# Unchaining in Design-space Optimization of Streaming Applications

**Shobana Padmanabhan**
**Yixin Chen**
**Roger D. Chamberlain**

Shobana Padmanabhan, Yixin Chen, and Roger D. Chamberlain, "Unchaining in Design-space Optimization of Streaming Applications," in *Proc. of Workshop on Data-Flow Execution Models for Extreme Scale Computing* (DFM), September 2013.

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

# Unchaining in Design-space Optimization 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*—Data-streaming applications are frequently pipelined and deployed on hybrid systems to meet performance requirements and resource constraints. With freedom in the design of algorithms and architectures, the search complexity can explode. A popular approach to reducing search complexity is to decompose the search space while preserving optimality.**

**We present a novel decomposition technique called unchaining that partitions the problem such that the resulting subproblems are less complex. Thanks to unchaining, the number of subproblems from the decomposition is linear in the number of chained blocks in the variable-constraint matrix (instead of being their product). Finally, we present a queueing network model and the quantitative search space reduction for a real-world implementation of a biosequence search application called BLASTN.**

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

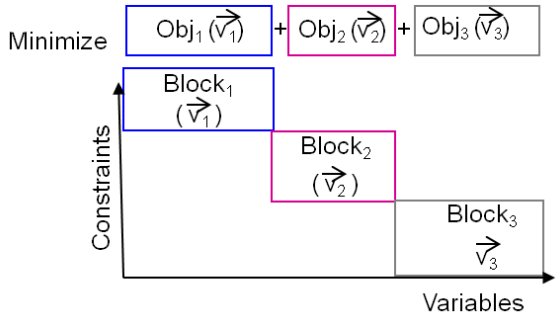


Fig. 1: Constraint-variable matrix illustrating decomposability from independent blocks; $\vec{v_i}$ denotes variables and $Obj$ denotes the objective function being optimized.

## I. Introduction

In course-grained data-flow applications, application data is fed over a *pipeline* of computational and communication elements to achieve high performance. Such streaming data applications abound in the areas of biosequence analysis, computer networking, signal processing, video processing, image processing, and computational science. These applications are sufficiently widespread that several programming languages such as Brook [3], StreamIt [22], and X [9] and deployment platforms such as Auto-Pipe [9] and System S [10] have been developed for them.

One of the attractions of the streaming programming paradigm is that it represents one form of "safe" parallelism. In contrast to being concerned with threads, locks, and races [16], the application developer is allowed to express the semantics of the computation in a manner quite similar to a sequential program. The operations on each data element are sequentially applied as the element moves down the pipeline. The parallelism comes from the pipelining itself.

High performance streaming applications are frequently deployed on architecturally diverse (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 further increased significantly 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.

We have approached the design-space exploration of streaming applications as an optimization problem that searches for a globally optimal configuration [18]. We use cost functions derived from queueing network models of the applications, in particular, BCMP networks [2]. Such optimization problem formulations tend to be mixed-integer nonlinear (MINLP) problems that are frequently non-convex and NP-hard. To reduce the search complexity, we exploit topological information about the application's pipelining that gets embodied in the queueing network models. More precisely, we use the topological information to identify independent blocks in the variable-constraint matrix (see Figure 1) and solve the blocks separately [5], thereby reducing the size of the search space.

When the BCMP independence property is violated however, service rates of the service centers are not independent. An example is when the buffers at the service centers in a queueing network are "bounded," service rate of a service center (node) will be affected by the queue occupancy at the downstream node in the network [19]. We call such service centers and the corresponding blocks in the variable-constraint

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

matrix as being *chained*. Here, we focus on blocks that are chained linearly and unidirectionally through one variable.

We present a novel decomposition technique that we are calling *unchaining* to unchain the blocks in such a way that they can be solved independently, while preserving feasibility. We require only a feasible solution during unchaining. We illustrate unchaining using the biosequence search application BLASTN [1].

The contributions of this paper are as follows: (1) identify sufficient conditions for unchaining and prove that unchaining preserves feasibility; and (2) show a queueing network model and the quantitative search space reduction for a real-world BLASTN implementation.

## II. Related Work

Design-space exploration has been researched extensively for embedded applications. Search heuristics as well as standard and modified optimization techniques have been well investigated [6], [20]. 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. 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 [14], [15] and examples of search heuristics include gradient ascent [15]. Code annotation-based empirical design-space exploration with search heuristics has been used with scientific applications [12]. Auto-tuning has been performed with compiler optimization using search heuristics [23]. 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 [24].

Decomposition is popular in existing methods for solving nonlinear programming (NLP) problems. Many mixed-integer nonlinear programming (MINLP) methods are based on subspace partitioning and decompose the search space of a problem instance into subproblems. Examples include the following: (a) generalized Benders decomposition (GBD) decomposes a problem space into multiple subspaces by fixing the values of its discrete variables, and by using a master problem to derive bounds and to prune inferior subproblems [11]; (b) outer approximation (OA) is similar to GBD except that the master problem is formulated using primal information and outer linearization [8]; (c) generalized cross decomposition (GCD) iterates between a phase solving the primal and dual subproblems and a phase solving the master problem [13]; and (d) branch-and-reduce methods solve MINLPs and NLPs by a branch-and-bound algorithm and exploit factorable programming to construct relaxed problems [21]. All these methods require the original problem to have special decomposable structures and the subproblems to have some special properties, such as nonempty and compact subspaces with convex objective and constraint functions. Another class of decomposition methods is separable programming methods based on duality [5]. By decomposing a large problem into multiple much simpler subproblems, they have similar advantages as our decomposition techniques. However, they are limited in their general applications because they have restricted assumptions, such as linearity or convexity of functions.

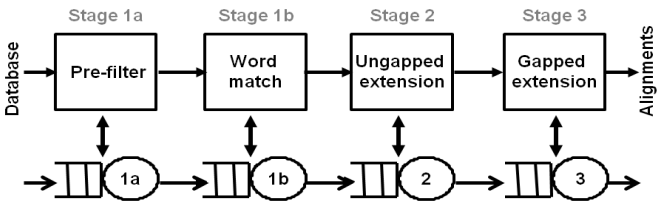


Fig. 2: BLASTN streaming application and example queueing network model.

## III. Background

BLASTN is a biosequence similarity search application where data flows from one kernel to the next as depicted in Figure 2 (which also shows a queueing network model of the application's data flow). Here, we provide a brief overview of our use of queueing network models using Figure 2. More details of the application and the queueing network model are presented in Section V.

The four-stage course-grain data-flow application is modeled with a four-stage queueing network where each stage represents a *service center* comprising a server and its buffer. For a BCMP queueing network, the *equivalence property* states that (under steady-state conditions) each service center can be analyzed independently [2]. In such a network, it is conventional to denote the mean job arrival rate at every service center $j$ as $\lambda_j \in \mathbb{R}$ and the mean service rate as $\mu_j \in \mathbb{R}$.

In real-world applications, there are usually design parameters (i.e., variables) that prevent the decomposition illustrated in Figure 1. Such variables are traditionally called complicating variables. Complicating variables are variables that concern more than one block.

We have developed a domain-specific branch and bound technique to find the optimal configuration. However, for MINLP optimization problems, standard relaxation techniques do not work well. Hence, to improve search efficiency, we exploit topological information about the pipelining to order the branching variables and also reduce the number of configurations to evaluate [18]. Our ordering heuristic is to first branch on the complicating variables, followed by non-complicating variables, and finally, variables associated with chained blocks.

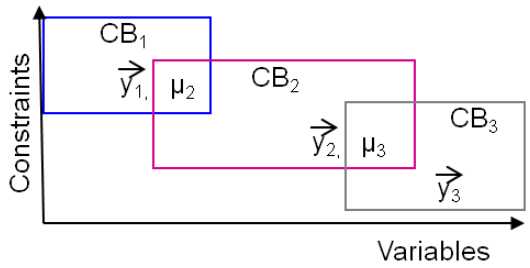


Fig. 3: Chaining of blocks.

## IV. Unchaining

We define a *chaining variable* as a derived variable that connects two adjacent blocks that are otherwise independent. A set of $k$ chaining variables would then chain $k+1$ adjacent blocks. When applying unchaining, we identify the following sufficient conditions: (a) the only variables remaining in the original optimization problem (corresponding to the design-space exploration) are the variables associated with the chained blocks; and (b) the value of the objective function in the original optimization problem is known (for instance, by virtue of already branching on the design parameters not involved in chaining). A consequence of condition (b) is that when we perform unchaining, we need to find only a feasible solution. In what follows, we prove that unchaining preserves feasibility.

**Notation.** Let $k$ adjacent blocks be *chained*. We index these chained blocks (CBs) as $CB_i, i = 1, 2, \cdots, k$. Let $P_F$ be the feasibility (a.k.a. satisfiability) problem in the general form shown below. Input design variables are denoted as $\vec{v} = (\vec{y}_1, \vec{y}_2, \ldots, \vec{y}_k)$. Each $\vec{y}_i = (y_{1i}, y_{2i}, \ldots, y_{in_i})$; $n_i$ denotes the number of elements in each $\vec{y}_i$ and the variables may be real or integer valued and are non-complicating. $\vec{d} = (\vec{\mu})$ are derived variables where $\vec{\mu} = (\mu_1, \mu_2, \ldots, \mu_k)$. The derived variables are all real-valued. $\vec{\mu}$, with the exception of $\mu_1$, are the variables that chain the blocks linearly as shown in $P_F$ below. An example of this is illustrated using three blocks in Figure 3 where $\mu_3$ chains $CB_3$ and $CB_2$ while $\mu_2$ chains $CB_2$ and $CB_1$. Functions $\vec{u}, \vec{g}, \vec{h}$ may not even be continuous but are assumed to have closed-form. The form of $P_F$ is:

*Find a feasible solution*

$$\begin{aligned} \text{subject to} \quad & \mu_i = u_i(\vec{y}_i,\ \mu_{i+1}),\ i = 1, 2, \ldots, k-1 \\ & \mu_k = u_k(\vec{y}_k) \\ & \mu_i > L_i,\ i = 1, 2, \ldots, k; L_i \text{ are given constants} \\ & \vec{g}_i\left(\vec{y}_i\right) \ \leq\ 0,\ i = 1, 2, \ldots, k \\ & \vec{h}_i\left(\vec{y}_i\right) \ =\ 0,\ i = 1, 2, \ldots, k \end{aligned} \tag{1}$$

Note that every chaining variable chains only two adjacent blocks. We refer to the block where a chaining variable gets defined as its *right-block* and the block where it gets used as its *left-block*. For example, in Figure 3, the right-block of $\mu_3$ is $CB_3$ and its left-block is $CB_2$.

We unchain the blocks as follows. We begin by finding the upper and lower bounds of every chaining variable in its right-block and using the bounds to constrain the selection of a range of feasible values of the chaining variable in its left-block. We find the bounds by formulating a pair of maximization and minimization problems for each of $\mu_k, \mu_{k-1}, \ldots, \mu_2$. These problems are denoted as $P_i^{MAX}, P_i^{MIN}, i = k, k-1, \cdots, 2$.

Then, we assign feasible values for the variables starting with $CB_1$. We denote this problem as $P_1$. We then proceed with assigning values to variables associated with each of $CB_2, CB_3, \ldots, CB_k$, using the solution of the chained variable selected in its left-block to assign feasible values for the variables of its right-block. These problems are denoted as $P_i, i = 2, 3, \ldots, k$. The solutions to these problems are denoted as $\vec{s}_1, \vec{s}_2, \ldots, \vec{s}_k$. We show that these solutions together constitute a feasible assignment to all variables $\vec{v}$ in $P_F$. The form of all the problems and the solutions mentioned above are described below and summarized in Figure 4.

The form of $P_k^{MAX}$ is as follows. Let $\overline{\mu_k}$ denote the optimal value of its objective function.

$$\begin{aligned} \overline{\mu_k} \ = \max_{\vec{y}_k} \quad & \mu_k \\ \text{subject to} \quad & \mu_k = u_k(\vec{y}_k) \\ & \mu_k > L_k \\ & \vec{g}_k\left(\vec{y}_k\right) \ \leq\ 0 \\ & \vec{h}_k\left(\vec{y}_k\right) \ =\ 0 \end{aligned} \tag{2}$$

The form of $P_k^{MIN}$ is similar and is shown below.

$$\begin{aligned} \underline{\mu_k} \ = \min_{\vec{y}_k} \quad & \mu_k \\ \text{subject to} \quad & \mu_k = u_k(\vec{y}_k) \\ & \mu_k > L_k \\ & \vec{g}_k\left(\vec{y}_k\right) \ \leq\ 0 \\ & \vec{h}_k\left(\vec{y}_k\right) \ =\ 0 \end{aligned} \tag{3}$$

For every $CB_i, i = k-1, k-2, \cdots, 2$, we let $P_i^{MAX}$ denote the maximization problem:

$$\begin{aligned} \overline{\mu_i} \ = \max_{\vec{y}_i,\ \mu_{i+1}} \quad & \mu_i \\ \text{subject to} \quad & \mu_i = u_i(\vec{y}_i,\ \mu_{i+1}) \\ & \mu_i > L_i \\ & \underline{\mu_{i+1}} \leq \mu_{i+1} \leq \overline{\mu_{i+1}} \\ & \vec{g}_i\left(\vec{y}_i\right) \ \leq\ 0 \\ & \vec{h}_i\left(\vec{y}_i\right) \ =\ 0 \end{aligned} \tag{4}$$

For every $CB_i, i = k-1, k-2, \cdots, 2$, we let $P_i^{MIN}$ denote

the minimization problem:

$$\begin{aligned} \underline{\mu_i} \;=\; \min_{\vec{y}_i,\ \mu_{i+1}} \quad & \mu_i \\ \text{subject to} \quad & \mu_i = u_i(\vec{y}_i,\ \mu_{i+1}) \\ & \mu_i > L_i \\ & \underline{\mu_{i+1}} \le \mu_{i+1} \le \overline{\mu_{i+1}} \\ & \vec{g}_i\,(\vec{y}_i) \;\le\; 0 \\ & \vec{h}_i\,(\vec{y}_i) \;=\; 0 \end{aligned} \tag{5}$$

The form of $P_1$ is:

*Find a feasible solution*

$$\begin{aligned} \text{subject to} \quad & \mu_1 = u_1(\vec{y}_1, \mu_2) \\ & \mu_1 > L_1 \\ & \underline{\mu_2} \le \mu_2 \le \overline{\mu_2} \\ & \vec{g}_1\,(\vec{y}_1) \;\le\; 0 \\ & \vec{h}_1\,(\vec{y}_1) \;=\; 0 \end{aligned} \tag{6}$$

The form of $P_i, i = 2, 3, \ldots, k-1$ is provided below. $\mu_i^*$ is the value selected in the solution of $P_{i-1}$ as shown in Equation (9).

*Find a feasible solution*

$$\begin{aligned} \text{subject to} \quad & u_i(\vec{y}_i, \mu_{i+1}) = \mu_i^* \\ & \mu_i > L_i \\ & \underline{\mu_{i+1}} \le \mu_{i+1} \le \overline{\mu_{i+1}} \\ & \vec{g}_i\,(\vec{y}_i) \;\le\; 0 \\ & \vec{h}_i\,(\vec{y}_i) \;=\; 0 \end{aligned} \tag{7}$$

The form of $P_k$ is:

*Find a feasible solution*

$$\begin{aligned} \text{subject to} \quad & u_k(\vec{y_k}) = \mu_k^* \\ & \mu_k > L_k \\ & \vec{g}_k\,(\vec{y}_k) \;\le\; 0 \\ & \vec{h}_k\,(\vec{y}_k) \;=\; 0 \end{aligned} \tag{8}$$

Solutions

$$\vec{s_i} = (\vec{y_i^*}, \mu_{i+1}^*), i = 1, 2, \ldots, k-1 \tag{9}$$

denote a feasible assignment to the respective variables. $\vec{s}_k$ is given by:

$$\vec{s}_k = (\vec{y_k^*}) \tag{10}$$

**Claim.** *When a feasibility problem has $k$ blocks that are chained as shown in $P_F$, we can unchain the blocks and solve for solutions $\vec{s}_1, \vec{s}_2, \ldots, \vec{s}_k$ as described above. These solutions together constitute feasible values for all the variables ($\vec{v}$) in $P_F$.*

*Proof:* To prove our claim, we need to show the following: (a) the value selected for every variable satisfies all the constraints related to that variable in $P_F$ and hence is feasible; (b) $\vec{s}_1, \vec{s}_2, \ldots, \vec{s}_k$ together constitute feasible values for all the variables ($\vec{v}$) in $P_F$. We prove these by induction.

**Basis**: When $k = 1$ (i.e.) there is only one block $CB_1$, $\vec{y}_1$ are the only variables in $P_F$. $P_F$ takes the form of

*Find a feasible solution*

$$\begin{aligned} \text{subject to} \quad & \mu_1 = u_1(\vec{y}_1) \\ & \mu_1 > L_1 \\ & \vec{g}_1\,(\vec{y}_1) \;\le\; 0 \\ & \vec{h}_1\,(\vec{y}_1) \;=\; 0 \end{aligned} \tag{11}$$

In this case, there is no decomposition to be done; $P_F$ becomes $P_1$ and its solution, denoted by $\vec{s}_1$, represents a feasible assignment to $\vec{y}_1$ (denoted by $\vec{y}_1^*$). $\vec{s}_1$ is feasible because $P_1$ satisfies all the constraints related to its variables in $P_F$. $\vec{s}_1$ also constitutes a solution to all the variables in $P_F$ because $P_1$ is the same as $P_F$.

**Inductive step**: Assuming our claim is true when $b$ blocks are chained, we show that it holds true when $b+1$ blocks are chained.

When there are $b$ blocks, the corresponding variables in $P_F$ are $\vec{v} = (\vec{y}_1, \vec{y}_2, \ldots, \vec{y}_b)$. Solutions to the $b$ blocks are $\vec{s}_1, \vec{s}_2, \ldots, \vec{s}_b$. By our inductive hypothesis, these solutions are feasible and when combined together constitute feasible assignments to all the variables in $P_F$. In particular, the solutions $\vec{s}_{b-1}$ and $\vec{s}_b$ are $(\vec{y}_{b-1}^*, \mu_b^*)$ and $(\vec{y}_b^*)$ respectively.

When there are $b+1$ blocks, the corresponding variables in $P_F$ are $\vec{v} = (\vec{y}_1, \vec{y}_2, \ldots, \vec{y}_b, \vec{y}_{b+1})$. By our inductive hypothesis, we already have feasible values for all the variables with the exception of $\vec{y}_{b+1}$. We can determine values for these variables by formulating and solving $P_{b+1}^{MAX}$ and $P_{b+1}^{MIN}$ and using their solutions to bound $\mu_{b+1}$ in $P_b^{MAX}$ and $P_b^{MIN}$. We then formulate $P_b$ and $P_{b+1}$ whose solutions are $\vec{s_b} = (\vec{y_b}^*, \mu_{b+1}^*)$ and $\vec{s}_{b+1} = (\vec{y}_{b+1}^*)$ respectively. These assignments are feasible because $\vec{s}_b$ and $\vec{s}_{b+1}$ satisfy all the constraints related to the involved variables $(\vec{y}_b, \vec{y}_{b+1})$ in $P_F$. Thus, $\vec{s}_b$ and $\vec{s}_{b+1}$ together constitute feasible assignments to all the new variables viz. $\vec{y}_{b+1}$. ■

*Reduction in the number of branch evaluations*: Thanks to our decomposition technique, when there are chaining variables the number of subproblems to solve is linear in the number of chained blocks (denoted by $k$) (i.e.) it is now $2(k-1) + k = 3k - 2$. Without the use of our technique, in the worst case (i.e.) complete enumeration, the number of configurations to be evaluated would be the product of the number of possible values of each variable in $\vec{v}$. For instance, if there are $k$ chained blocks with $n$ variables each, the number of configurations to search without the use of unchaining would be $n^k$. Use of unchaining, however, reduces the number to $k \times n$.

## V. BLASTN Application

BLAST, the Basic Local Alignment Search Tool, is the leading algorithm for searching genomic and proteomic sequence data. A variant, BLASTN search heuristic, compares a query string (composed from the alphabet $\{A, T, G, C\}$) to a

| Problem | Description | Variables | Solutions |
|---|---|---|---|
| $P_F$ | Unchaining of $k$ blocks starts here | $\vec{y_i}, i = 1, 2, \cdots, k$ | $\vec{y_i^*}, i = 1, 2, \cdots, k$ |
| $P_k^{MAX}$, $P_k^{MIN}$ | Optimization problems of $k$th block | $\vec{y_k}$ | $\overline{\mu_k}$, $\underline{\mu_k}$ |
| $P_i^{MAX}$, $P_i^{MIN}$ | Optimization problems of $i$th block, $i = k-1, k-2, \cdots, 2$ | $\vec{y_i}$ | $\overline{\mu_i}$, $\underline{\mu_i}$ |
| $P_1$ | Feasibility problem | $\vec{y_1}, \mu_2$ | $\vec{s_1} = (\vec{y_1^*}, \mu_2^*)$ |
| $P_i$ | Feasibility problem of $i$th block, $i = 2, 3, \ldots, k-1$ | $\vec{y_i}, \mu_{i+1}$ | $\vec{s_i} = (\vec{y_i^*}, \mu_{i+1}^*)$ |
| $P_k$ | Feasibility problem | $\vec{y_k}$ | $\vec{s_k} = (\vec{y_k^*})$ |

Fig. 4: Summary of notation used in unchaining.

| Variable | Symbol | Constraints |
|---|---|---|
| Clock freq. (stages 1a, 1b, 2) (14 values each) | $f_{1a}$,$f_{1b}$,$f_2$ | $10 \leq f \leq 133.3$ MHz |
| Processor cores (stage 3) | $c \in \mathbb{Z}_+$ | $1 \leq c \leq 4$ |
| Bloom filter hash functions | $h \in \mathbb{Z}_+$ | $2 \leq h \leq 10$ |
| Bloom filter memory size | $m$ | $1000, 1004, \cdots, 2000$ |
| Query length | $q$ | $40k, 41k, \cdots, 65k$ |
| Word size | $w \in \mathbb{Z}_+$ | $10 \leq w \leq 13$ |
| Buffer size | $b_i \in \mathbb{Z}_+$ | $2 \leq b_i \leq 16$, $1 \leq i \leq r$ |
| Reduction tree size | $r$ | 5 or 6 |
| Stage 2 threshold (10 values) | $p_2$ | $10^{-8} \leq p_2 \leq .005$ |
| Input arrival rate (100 values) | $\lambda_{in} \in \mathbb{R}_+$ | By solving (16) |

Fig. 5: Design variables for BLASTN [18].

genomic database, looking for biologically significant approximate matches. Buhler et al. [4] describe a hardware accelerated implementation of BLASTN. In this implementation, after the loading of a query string, the database is streamed in from the left of the figure. The database flows across the PCI-X bus into a field-programmable gate array (FPGA) and enters stage 1a which is decomposed internally into substages 1a1 to 1a6. Database entries that match in the Bloom filter flow to stage 1b, where they are checked against a hash table built from the query string. If found in the hash table, hits flow downstream to stage 2, where they are filtered and hits that survive stage 2 filtering are moved back across the PCI-X bus to stage 3 which is executed in software on the processor.

For this implementation, we presented a queueing network model earlier [18] which extends the model developed and validated by Dor et al. [7]. In this extension, we model the substages of 1a as individual service centers so as to model explicitly the bounded queue sizes within stage 1a.

The set of design variables (corresponding to the design parameters), along with their descriptions and constraints, are shown in Figure 5. The topology of the queueing network model is shown in Figure 6.

Service rates of all the service centers implemented in hardware (stages 1a through stage 2) are proportional to the corresponding clock frequencies. The bounded buffer sizes within stage 1a imply the mean service rate of an upstream node is scaled by the queue occupancy of its immediate downstream node as expressed below. Note that substage 1a6 is not affected.

$$\mu_{1ai} = \left(1 - \left(\frac{\lambda_{1a_{i+1}}}{\mu_{1ai+1}}\right)^{b_{i+1}}\right) \cdot f_{1a},\ i = 1, 2, \cdots, 5 \quad (12)$$

Service rates of the remaining stages are shown below where $\mathcal{F}_{1a_6}$, $\mathcal{F}_{1b}$, and $\mathcal{F}_2$ are constants.

$$\begin{aligned} \mu_{1a_6} &= \mathcal{F}_{1a_6} \cdot f_{1a} \\ \mu_{1b} &= \mathcal{F}_{1b} \cdot f_{1b} \\ \mu_2 &= \mathcal{F}_2 \cdot f_2 \end{aligned} \quad (13)$$

Service rate of stage 3 implemented in software is proportional to the number of processor cores denoted by $c$, with $\mathcal{C}$ denoting a constant:

$$\mu_3 = \mathcal{C} \cdot c \quad (14)$$

$p_{1a}$ and $p_{1b}$ below represent the probability that stages 1a and 1b respectively pass an input to their downstream neighbor. $dCG$, $qCG$, $dAT$, and $qAT$ are constants that reflect the fraction of $A$, $T$, $C$, and $G$ characters in the database and the query.

$$\begin{aligned} T &= \frac{dCG \cdot qCG + dAT \cdot qAT}{2} \\ p_1 &= 1 - (1 - T^w)^q \\ p_F &= (1 - e^{-w\frac{h}{m}})^h \\ p_{1a} &= p_1 + p_F \\ p_{1b} &= \frac{p_1}{p_{1a}} \end{aligned} \quad (15)$$

The relationship between the arrival rates at each stage are given by:

$$\begin{aligned} \lambda_{PCI} &= \lambda_{in} + \lambda_3 \\ \lambda_{1a1} &= \lambda_{in} \\ \lambda_{1a2} &= 2 \cdot p_{1a} \cdot \lambda_{1a1} \\ \lambda_{1a,j} &= 2 \cdot \lambda_{1a,j-1} \quad for \quad 3 \leq j \leq r \\ \lambda_{1b} &= \lambda_{1a,r} \\ \lambda_2 &= p_{1b} \cdot \lambda_{1b} \\ \lambda_3 &= p_2 \cdot \lambda_2 \end{aligned} \quad (16)$$

Given recent technology trends, power has become of increasing concern to system designers. As a result, there many circumstances under which balancing the desire to reduce power consumption with the desire to increase data throughput is an explicit goal in the development of streaming applications. The BLASTN is an example of just such a circumstance. The overall completion time is inversely related to the data throughput (i.e., the database ingest rate on the left of the figure) and the total energy required is the product of completion time and power consumption. As a result, power and throughput are reasonable choices for a multi-objective

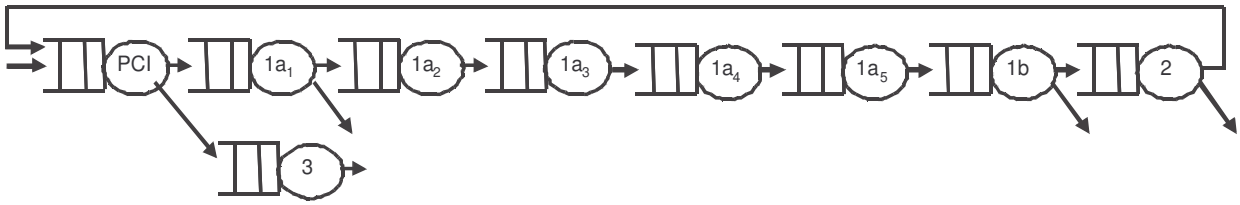


Fig. 6: Queueing network-based performance model of BLASTN application [18].

optimization cost function. The FPGA power is modeled as a combination of dynamic power (linearly related to clock frequency) and static power (a constant, independent of clock frequency), resulting in a power equation of the form shown below. Values for $m_{1a}$, $m_{1b}$, $m_2$, and $\wp_{static}$ are derived from Xilinx Power Estimator, ISE v11.5i [25].

$$\wp = m_{1a} \cdot f_{1a} + m_{1b} \cdot f_{1b} + m_2 \cdot f_2 + \wp_{static} \tag{17}$$

We wish to optimize the combination of throughput (measured by $\lambda_{in}$) and FPGA power consumption. The resulting cost function is a weighted sum (a standard technique [17]) of the individual optimization goals, i.e.,

$$minimize \quad W_1 \cdot \wp - W_2 \cdot \lambda_{in} \tag{18}$$

where $W_1$ and $W_2$ encode both the weights and normalization factors. Equations in Figure 5 and Equations 12 through 16 constrain the objective function (Equation 18) in our optimization problem formulation of BLASTN.

## VI. Empirical Results

A complete enumeration of the search space requires $6 \times 10^{10}$ configurations of the complicating variables $f_{1a}$, $h$, $m$, $q$, $w$, $r$, $p_2$, and $\lambda_{in}$. For the variables that correspond to substages of 1a, each of 5 input variables $b_i$ has 15 discrete values, resulting in $15^5 = 759,375$ configurations. Finally, stages 1b, 2, and 3 comprise $14 \times 14 \times 5 = 980$ configurations (choosing $f_{1b}$, $f_2$, and $c$). Overall, this results in $6 \times 10^{10} \times 759,375 \times 980 \approx 4 \times 10^{18}$ configurations.

Figure 7 shows the variable-constraint matrix for the BLASTN application when $\vec{\mu}$ chains blocks representing substages of 1a. Each column corresponds to an individual variable in the problem formulation, and each row corresponds to an individual constraint. The last row represents the objective function to be optimized. The matrix is a Boolean-valued matrix, with a 1 in an entry if the variable associated with the column is present in the constraint associated with the row. Complicating variables are enumerated at the left-hand side of the matrix. Independent blocks in the matrix are indicated via the rectangular boxes at the far right-hand side of the figure, corresponding to the constraints of stages 1b, 2, and 3. Each block corresponds to an individual service center in the queueing network and using our earlier techniques [18], each block can be optimized independently. This reduces 980 configurations from stages 1b, 2, and 3 down to 33 which reduces the overall search size from $4 \times 10^{18}$ down to $6 \times 10^{10} \times 759,375 \times 33 \approx 10^{17}$.

Note that the majority of the middle columns comprise a set of chained blocks corresponding to the substages of 1a. The set of equations that relate the $\mu$'s associated with application stage 1a gives rise to the chain. This relationship is illustrated in Equation (12). During branch and bound, we branch on all the other variables (starting with the complicating variables) before branching on the variables of the chained blocks.

Overall, the intuitive notion one gets from observing this matrix is that if the structure exposed in the matrix can be exploited during the design space search process, clear benefits will accrue. Note that the objective function row in the variable-constraint matrix does not have entries in the columns associated with stage 1a. This implies that with a suitable ordering of branching variables, it is plausible to have fixed the value of the objective function by the time the variables associated with stage 1a are being chosen.

Exploiting the chaining structure results in significant benefits. Unchaining allows the $15^5$ configurations associated with stage 1a to be reduced to $15 \times 5 \times 2 = 150$ configurations. The multiple of 2 is due to the fact that at each chaining step one must solve for both a minimum and a maximum value of the chaining variable. This reduces the overall search size down to $6 \times 10^{10} \times 150 \times 33 \approx 3 \times 10^{13}$ configurations. This represents 5 orders of magnitude reduction in the overall search size relative to complete enumeration.

*a) Optimal configuration:* The objective function value of the global optimum is 62.6 which corresponds to the configuration of Figure 8. The properties of this configuration are consistent with what is known about the system that has been physically constructed [4]. Stage 1a is the bottleneck stage, and the configuration of Figure 8 gives it a high clock frequency. The latter stages have much less work to perform, making their lower clock frequency beneficial. Note that the implementation in [4] made no attempt to simultaneously lower power consumption, so we would not expect the two configurations to be identical.

## VII. Conclusion

This paper has introduced the technique of unchaining to reduce the search space complexity of streaming applications. When $\vec{\mu}$ variables chain some of the service centers in the queueing network, we demonstrated how we can perform unchaining and still preserve feasibility. For BLASTN, we showed that unchaining reduces the search size from $10^{17}$ to $10^{13}$ configurations. The reduced space can be searched in less than 3 hours on 1000 processor cores if each cost

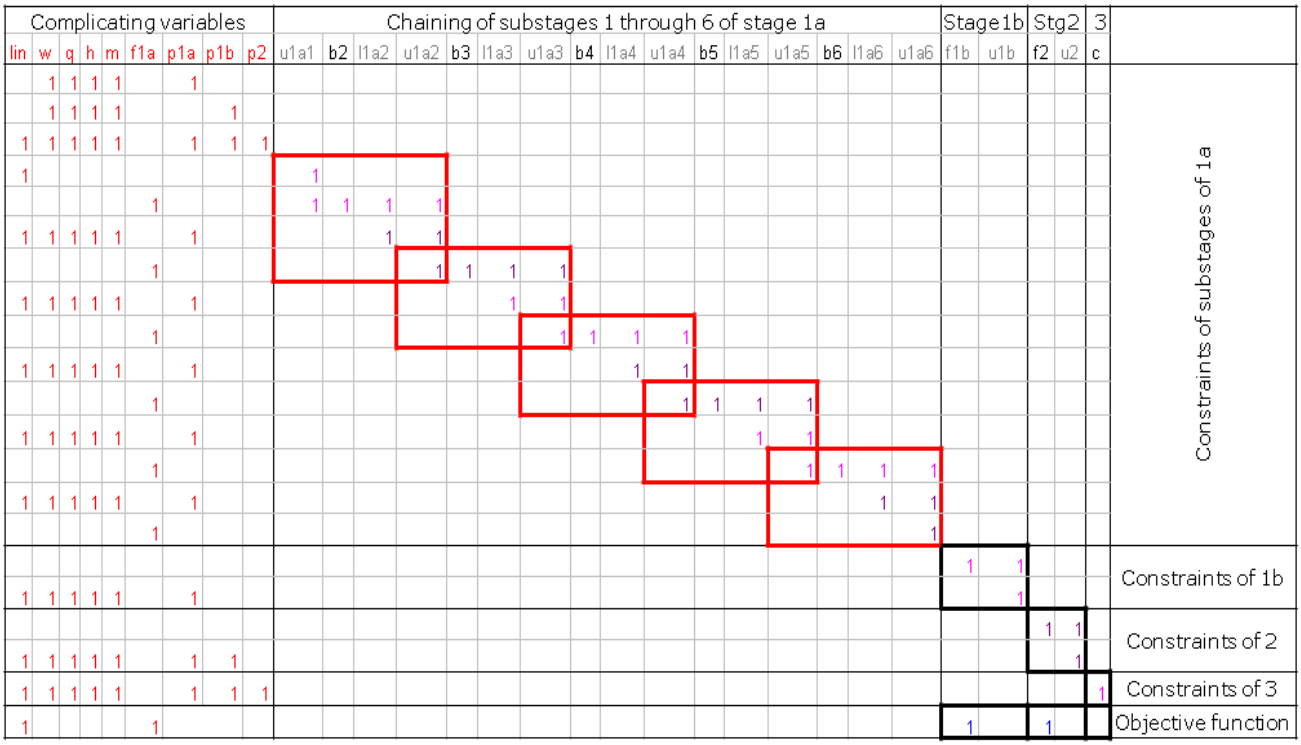


Fig. 7: BLASTN variable-constraint matrix when $\vec{\mu}$ chains blocks associated with stage 1.

| Variable | Value in the solution configuration |
|---|---|
| $r$ | 5 |
| $\lambda_{in}$ | 132 MBases/sec |
| $h$ | 3 |
| $q$ | 40,000 bases |
| $w$ | 10 bases |
| $m$ | 1000 |
| $p_2$ | $4.9 \times 10^{-8}$ |
| $f_{1a}$ | 133.3 MHz |
| $b_1$ | 3 |
| $b_2$ | 2 |
| $b_3$ | 12 |
| $b_4$ | 2 |
| $b_5$ | 2 |
| $f_{1b}$ | 10 MHz |
| $f_2$ | 10 MHz |
| $c$ | 4 |

Fig. 8: BLASTN: variable values in the solution.

evaluation takes 1 $\mu$s. Ongoing work includes the investigation of unchaining when the value of the objective function is not determined as well as nonlinear (e.g., tree-structured) unchaining. An example scenario where nonlinear chaining would occur is when two or more upstream nodes feed into a downstream node in a queueing network. Further, we are investigating whether our unchaining guarantees finding a feasible solution whenever it exists.

## REFERENCES


[1] S. F. Altschul, W. Gish, W. Miller, E. W. Myers *et al.*, "Basic local alignment search tool," *J. of Molecular Biology*, vol. 215, pp. 403–10, 1990.

[2] 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.

[3] 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.

[4] J. D. Buhler, J. M. Lancaster, A. C. Jacob, and R. D. Chamberlain, "Mercury BLASTN: Faster DNA Sequence Comparison Using a Streaming Hardware Architecture," in *Proc. of Reconfigurable Systems Summer Institute*, Jul. 2007.

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

[6] 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.

[7] R. Dor *et al.*, "Using queuing theory to model streaming applications," in *Symp. on Application Accelerators in High Perf. Computing*, 2010.

[8] M. A. Duran and I. E. Grossmann, "An outer approximation algorithm for a class of mixed-integer nonlinear programs," *Mathematical Programming*, vol. 36, pp. 307–339, 1986.

[9] M. A. Franklin, E. Tyson, J. Buckley, P. Crowley, and J. Machmeyer, "Auto-pipe and the *X* language: A pipeline design tool and description language," in *Proc. of Int'l Parallel and Distributed Processing Symp.*, 2006.

[10] B. Gedik, H. Andrade, K. Wu, P. Yu, and M. Doo, "SPADE: The System S declarative stream processing engine," in *Proc. of ACM SIGMOD Int'l Conf. on Management of Data*, 2008, pp. 1123–1134.

[11] A. M. Geoffrion, "Generalized benders decomposition," *J. Optim. Theory and Appl.*, vol. 10, no. 4, pp. 237–241, 1972.

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

[13] K. Holmberg, "On the convergence of the cross decomposition," *Mathematical Programming*, vol. 47, pp. 269–316, 1990.

[14] 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.

[15] B. C. Lee and D. Brooks, "Roughness of microarchitectural design topologies and its implications for optimization," in *Proc. of IEEE Int'l Symp. on High Performance Computer Architecture*, 2008, pp. 240–251.

[16] E. A. Lee, "The problem with threads," *Computer*, vol. 39, no. 5, pp. 33–42, May 2006.

[17] R. Marler and J. Arora, "Survey of multi-objective optimization methods for engineering," *Structural and Multidisciplinary Optimization*, vol. 26, no. 6, pp. 369–395, 2004.

[18] S. Padmanabhan, Y. Chen, and R. D. Chamberlain, "Optimal design-space exploration of streaming applications," in *Int'l Conf. on Application-specific Systems, Architectures and Processors*, Sep. 2011.

[19] H. Perros and T. Altiok, "Approximate analysis of open networks of queues with blocking: Tandem configurations," *IEEE Trans. Soft. Eng.*, vol. 12, pp. 450–461, 1986.

[20] 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.

[21] N. V. Sahinidis, "Baron: A general purpose global optimization software package," *J. of Global Optimization*, vol. 8, no. 2, pp. 201–205, 1996.

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

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

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

[25] Xilinx Inc., "Xilinx power estimator," www.xilinx.com/products/design_tools/logic_design/xpe.htm.