Convexity in Non-convex Optimizations of Streaming Applications

Shobana Padmanabhan
Yixin Chen
Roger D. Chamberlain

Shobana Padmanabhan, Yixin Chen, and Roger D. Chamberlain, "Convexity in Non-convex Optimizations of Streaming Applications," in *Proc. of IEEE 18th International Conference on Parallel and Distributed Systems* (ICPADS), Dec. 2012, pp. 668-675.

Dept. of Computer Science and Engineering
Washington University in St. Louis

2012 IEEE 18th International Conference on Parallel and Distributed Systems

Convexity in Non-convex Optimizations of Streaming Applications

Shobana Padmanabhan, Yixin Chen, and Roger D. Chamberlain
Dept. of Computer Science and Engineering, Washington University in St. Louis
{spadmanabhan, ychen25, roger}@wustl.edu

***Abstract*—Streaming data applications are frequently pipelined and deployed on application-specific systems to meet performance requirements and resource constraints. Typically, there are several design parameters in the algorithms and architectures used that impact the application performance as well as resource utilization. Efficient exploration of this design space is the goal of this research.**

When using architecturally diverse systems to accelerate streaming applications, the design search space is often complex. The search complexity can be reduced by recognizing and exploiting *convex variables* to perform *convex decomposition*, preserving optimality even in the context of a non-convex optimization problem. This paper presents a formal treatment of convex variables and convex decomposition, including a proof that the technique preserves optimality. It also quantifies the reduction in the search space that is realized, at minimum equal to the number of distinct values of the convex variable and potentially much higher.

***Keywords*-design-space exploration; domain-specific branch and bound; decomposition of queueing networks**

I. Introduction

In streaming applications, application data is fed forward over a pipeline of computational and communication elements to achieve high performance. Such applications abound in the areas of biosequence analysis, computer networking, signal processing, video processing, image processing, and computational science. Streaming applications are sufficiently widespread that several programming languages have been developed for them, including Brook [1], and StreamIt [2].

High performance streaming applications are frequently deployed on hybrid systems (employing chip multiprocessors, graphics engines, and reconfigurable logic). Typically, there are many design parameters that impact application performance and resource utilization. The number of options available to the designer is significantly further increased when considering application-specific systems, as the parameter space increases to include architecture parameters as well (e.g., custom datapath designs, cache sizes, instruction sets, etc.). Searching the design space of possible configurations, commonly referred to as design-space exploration, is challenging because:

1) the number of configurations is exponential in the number of design parameters,
2) the design parameters may interact nonlinearly, and
3) the goals of the design-space exploration are often multiple and conflicting.

Sponsored by NSF under grants CNS-0905368 and CNS-0931693.

We approach design-space exploration as an optimization problem in which we seek a globally optimal configuration. Such optimization problems tend to be mixed-integer nonlinear (MINLP) problems that are frequently non-convex and NP-hard. Hence, we have developed in our earlier work a domain-specific branch-and-bound search framework [3]. Starting from queueing network (QN) models [4] that describe the application's performance under steady state, we decompose the optimization problem by exploiting the topology of the application's pipelining, thereby reducing the size of the search space. After decomposition, however, the search space size can still be considerable.

In this paper, we introduce the concept of *convex variables* in the context of an overall non-convex optimization problem. A convex variable is a design parameter for which the optimization problem's objective function is convex when all other parameters are held constant, (i.e., the objective function is convex along the single dimension defined by the convex variable). By recognizing and exploiting these variables, one can achieve additional search space size reductions through what we call *convex decomposition*. In convex decomposition, we decompose the optimization problem and solve for each convex variable separately from the remaining parameters, thus reducing search complexity but preserving optimality. We illustrate this decomposition using an application motivated from computational finance.

Consider, for example, the ingest rate of an application's pipeline. Let us denote this by λ_{in}. If the design goals include both high throughput and low latency, the ingest rate of the pipeline has a conflicting influence on these two goals. Higher ingest rate is clearly associated with a higher throughput. However, due to queueing in the system, a higher ingest rate can also be associated with longer latency. When such design goals are combined using the standard technique of weighted sum, it is possible that the ingest rate is a convex variable, even when the overall optimization problem is non-convex. To our knowledge, we are the first to identify such convex variables and exploit them to reduce the search complexity.

The contributions of this paper are as follows: (1) the formal specification of the MINLP optimization problem (including identifying the problem's general form) and its

1521-9097/12 $26.00 © 2012 IEEE
DOI 10.1109/ICPADS.2012.95

decomposition into Jordon block form, making more precise the informal definitions presented in [3]; (2) a proof that convex decomposition using convex variables, using λ_{in} as an example, preserves optimality; and (3) a queueing network model and the quantitative search space reduction for a real-world problem from computational finance. This paper expands upon results initially disclosed as an extended abstract [5].

II. Related Work and Background

Design-space exploration has been researched extensively for embedded applications. Search heuristics as well as standard and modified optimization techniques have been well researched [6], [7], [8]. Search heuristics, including evolutionary algorithms such as simulated annealing (SA), genetic algorithms, and ant colony algorithms, have limited theory to guarantee optimal solutions, wherein branch and bound guarantees an optimal solution. Even when there is theory to guarantee an optimal solution, as with SA, these approaches can be unrealistic and not practical for finding a global optimum. An example is the logarithmic cooling schedule required by SA to achieve optimality. A recent trend has been to model the application's performance analytically (mathematically) and use the model with search heuristics. Examples of such models include predictive models [9], [10] and examples of search heuristics include gradient ascent [10]. Predictive models are based on regression or machine learning and trained with empirical experimentation through simulation or direct execution. Such models are general and hence can be applied to any design space. Code annotation-based empirical design-space exploration with search heuristics has been used with scientific applications [11]. Auto-tuning has been performed with compiler optimization using search heuristics [12]. For high-performance distributed streaming applications on System S, efficient operator fusion (through graph partitioning and using Integer Programming) is used in the physical processing element graph [13].

Decomposition techniques exist for dealing with complicating variables [14], [15]. For MINLP problems in particular, integer variables are treated as complicating variables and if the resulting nonlinear program (NLP) is convex at least locally, Benders decomposition is known to converge to an optimal solution (or to some small duality gap). Another approach with MINLP problems is to consider the nonlinear constraints as complicating constraints and apply the outer linearization algorithm, provided the nonlinear constraints are inequalities and the objective function is linear (both of which can often be achieved through simple transformations). However, it is often the case that the number of complicating variables and nonlinear constraints in streaming applications is too large for these decomposition techniques to be effective, hence our focus on developing domain-specific decomposition techniques.

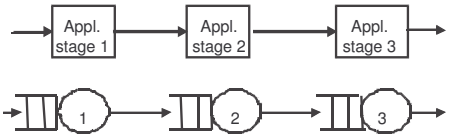

Figure 1: Example streaming application and associated queueing network model.

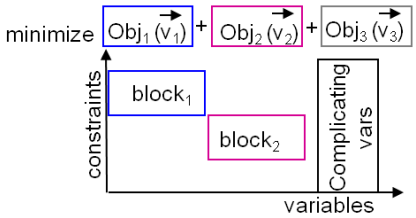

Figure 2: Jordan block form with complicating variables.

A brief overview of our use of QN models through the example of Figure 1 follows. A three-stage pipelined application is modeled with a three-stage QN. Each stage in the QN represents a queueing station (service center). For a BCMP queueing network, the equivalence property states that (under steady-state conditions) each station can be analyzed independently [4]. In such a network, it is conventional to denote the mean job arrival rate at every station j as λ_j and the mean service rate as μ_j. Analytic models for the service rates and network branching probabilities are assumed to come from the user (see Section V).

In our prior work [3], we demonstrated that by exploiting the canonical Jordan block (JB) form [16] (Figure 2 shows the JB form in a generic constraint-variable matrix), we can decompose the problem and optimize the queueing stations in the network separately (as independent Jordan blocks), preserving optimality and achieving significant reductions in the search complexity. In real-world applications, however, there are usually design parameters (i.e., variables) that prevent such decomposition, also shown in Figure 2. Such variables are traditionally called complicating variables. For our applications, we define complicating variables as those variables that concern more than one Jordan block, including the variables that alter the queueing network topology itself.

We categorize the variables informally as follows. Complicating variables that alter the queueing topology are called *topological* variables, and the non-topological complicating variables are called *multi-JB* variables. Variables associated with only a single Jordan block are called *single-JB* variables. Variables that are expressed in terms of the design variables are *derived* variables. Examples are performance metrics, the value of the cost function, and abstractions introduced by queueing theory (which we call *intermediary* variables), such as μ_j and λ_j.

III. Formal Specification

Formal definitions of the variable categories, including a vector form and a set form for each category, are provided below. The relationships between the variable categories are summarized in Figure 3. Subsequently, the general form of the optimization problem is presented.

1) $var = \{var_i | var_i \in \mathbb{R}_+ \cup \{0\}\}$ is the set of design variables in the optimization problem that correspond to the user-identified design parameters. Many of these variables are integer- or binary-valued. $n_v \doteq |var|$ and $\vec{v} \in (\mathbb{R}_+ \cup \{0\})^{n_v}$ is the vector of the variables in var. $var = top \cup mj \cup sj$.
   a) $top \subseteq var$ is the set of topological variables that result in distinct alternative QN topologies. $n_t \doteq |top|$. $\vec{t}$ is the vector formed by these variables.
   b) $mj \subseteq var$ is the set of multi-JB variables, with each element in the domain of more than one $u_j(\cdot)$. $n_{mj} \doteq |mj|$. $\vec{m}$ is the vector formed by these variables.
   c) $sj \subseteq var$ is the set of single-JB variables. Each element of sj is in the domain of only one $u_j(\cdot)$. $n_{sj} \doteq |sj|$. $\vec{s}$ is the vector formed by these variables.
2) $der = \{der_i | der_i \in \mathbb{R}_+ \cup \{0\}\}$ is the set of derived variables that depend on one or more elements in var. $der \bigcap var = \emptyset$.
   a) $met \subseteq der$ is the set of performance metrics. $n_m \doteq |met|$. $\vec{\mathcal{M}}$ is the vector formed by the variables in *met*. $\mathcal{M}_k = o_k(\vec{v}) : (\mathbb{R}_+ \cup \{0\})^{n_v} \rightarrow \mathbb{R}_+ \cup \{0\}$. $\vec{o}$ is the vector of functions that define $\vec{\mathcal{M}}$.
   b) $z \in der$ is the value of the objective function (cost function). $z = \sum_{k=1}^{n_m} W_k \times \mathcal{M}_k : (\mathbb{R}_+ \cup \{0\})^{n_m} \rightarrow \mathbb{R}_+, \sum_{k=1}^{n_m} W_k = 1$, W_k are the weights (including any required normalization).
   c) $ivar \subseteq der$ is the set of intermediary variables. $\vec{i}$ is the vector of variables in $ivar$. $\vec{i} = (\vec{\lambda}, \vec{\mu})$.
      i) $mu \subseteq ivar$ is the set of mean service rates at each queueing station. $\vec{\mu}$ is the vector formed by the variables in *mu*. $\mu_j = u_j(\vec{v}) : (\mathbb{R}_+ \cup \{0\})^{n_v} \rightarrow \mathbb{R}_+$. $\vec{u}$ is the vector of functions that define $\vec{\mu}$.
      ii) $lam \subseteq ivar$ is the set of mean job arrival rates at each queueing station. $\vec{\lambda}$ is the vector formed by the variables in *lam*. Elements of $\vec{\lambda}$ are related by a system of linear equations with a unique solution in terms of the ingest rate, denoted by $\lambda_{in} \in \mathbb{R}_+$. The ingest rate is traditionally referred to as the input mean job arrival rate in the queueing theory literature. $\lambda_j = l_j(\lambda_{in}, \vec{t}) : \mathbb{R}_+ \cup \{0\} \times \mathbb{Z}_+^{n_t} \rightarrow \mathbb{R}_+ \cup \{0\}$. $\vec{l}$ is the vector of functions that define $\vec{\lambda}$.

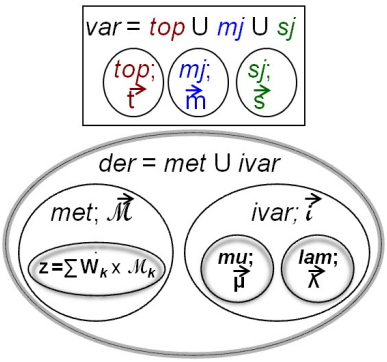

Figure 3: Categories of design variables.

QN topologies are annotated digraphs that may be cyclic. Annotations include, at the minimum, expressions for each of $\vec{u}$, $\vec{l}$, $\vec{o}$, and z; each node represents a queueing station which is a service facility with its queue and each edge, the communication link between the two connected nodes.

In the general form of the optimization problem, functions $\vec{o}, \vec{u}, \vec{l}, \vec{g}, \vec{h}$ might not be convex, differentiable, or even continuous but are assumed to have closed form. The constraints in Equation (4), compared element-wise, ensure stability of the system. The constraints in Equations (5) and (6) are general user-specified constraints.

$$\min_{\vec{v}} \quad z = \sum_{k=1}^{n_m} W_k \cdot o_k(\vec{v}) \tag{1}$$

$$\text{subject to} \quad \mu = \vec{u}(\vec{v}) \tag{2}$$

$$\lambda = \vec{l}(\lambda_{in}, \vec{t}) \tag{3}$$

$$\vec{\lambda} < \vec{\mu} \tag{4}$$

$$\vec{g}(\vec{v}) \leq 0 \tag{5}$$

$$\vec{h}(\vec{v}) = 0 \tag{6}$$

In [3], we provided a search procedure and heuristic variable branching order that exploits the queueing network topology, reducing the search space from the product of the domain sizes of variables in sj to their sum. In what follows, we show conditions under which the optimum value for $\lambda_{in} \in mj$ can be analytically determined (i.e., does not need to have all of its possible values considered).

IV. Convex Decomposition

As stated earlier, the application's ingest rate, denoted by λ_{in}, is a convex variable if $z(\lambda_{in})$ is convex. A real differentiable function of a single variable is convex on an interval if and only if its derivative is monotonically non-decreasing on that interval [17]. In the presence of a convex variable, if one can effectively choose optimal values for the remaining variables, independent of the value of the convex variable, determination of the value of the convex variable

is then straightforward. This is precisely the focus of convex decomposition. In convex decomposition, we decompose the optimization problem to exclude the convex variable and solve for the remaining variables as a new optimization problem (with a new objective function). The solution is then substituted back in the original problem and we then solve for the convex variable analytically (rather than branching on all of its values). In what follows, we prove that the resulting configuration is globally optimum.

Notation. When the general optimization problem formulated above satisfies the sufficient conditions for λ_{in} being a convex variable and is amenable for convex decomposition, it takes the form shown below. We denote such a problem as P. The variables in $\vec{v}$ are organized as $\vec{v} = (\vec{m},\ \vec{s_1}, \ldots, \vec{s_n})$. $\vec{m}$ are the only complicating variables remaining after branching on topological variables, per the heuristic for ordering branching variables [3]. Recall that $\lambda_{in} \in mj$. Let us denote variables other than λ_{in} in $\vec{v}$ by $\vec{v'}$. The single-JB variables (non-complicating) $\vec{s_j}$ concern queueing station j; $\vec{s_j} = (s_{j1}, s_{j2}, \ldots, s_{jn_j})$. n_j denotes the number of elements in each $\vec{s_j}$. It is assumed that there are no additional constraints specified by the user. The problem formulation of P is:

$$\min_{\vec{v}} \quad z = \sum_{k=1}^{n_m} W_k \cdot o_k(\vec{i}) \tag{7}$$

$$\text{subject to} \quad \vec{\mu} = \vec{u}(\vec{v'}) \tag{8}$$

$$\lambda_j = K_j \cdot \lambda_{in},\ \lambda_{in} \in \mathbb{R}^+; \tag{9}$$

$$K_j \text{ are scaling factors}\,; \quad j = 1, 2, \ldots, n$$

$$\vec{\lambda} < \vec{\mu} \tag{10}$$

For a convex variable to result in convex decomposition, we identify the following sufficient conditions: a) none of performance metrics in the objective function suffer from higher service rates at any queueing station; b) variables in $\vec{\mu}$ depend only on variables in $\vec{v'}$ whereas variables in $\vec{\lambda}$ depend only on λ_{in}, as shown in Equations (8) and (9); and c) λ_{in} is not involved in the bound of any other design variable, while it can be bounded itself.

Let P_λ be the optimization problem when λ_{in} is assigned a particular value Λ.

$$\min_{\vec{v'}} \quad z_\lambda = \sum_{k=1}^{n_m} W_k \cdot o_k(\vec{i}) \tag{11}$$

$$\text{subject to} \quad \vec{\mu} = \vec{u}\left(\vec{v'}\right) \tag{12}$$

$$\lambda_j = K_j \cdot \Lambda,\ j = 1, 2, \ldots, n \tag{13}$$

$$\vec{\lambda} < \vec{\mu} \tag{14}$$

Let P_T be the problem created by transforming P_λ as shown below.

$$\max_{\vec{v'}} \quad z_T = \sum_{j=1}^{n} \mu_j \tag{15}$$

$$\text{subject to} \quad \vec{\mu} = \vec{u}\left(\vec{v'}\right) \tag{16}$$

Let $\vec{s}_T$ be the solution to P_T (i.e., $\vec{s}_T$ is the optimal assignment to v') and let $\vec{\mu}^*$ represent optimal values of $\vec{\mu}$ in the solution of P_T.

Let P' be the problem obtained by substituting $\vec{s}_T$ in P as shown below. λ_{in} is the only design variable to be solved for in P'.

$$\min_{\lambda_{in}} \quad z' = \sum_{k=1}^{n_m} W_k \cdot o_k(\vec{\mu}^*, \vec{\lambda}) \tag{17}$$

$$\text{subject to} \quad \vec{\lambda} < \vec{\mu}^* \tag{18}$$

$$\lambda_j = K_j \cdot \lambda_{in},\ j = 1, 2, \ldots, n \tag{19}$$

P' is a convex optimization problem because $z(\lambda_{in})$ being convex means $z'(\lambda_{in})$ is also convex and the constraints are linear. The solution to P', denoted by s', will provide an optimal value for λ_{in} in P'. The corresponding optimal values of $\vec{\lambda}$ are denoted by $\vec{\lambda}^*$.

Claim. *When each measure in the objective function of P (denoted by o_k) decreases monotonically with each variable in $\vec{\mu}$, we can solve P optimally by solving optimally the problems of P_T and P' described above. That is, solution $\vec{s}_T$ from P_T and s' from P' together constitute an optimal solution of $\vec{v}$ in P.*

Proof: Let $\vec{\tilde{\lambda}}$ and $\vec{\tilde{\mu}}$ denote the optimal values of $\vec{\lambda}$ and $\vec{\mu}$ if we solved P (optimally). Let $\tilde{z}$ denote the objective function value obtained by substituting $\vec{\tilde{\lambda}}, \vec{\tilde{\mu}}$ in z, (i.e.),

$$\tilde{z} = \sum_{k=1}^{n_m} W_k \cdot o_k(\vec{\tilde{\mu}}, \vec{\tilde{\lambda}}) \tag{20}$$

Let z^* denote the objective function value in P obtained by substituting $\vec{\mu}^*$ from the optimal solution of P_T and $\vec{\tilde{\lambda}}$ from the optimal solution of P, (i.e.),

$$z^* = \sum_{k=1}^{n_m} W_k \cdot o_k(\vec{\mu}^*, \vec{\tilde{\lambda}}) \tag{21}$$

$\vec{\mu}^* \geq \vec{\tilde{\mu}}$ because P_T maximizes each μ_j while the original objective function in P may not always maximize each μ_j. Further, each o_k decreases monotonically with each element of $\vec{\mu}$. These together make $z^* \leq \tilde{z}$. It then follows that $\vec{s}_T$ contains optimal values of $\vec{v'}$ in P. With $\vec{v'}$ fixed at their optimal values, when we solve for λ_{in} in P', we get the optimal value of λ_{in} in P'. Recall that P' is the problem remaining when only λ_{in} is to be solved for in P. Thus, we now have optimal values for all the variables in $\vec{v}$ in P.

This proves our claim.

■

Remark. If P_T happens to be convex, theory exists for solving P_T in polynomial time. If P_T is not a convex problem, however, the decomposition technique from [3] can be used to reduce the search complexity.

Reduction in the number of branch evaluations: Let there be n_v variables with each variable having k values. Then, the number of leaves in a branch and bound tree is given by k^{n_v}. However, when there is a convex variable, the number of leaves reduces to k^{n_v-1}. That is, the reduction in the number of leaves is directly proportional to the number of values considered for the variable. When P_T is a convex optimization problem, however, any tree node evaluating the convex variable becomes a terminal node (leaf). Let us say the convex variable is evaluated at height h of the tree. In this case, the number of leaves reduces to k^h. When $h << n_v$, the reduction in the number of leaves is significant.

V. Application

There are a number of circumstances under which balancing latency and throughput is a valid concern for the application developer. In computational finance, the problem of risk assessment for a portfolio entails both latency concerns (trades need to be executed quickly) and throughput concerns (many potential candidate portfolios need to be assessed) [18]. This application includes a sort function which is used below to illustrate our technique. Generally, any decision-making process that is time sensitive and operates on large data sets frequently has this need to balance latency and throughput. In addition to computational finance, examples include sorting of agricultural seeds [19], face recognition [20], etc.

In computational finance, while computing the value-at-risk for a portfolio of financial instruments, there is a repeated requirement to sort a reasonably large number of elements (e.g., a few million) at high throughput with minimum latency [18]. The effective use of architecturally diverse platforms for sorting has received considerable attention in the literature [21], [22].

In the streaming sort application, input data is *split* into parts and each part is sent to a *sort* instantiation. This application topology is shown in Figure 4. The sort instantiation is referred to as a sort "block". The sort blocks execute in parallel and when done, each block sends its output to a *merge* block. The merge block then merges its inputs and sends out the sorted data.

The application parameters, or design variables, include the degree of parallelism embodied in the number of sort blocks employed. The number of sort blocks impacts the application topology, as illustrated in Figure 5, which shows a 4 sort block instantiation. Additional design variables include: number of elements to be sorted, computational

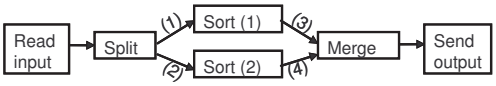

Figure 4: Streaming sort application.

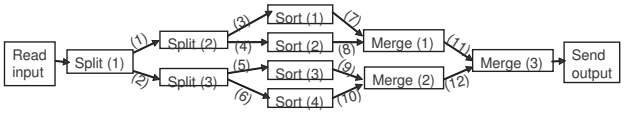

Figure 5: A streaming sort application with 4 sort blocks.

resource types (processor cores or FPGA) and their counts (how many of each), sorting algorithm (within the sort blocks), message size on a communications link, data ingest rate, and others. The complete set of design variables, along with their characterization, bounds, and constraints are enumerated in Figure 6. Only a few interesting and illustrative mappings are considered here. t_0 models the topology when every compute block in the application gets its own set of resources, t_1 models when all compute blocks share a single resource, t_{CR} models when all split blocks share a single computation resource, and t_{IR} models when the communication links into and out of the sort blocks are shared.

For the application topology shown in Figure 4, an $M/M/1$ BCMP queueing model is shown in Figure 7. Note that each pipeline stage, including the communication links, is modeled as an individual queueing station. Some mapping choices change the queueing network's topology. An example of such a mapping choice is t_{IR}. The resulting queueing network in illustrated in Figure 8, where the server "Comm" is handling all of the communication both into and out of the "Sort" server.

The expressions for mean service rate at each server, μ_j, and the relationships between the mean arrival rates, λ_j, are

Figure 7: Queuing network model for the application topology in Figure 4.

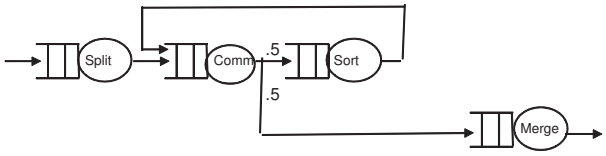

Figure 8: Queueing network has feedback when communication edges to/from sort blocks are shared.

Variable	Symbol	Ranges and Constraints
Number of elements to be sorted	2^{B_SZ}	$B\_SZ \in \mathbb{Z}_+$ (e.g., 20)
Number of sort blocks	2^N	N = 1, 2, 3, ..., $\frac{B\_SZ}{2}$; $N \in top$
Index of split stages of link stages left of sort stage of sort stage of link stages right of sort stage of merge stages	j	$j = 0, 2, ..., 2N-1$ $j = 1, 3, ..., 2N-1$ $j = 2N$ $j = 2N+1, 2N+3, ..., 4N-1$ $j = 2N+2, 2N+4, ..., 4N$
Compute resource type	binary: cpu_j or $fpga_j$	$j = 0, 2, ..., 4N$; $cpu_j + fpga_j = 1$ each $cpu_j \in mj$; each $fpga_j \in mj$;
Communication resource type	binary: $smem_j$ or $gige_j$	$j = 1, 3, ..., 4N-1$; $smem_j + gige_j = 1$ each $smem_j \in mj$; each $gige_j \in mj$;
Number of compute resources	$nRes_j$	$j = 0, 2, ..., 4N$; $\forall j$, $nRes_j \geq 1$; $nRes_j \in sj$ $\forall$ split, sort stages: $nRes_j \leq 2^{\frac{j}{2}}$ $\forall$ merge stages: $nRes_j \leq 2^{\frac{4i-j}{2}}$
Number of communication resources	$nRes_j$	$j = 1, 3, ..., 4N-1$; $\forall\ j$, $nRes_j \geq 1$; $nRes_j \in sj$ $\forall$ links left of sort: $nRes_j \leq 2^{\frac{j+1}{2}}$ $\forall$ links right of sort: $nRes_j \leq 2^{\frac{4i-j+1}{2}}$
Mapping choices	binary: t_0, t_1, t_{CR}, t_{IR}	t_0 + t_1 + t_{CR} + t_{IR} = 1; each $t_i \in top$
System-wide comm message size	2^M	$M = 0, 1, \ldots, M_{UB}$ where $M_{UB} \leq N$; $M_{UB} \in \mathbb{Z}_+$ (e.g., 14); $M \in mj$
Sort algorithm (only with cpu_j mapping)	binary: alg_1 or alg_2	$t_1(fpga_0 + alg_1 + alg_2)$ $+(1-t_1)(fpga_j + alg_1 + alg_2) = 1, j = 2N$; $alg_1 \in sj$; $alg_2 \in sj$;
Input mean job arrival rate	$\lambda_{in} \in \mathbb{R}_+$	Constrained by Equation (3); $\lambda_{in} \in mj$

Figure 6: Design variables for streaming sort. Binary variables are used to select among resource types (processor core vs. FPGA), mappings, and sort algorithms.

derived from first principles or using spline fitting [10] and have been validated to be within 1% of published empirical results [21], [23]. Arrival rates at each queueing station are related as follows.

$$\begin{aligned}
\lambda_j &= \lambda_{in} \quad for \quad j = 1, 2, ..., 2N-2 \\
\lambda_j &= 2\lambda_{in} \quad for \quad j = 2N-1 \\
\lambda_j &= \lambda_{in} \quad for \quad j = 2N \\
\lambda_j &= (1-t_1) \cdot \lambda_{in} \quad for \quad j = 2N+1 \\
\lambda_j &= \lambda_{in} \quad for \quad j = 2N+2, 2N+3, ..., 4N
\end{aligned} \tag{22}$$

The expression for the mean service rate of the sort blocks is shown below and the expressions for split and merge blocks are similar. The Cs in the equation are constants, while cpu_j, $fpga_j$, t_i, and alg_i are binary selection variables that indicate the type of compute resource (processor core or FPGA), mapping choice, and algorithm. The number of resources deployed at level j in the application topology is indicated by $nRes_j$. The performance effect of changing the number of computational resources assigned to a stage is modeled through a scaling factor.

$$\begin{aligned}
t_c &= \frac{alg_1 \cdot C_1 \cdot nRes_0}{2^{B_SZ} \cdot \log\left(\frac{2^{B_SZ}}{2^{j/2}}\right)} + \frac{alg_2 \cdot C_2 \cdot nRes_0}{2^{2 \cdot B_SZ}} \\
t_f &= \frac{C_3 \cdot nRes_0}{2^{B_SZ}} \\
r_c &= \frac{alg_1 \cdot C_1 \cdot nRes_j}{2^{B_SZ} \cdot \log\left(\frac{2^{B_SZ}}{2^{j/2}}\right)} + \frac{alg_2 \cdot C_2 \cdot nRes_j}{2^{2 \cdot B_SZ}} \\
r_f &= \frac{C_3 \cdot nRes_j}{2^{B_SZ}} \\
\\
\mu_j &= (t_1) \cdot (cpu_0 \cdot t_c + fpga_0 \cdot t_f) + \\
&\quad (1-t_1) \cdot (cpu_j \cdot r_c + fpga_j \cdot r_f) \\
&\quad j \in \{sortIndex\}
\end{aligned} \tag{23}$$

For communication elements, μ_j is defined as a function of the communication message size denoted by M, in addition to the factors mentioned earlier. The normalized effective data rate (as a function of message size, M) is denoted by R, and is modeled using a restricted cubic spline model [10]. β_i are the coefficients and κ_i are the knot positions obtained from the regression fitting.

$$R = \beta_0 + \beta_1 \cdot 2^M + \sum_{i=2}^{i=8} \beta_i \cdot \max(2^M - \kappa_{i-1}, 0)^3 \quad (24)$$

$$\begin{aligned}
\mu_t &= (cpu_0 \cdot C_4 + fpga_0 \cdot C_5)\, \frac{2^{\frac{j+1}{2}} \cdot R}{2^{B_SZ}} \\
\mu_c &= (fpga_0 \cdot fpga_{j+1} \cdot C_6 + \\
&\quad cpu_0 \cdot cpu_{j+1} \cdot C_7 + \\
&\quad [fpga_0 \cdot cpu_{j+1} + cpu_0 \cdot fpga_{j+1}] \cdot C_8) \cdot \\
&\quad \frac{2^{\frac{j+1}{2}} \cdot R \cdot nRes_j}{2^{B_SZ}} \\
\mu_r &= (fpga_{j-1} \cdot fpga_{j+1} \cdot C_6 + \\
&\quad cpu_{j-1} \cdot cpu_{j+1} \cdot C_7 + \\
&\quad [fpga_{j-1} \cdot cpu_{j+1} + cpu_{j-1} \cdot fpga_{j+1}] \cdot C_8) \cdot \\
&\quad \frac{2^{\frac{j+1}{2}} \cdot R \cdot nRes_j}{2^{B_SZ}} \\
\mu_j &= (t_1) \cdot \mu_t + (t_{CR}) \cdot \mu_c + (t_0 + t_{IR}) \cdot \mu_r \quad (25)
\end{aligned}$$

As motivated by Singla et al. [18], the performance metrics for the streaming sort application include both minimizing latency and maximizing throughput. These metrics are normalized and combined using the standard technique of weighted sums, as shown in Equation (26). Note that if we optimized only the application's throughput, given by λ_{in}, the problem degenerates to identifying the bottleneck in the pipeline.

$$minimize \quad z = W_1 \cdot Latency + W_2 \cdot \frac{1}{\lambda_{in}}, \sum_{1}^{2} W_i = 1 \quad (26)$$

From queueing theory (for $M/M/1$ BCMP networks), latency is given by:

$$Latency = \sum_{j=0}^{4N} \frac{1}{\mu_j - \lambda_j} \quad (27)$$

The number of variables and constraints in the original problem range from (50, 30) to (399, 3077) as N is increased from 1 to 13. This corresponds to $2^N = 2$ to 8192 sort blocks. Variables other than λ_{in} are integer-valued, making the problem mixed-integer, and the objective function and constraints are non-linear in a number of design variables.

VI. Empirical Results

In this section, we will articulate the benefits of identifying a convex variable and exploiting it via convex decomposition. We accomplish this through the use of the sort application (and its queueing model) described in the previous section.

$z(\lambda_{in})$ in Equation (26) is convex and the optimization problem formulation for sort satisfies the sufficient conditions for convex decomposition. The reduction in the number of branch evaluations we enjoy thanks to the convex decomposition is illustrated in Figure 9. The figure assumes that without convex decomposition we would branch on at least 100 discrete values of λ_{in}.

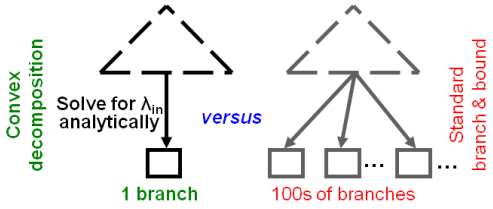

Figure 9: Branching on λ_{in}.

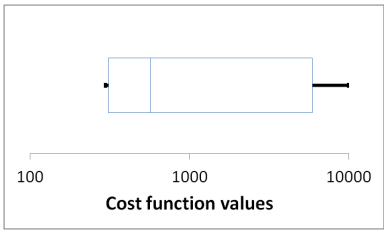

Figure 10: Box-whisker plot of cost function values of 1000 randomly generated configurations. Values greater than 10,000 are excluded as outliers (extending well into the millions).

To illustrate the breadth of possible cost function values for this problem, Figure 10 shows a box-whisker plot of cost function values for 1000 uniformly distributed randomly generated configurations. The box-whisker plot shows the minimum, 25th quartile, median, 75th quartile, and maximum (excluding outliers) for the sampled cost function values. As is clear from the figure, poorly chosen configurations can result in very poor performance.

For the sorting application, the size of the search space is a strong function of the allowed range for N, which determines the maximum number of parallel sorts. For N ranging from just 1 to 3 (i.e., 2 to 8 parallel sorts), the size of the completely enumerated branch and bound search space is approximately 10^{20} possible configurations. Using the techniques presented earlier [3], this search space decreases to approximately 10^{10} configurations, reducing by a factor of 10 million. Convex decomposition reduces the space by an additional factor of 100, assuming 100 discrete values for λ_{in}, which results in a search space of approximately 10^8. If instead, 1000 discrete values get evaluated for λ_{in}, the reduction would be a factor of 1000 (i.e., the reduction is directly proportional to the discretization of the convex variable).

While in the general case the search space size is still exponential in the number of the design parameters, for the sort application presented above the reduced size of the

search space can now be searched in approximately 1 hr if each evaluation takes no more than 10 μs.

VII. Conclusions

This paper has introduced the notion of a convex variable in the context of an overall non-convex optimization problem. For streaming applications, the data ingest rate is a convex variable when $z(\lambda_{in})$ is convex and sufficient conditions are identified for a convex variable to lead to convex decomposition (providing the ability to exploit the convex variable to reduce the size of the search space). A proof is presented to demonstrate the fact that convex decomposition preserves the optimality of the search results, and a quantification of the search space size reduction is given. These ideas are illustrated using a real-world problem as an example, showing both the performance model and the reduction in the size of the search space.

Future work includes: (1) an attempt to identify other common design parameters that might be frequently convex, such as clock frequency when optimizing for both throughput and power consumption; and (2) the investigation of other potential decomposition opportunities applicable to streaming applications.

References

[1] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan, "Brook for GPUs: stream computing on graphics hardware," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 777–786, 2004.

[2] W. Thies, M. Karczmarek, and S. Amarasinghe, "StreamIt: A Language for Streaming Applications," *11th Int'l Conf. on Compiler Construction*, pp. 179–196, 2002.

[3] S. Padmanabhan, Y. Chen, and R. D. Chamberlain, "Optimal design-space exploration of streaming applications," *22nd IEEE Int'l Conf. on Application-specific Systems, Architectures and Processors*, pp. 227–230, Sep. 2011.

[4] F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios, "Open, closed, and mixed networks of queues with different classes of customers," *J. ACM*, vol. 22, no. 2, pp. 248–260, 1975.

[5] S. Padmanabhan, Y. Chen, and R. Chamberlain, "Design-space optimization for automatic acceleration of streaming applications," *Symp. on Application Accelerators in High Performance Computing*, Jul. 2010, [Extended abstract].

[6] J. Cong, K. Gururaj, and G. Han, "Synthesis of reconfigurable high-performance multicore systems," *ACM Int'l Symp. on Field Programmable Gate Arrays*, pp. 201–208, 2009.

[7] A. Dasgupta and R. Karri, "Optimal Algorithms for Synthesis of Reliable Application-Specific Heterogeneous Multiprocessors," *IEEE Trans. on Reliability*, vol. 44, no. 4, pp. 603–613, Dec. 1995.

[8] A. Pimentel, C. Erbas, and S. Polstra, "A systematic approach to exploring embedded system architectures at multiple abstraction levels," *IEEE Trans. on Computers*, vol. 55, no. 2, pp. 99–112, Feb. 2006.

[9] E. Ipek, S. A. McKee, K. Singh, R. Caruana, B. R. de Supinski, and M. Schulz, "Efficient architectural design space exploration via predictive modeling," *ACM Trans. Archit. Code Optim.*, vol. 4, no. 4, pp. 1–34, 2008.

[10] B. C. Lee and D. Brooks, "Roughness of microarchitectural design topologies and its implications for optimization," *High Performance Computer Arch.*, pp. 240–251, 2008.

[11] A. Hartono, B. Norris, and P. Sadayappan, "Annotation-based empirical performance tuning using Orio," *Int'l Parallel and Distributed Processing Symp.*, 2009.

[12] A. Tiwari, C. Chen, J. Chame, M. Hall, and J. K. Hollingsworth, "A scalable auto-tuning framework for compiler optimization," *Int'l Parallel and Distributed Processing Symp.*, 2009.

[13] J. Wolf, R. Khandekar, K. Hildrum, S. Parekh, D. Rajan, and K.-L. Wu, "COLA: Optimizing Stream Processing Applications Via Graph Partitioning," *ACM/IFIP/USENIX 10th Int'l Middleware Conf.*, 2009.

[14] A. J. Coneja, E. Castillo, R. Minguez, and R. Garcia-Bertrand, *Decomposition Techniques in Mathematical Programming Engineering and Science Applications*. Springer, 2006.

[15] D. Bertsekas, *Nonlinear Programming*, 2nd ed. Athena Scientific, 2003.

[16] G. Strang, *Introduction to Linear Algebra*. Wellesley-Cambridge Press, 1980.

[17] W. Rudin, *Principles of Mathematical Analysis*. McGraw-Hill, 1976.

[18] N. Singla, M. Hall, B. Shands, and R. D. Chamberlain, "Financial Monte Carlo Simulation on Architecturally Diverse Systems," *Workshop on High Performance Computational Finance*, Nov. 2008.

[19] A. Dell'Aquila, "Development of novel techniques in conditioning, testing, and sorting seed physiological quality," *Seed Science and Technology*, vol. 37, pp. 608–624, 2009.

[20] P. Viola and M. Jones, "Robust real-time object detection," *Int'l J. of Computer Vision*, vol. 57, no. 2, pp. 137–154, 2002.

[21] R. D. Chamberlain and N. Ganesan, "Sorting on architecturally diverse computer systems," *Int'l Workshop on High-Performance Reconfigurable Computing Technology and Applications*, Nov. 2009.

[22] C. Grozea, Z. Bankovic, and P. Lasko, "FPGA vs. multi-core CPUs vs. GPUs: Hands-on experience with sorting," *Facing the Multi-Core Challenge: Conf. for Young Scientists at the Heidelberger Akademie der Wissenschaften*, 2010.

[23] R. D. Chamberlain, G. A. Galloway, and M. A. Franklin, "Sorting as a streaming application executing on chip multiprocessors," Dept. of Computer Science and Engineering, Washington University, Tech. Rep. WUCSE-2010-21, 2010.