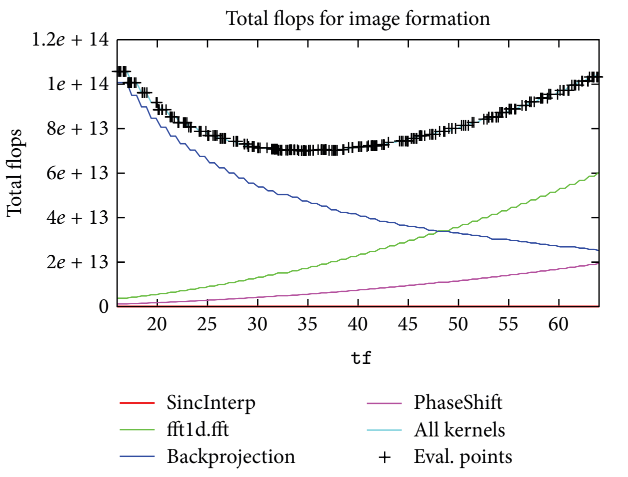

FIGURE 2: A characterization of the total number of flops required for SSCP image formation by kernel. Each black tick indicates a single evaluation of the objective function by the nonlinear optimizer as it is executed.

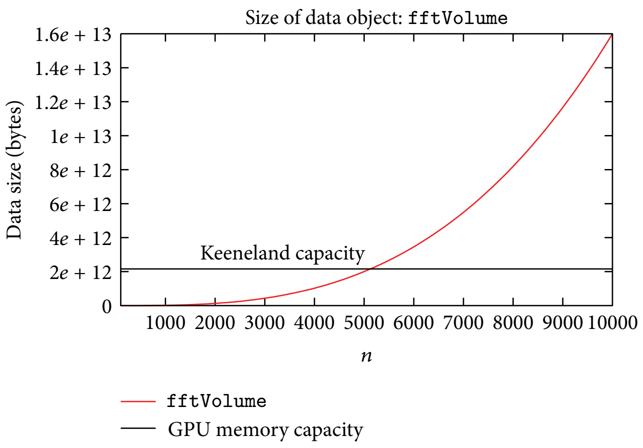

FIGURE 3: This chart shows the growth in the `fftVolume` data structure as a function of n relative to the memory capacity constraint (GPU physical memory on Keeneland).

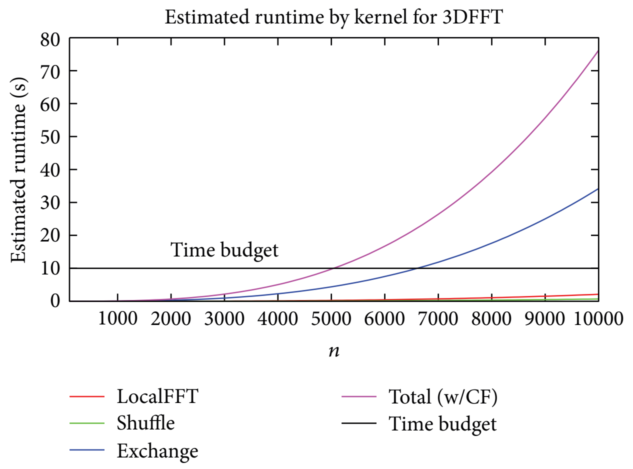

FIGURE 4: This chart shows the increase in predicted runtime as a function of the dimension of the volume n and its relation to the specified constraint of five seconds. In addition to overall runtime (CF = control flow), runtime per invocation of each kernel is also shown.

constraints can consist of time, power, energy, and/or capacity limits. In addition to traditional runtime and allocation planning, searching this design space can help provide an application-specific perspective on the benefits of obtaining new hardware by comparing results across different machine models.

To motivate this tool, we consider a model for a 3DFFT [1] and want to answer the question of what is the largest 3DFFT we can solve such that

(i) the `fftVolume` data structure fits into the aggregate memory of the GPUs on the NSF Keeneland system [39];

(ii) it has an estimated runtime of less than ten seconds;

(iii) it has an estimated total energy consumption of no more than five megajoules.

Our optimization problem, then, is a maximization problem of dimensionality one where the single decision variable (and objective function) is n, the dimension of the 3DFFT volume. Furthermore, each of the three requirement statements above corresponds to a single inequality constraint. Figures 3, 4, and 5 show how the requirements for the 3DFFT scale with n.

This energy calculation is based on a simple power model where the dynamic power requirement of the GPU is the manufacturer's stated thermal design point (250 W) when performing floating point operations or memory transfers, and the static/idle power is that measured using the NVIDIA system management interface (30 W). Transitions between states are assumed to be instantaneous and without cost. While simple, this model approximates the race-to-idle behavior. In future work, this model could be improved by measuring power draw for each resource using a synthetic benchmark (e.g., only flops, only loads/stores, and only MPI messages).

5.2.3. Machine Planner.

5.2.3. Machine Planner. The third interface for formulating optimization problems with Aspen models is the Machine Planner. In contrast to the first two tools, the Machine Planner fixes the application model and focuses on identifying application-specific targets for machine capabilities. In other words, it explores what minimum level of performance the machine must attain to complete a workload within a set amount of time, energy, and/or other constraints.

This scenario is typically a minimization problem over parameters in the abstract machine model. As an illustrative example, we consider a model for the CoMD molecular dynamics proxy application and the Keeneland AMM. Specifically, we want to find the minimum clock frequencies for a Fermi GPU's cores and memory that are required to complete a thousand iterations of CoMD's embedded atom

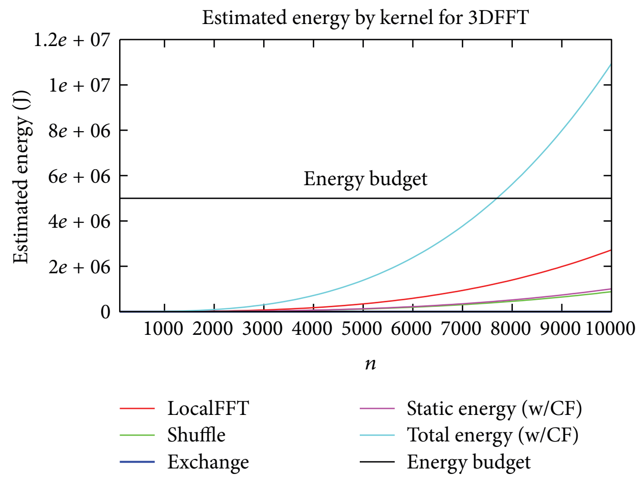

Figure 5: This chart shows the relationship between n and overall energy consumption, relative to the constraint of five megajoules. In addition to overall and per-kernel dynamic consumption, static costs for Keeneland are also shown.

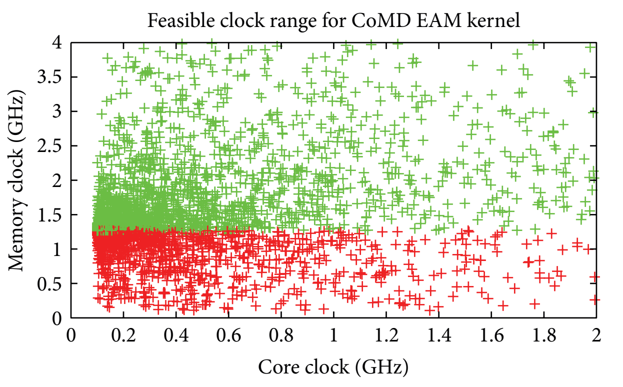

Figure 6: This figure shows the feasible core and memory clock ranges for computing one thousand iterations of the CoMD EAM force kernel on a Fermi GPU within one second. Green markers indicate evaluation points that satisfy the constraints and red markers indicate infeasible clock settings.

method (EAM) force kernel for just over a million atoms (1048576) in one second.

The parameter hierarchy in Listing 4 shows how the effective memory bandwidth is computed as a derived parameter from the clock rate that incorporates aspects of the GDDR5 architecture including the interface width and the measured overheads associated with using ECC. GDDR5's quad pumping, transferring a word on the rising and falling edge of two clocks, is accounted for within the `gddr5Clock` parameter, although this could be broken out into a separate parameter. Furthermore, `eccPenalty` accounts for overheads and `sustained` is based on measurements from the SHOC benchmark suite [40] that accounts for the difference between maximum sustained and peak bandwidth.

Figure 6 shows the feasible range for both clocks and provides two insights. First, the EAM kernel is strongly memory-bound and is feasible at the lowest point in the core clock range. And second, the increased concentration of evaluation points toward the computed optimum ($1e+08, 1.27e+09$) shows NLopt converging on the solution.

5.2.4. AMM Architect. The fourth tool is the AMM Architect which focuses on application-independent analyses. It primarily facilitates solving two types of problems—capacity planning under constraints and optimizing within a bounded projection for future performance targets (similar to the projections from the Echelon project [41] and DARPA Exascale Study [42]). These scenarios are typically maximization problems, where the objective function is some machine capability like peak flops, bandwidth, or capacity.

As an example calculation, we consider a sample problem which maximizes the floating-point capability for a Keeneland-like architecture under the following constraints:

(1) space and power budget of 42 U (one rack) and 18 KW, respectively,

(2) minimum double precision FP capability of 50TF,

(3) minimum aggregate FP capability to memory bandwidth ratio of 3 : 1 flops : bytes.

The decision variables correspond to all the ranges in the machine model including the number of nodes, number of sockets (1–4) and GPUs per node (1–8), and all the clock frequencies (CPU core, DDR3 memory, GPU core, and GDDR5 memory).

After running the AMM Architect, we discover that this problem has no feasible solution. This problem was chosen to highlight one of the limitations of the optimization-based approach: when there is no feasible solution for a multiconstraint problem, determining why the solution is not feasible or how "close" to feasibility the best point is requires nontrivial postprocessing. In practice, however, this can usually be overcome by iteratively relaxing the constraints.

6. Conclusions

Most scientists that use performance modeling are seeking to understand systems or optimize specific configurations, rather than generating a single forward performance projection. Likewise, many of the performance modeling scenarios facilitated by Aspen are concerned with the exploration of a multidimensional design space. The addition of user-defined resources, parameter ranges, and AMM costs substantially increases Aspen's flexibility and helps facilitate more complex modeling workflows. The ability to specify static and dynamic energy costs is especially important for models that describe extreme-scale or energy-constrained environments.

With these new costs, the vast array of potential optimization problems can be unwieldy. Aspen attempts to streamline problem formulation by constraining the interface to four specific scenarios. While these tools do not address all potential problems of interest (and we anticipate that expert users will modify these tools and generate their own scenarios), they do automate the process for common performance modeling tasks.

```
(1) // Effective clock (includes quad-pump)
(2) param gddr5Clock = 3.7 * giga in 100 * mega .. 4 * giga
(3) param gddr5Width = 48
(4) param eccPenalty = 0.88
(5) param sustained = 0.852// measured
(6) param gddr5BW = gddr5Clock * gddr5Width * eccPenalty * sustained
```

Listing 4: Aspen parameters for GDDR5 bandwidth.

6.1. Future Work. In the course of this work, we have identified two major challenges that require further study. First, complex models, especially those with high dimensionality, will require additional techniques to effectively visualize the design space. While some visualizations geared towards multidimensional data exist (e.g., parallel coordinates), visualizing ten or more dimensions is a common problem in scientific visualization. The current optimization tools write out a data file that contains each evaluation of the objective function, and the search space can be visualized a few dimensions per plot.

Another challenge for generating optimization problems involves specifying weights for complex objective functions. Directly adding weights to Aspen parameter definitions proved cumbersome and failed to address objective functions with nonparameter, derived quantities. Instead, the current tools require explicit command-line options for these weights.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

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 nonexclusive, 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] K. L. Spafford and J. S. Vetter, "Aspen: a domain specific language for performance modeling," in *Proceedings of the 24th International Conference for High Performance Computing, Networking, Storage and Analysis (SC '12)*, pp. 1–11, IEEE, Salt Lake City, Utah, USA, November 2012.

[2] J. J. Yi, L. Eeckhout, D. J. Lilja, B. Calder, L. K. John, and J. E. Smith, "The future of simulation: a field of dreams?" *Computer*, vol. 39, no. 11, pp. 22–29, 2006.

[3] D. Campbell, D. Cook, and B. Mulvaney, "A streaming sensor challenge problem for ubiquitous high performance computing," in *Proceedings of 15th Annual Workshop on High Performance Embedded Computing (HPEC '11)*, November 2011.

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

[5] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman, "LogGP: incorporating long messages into the LogP model," in *Proceedings of the 7th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '95)*, pp. 95–105, July 1995.

[6] D. Culler, R. Karp, D. Patterson et al., "LogP: Towards a realistic model of parallel computation," in *Proceedings of the 4th ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming*, pp. 1–12, May 1993.

[7] M. Curtis-Maury, J. Dzierwa, C. D. Antonopoulos, and D. S. Nikolopoulos, "Online power-performance adaptation of multithreaded programs using hardware event-based prediction," in *Proceedings of the 20th Annual International Conference on Supercomputing (ICS '06)*, pp. 157–166, Association for Computing Machinery, July 2006.

[8] S.-J. Lee, H.-K. Lee, and P.-C. Yew, "Runtime performance projection model for dynamic power management," in *Advances in Computer Systems Architecture*, vol. 4697 of *Lecture Notes in Computer Science*, pp. 186–197, Springer, Berlin, Germany, 2007.

[9] D. Snowdon, G. Van Der Linden, S. Petters, and G. Heiser, "Accurate runtime prediction of performance degradation under frequency scaling," in *Proceedings of the Workshop on Operating Systems Platforms for Embedded Real-Time Applications*, 2007.

[10] S. Song and K. W. Cameron, "System-level power-performance efficiency modeling for emergent GPU architectures," in *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques (PACT '12)*, pp. 473–474, ACM, September 2012.

[11] S. Song, M. Grove, and K. W. Cameron, "An iso-energy-efficient approach to scalable system power-performance optimization," in *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER '11)*, pp. 262–271, September 2011.

[12] S. Song, C.-Y. Su, R. Ge, A. Vishnu, and K. W. Cameron, "Iso-energy-efficiency: an approach to power-constrained parallel computation," in *Proceedings of the 25th IEEE International Parallel and Distributed Processing Symposium (IPDPS '11)*, pp. 128–139, IEEE, May 2011.

[13] B. Rountree, D. K. Lowenthal, M. Schulz, and B. R. De Supinski, "Practical performance prediction under dynamic Voltage frequency scaling," in *Proceedings of the International Green Computing Conference (IGCC '11)*, July 2011.

[14] B. Rountree, D. K. Lowenthal, S. Funk, V. W. Freeh, B. R. de Supinski, and M. Schulz, "Bounding energy consumption in

large-scale MPI programs," in *Proceedings of the ACM/IEEE Conference on Supercomputing (SC '07)*, pp. 49:1–49:9, ACM, November 2007.

[15] B. Rountree, D. K. Lowenthal, B. R. de Supinski, M. Schulz, V. W. Freeh, and T. Bletsch, "Adagio: making DVS practical for complex HPC applications," in *Proceedings of the 23rd International Conference on Supercomputing (ICS '09)*, pp. 460–469, ACM, Newport Beach, Calif, USA, June 2009.

[16] B. Rountree, D. H. Ahn, B. R. de Supinski, D. K. Lowenthal, and M. Schulz, "Beyond DVFS: a first look at performance under a hardware-enforced power bound," in *Proceedings of the IEEE 26th International Parallel and Distributed Processing Symposium Workshops (IPDPSW '12)*, pp. 947–953, Shanghai, China, May 2012.

[17] H. Gahvari and W. Gropp, "An introductory exascale feasibility study for FFTs and multigrid," in *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS '10)*, pp. 1–9, IEEE, Atlanta, Ga, USA, April 2010.

[18] A. Bhatele, P. Jetley, H. Gahvari, L. Wesolowski, W. D. Gropp, and L. Kalé, "Architectural constraints to attain 1 exaflop/s for three scientific application classes," in *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS '11)*, pp. 80–91, IEEE, Anchorage, Alaska, USA, May 2011.

[19] K. Czechowski, C. Battaglino, C. McClanahan, K. Iyer, P.-K. Yeung, and R. Vuduc, "On the communication complexity of 3D FFT and its implications for exascale," in *Proceedings of the 26th ACM International Conference on Supercomputing (ICS '12)*, pp. 205–214, June 2012.

[20] K. S. Chatha and R. Vemuri, "An iterative algorithm for hardware-software partitioning, hardware design space exploration and scheduling," *Design Automation for Embedded Systems*, vol. 5, no. 3, pp. 281–293, 2000.

[21] E. Sotiriades and A. Dollas, "Design space exploration for the BLAST algorithm implementation," in *Proceedings of the 15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '07)*, pp. 323–325, April 2007.

[22] A. Stammermann, L. Kruse, W. Nebel et al., "System level optimization and design space exploration for low power," in *Proceedings of the 14th International Symposium on System Synthesis (ISSS '01)*, pp. 142–146, ACM, October 2001.

[23] J. Keinert, M. Streubuhr, T. Schlichter et al., "SystemCoDesigner an automatic ESL synthesis approach by design space exploration and behavioral synthesis for streaming applications," *ACM Transactions on Design Automation of Electronic Systems*, vol. 14, no. 1, article 1, 2009.

[24] K. Lahiri, A. Raghunathan, and S. Dey, "Efficient exploration of the SoC communication architecture design space," in *Proceedings of the IEEE/ACM International Conference on Computer Aided Design (ICCAD '00)*, pp. 424–430, IEEE, Piscataway, NJ, USA, 2000.

[25] K. Lahiri, A. Raghunathan, and S. Dey, "Design space exploration for optimizing on-chip communication architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 6, pp. 952–961, 2004.

[26] M. Palesi and T. Givargis, "Multi-objective design space exploration using genetic algorithms," in *Proceedings of the 10th International Symposium on Hardware/Software Codesign (CODES '02)*, pp. 67–72, May 2002.

[27] H. P. Peixoto and M. F. Jacome, "Algorithm and architecture-level design space exploration using hierarchical data flows," in *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP '97)*, pp. 272–282, July 1997.

[28] P. Mishra, N. Dutt, and A. Nicolau, "Functional abstraction driven design space exploration of heterogeneous programmable architectures," in *Proceedings of the 14th International Symposium on System Synthesis (ISSS '01)*, pp. 256–261, ACM, New York, NY, USA, October 2001.

[29] I. Karkowski and H. Corporaal, "Design space exploration algorithm for heterogeneous multi-processor embedded system design," in *Proceedings of the 35th Annual Design Automation Conference (DAC '98)*, pp. 82–87, San Francisco, Calif, USA, June 1998.

[30] R. Szymanek, F. Catthoor, and K. Kuchcinski, "Time-energy design space exploration for multi-layer memory architectures," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, vol. 1, pp. 318–323, February 2004.

[31] C. L. Janssen, H. Adalsteinsson, and J. P. Kenny, "Using simulation to design extremescale applications and architectures: programming model exploration," *ACM SIGMETRICS Performance Evaluation Review*, vol. 38, no. 4, pp. 4–8, 2011.

[32] A. F. Rodrigues, K. S. Hemmert, B. W. Barrett et al., "The structural simulation toolkit," *ACM SIGMETRICS Performance Evaluation Review*, vol. 38, no. 4, pp. 37–42, 2011.

[33] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, 1965.

[34] S. G. Johnson and M. Frigo, "A modified split-radix FFT with fewer arithmetic operations," *IEEE Transactions on Signal Processing*, vol. 55, no. 1, pp. 111–119, 2007.

[35] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran, "Cache-oblivious algorithms," in *Proceedings of the IEEE 40th Annual Conference on Foundations of Computer Science*, pp. 285–297, October 1999.

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

[37] T. P. Runarsson and X. Yao, "Search biases in constrained evolutionary optimization," *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, vol. 35, no. 2, pp. 233–243, 2005.

[38] S. Johnson, "The NLopt nonlinear optimization package," http://ab-initio.mit.edu/nlopt.

[39] J. S. Vetter, R. Glassbrook, J. Dongarra et al., "Keeneland: bringing heterogeneous GPU computing to the computational science community," *Computing in Science and Engineering*, vol. 13, no. 5, pp. 90–95, 2011.

[40] A. Danalis, G. Marin, C. McCurdy et al., "The scalable heterogeneous computing (SHOC) benchmark suite," in *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units (GPGPU '10)*, pp. 63–74, ACM, March 2010.

[41] S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, and D. Glasco, "GPUs and the future of parallel computing," *IEEE Micro*, vol. 31, no. 5, pp. 7–17, 2011.

[42] P. Kogge, K. Bergman, S. Borkar et al., "Exascale computing study: technology challenges in achieving exascale systems," Tech. Rep., DARPA Information Processing Techniques Office, 2008.