Design-space Optimization for Automatic Acceleration of Streaming Applications

Shobana Padmanabhan
Yixin Chen
Roger D. Chamberlain

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

Design-space Optimization for Automatic Acceleration 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*—When using architecturally diverse systems to accelerate streaming applications, the design search space is often complex. We present a global optimization framework comprising a novel domain-specific variation of branch-and-bound that reduces search complexity by exploiting topology of the application's pipelining.**

I. Introduction

High performance streaming applications are *pipelined* asynchronously and frequently deployed on architecturally *diverse* systems [4], [5]. Typically, there are several *design parameters* (examples in Section III) in the algorithms and architectures used that when customized impact the tradeoff between application performance and resource usage. Searching the design space of possible configurations is hard because: (1) the number of configurations is exponential in the number of design parameters, (2) the design parameters may interact nonlinearly, and (3) goals of the design-space exploration are often multiple and conflicting. Though this problem has been researched for system-on-chip applications, to our knowledge, we are the first to focus on it for streaming applications.

We approach design space exploration as an optimization problem. Cost functions are derived increasingly using *models* rather than through direct execution or simulation. We support BCMP queueing network (QN) models because they embody the queueing and topology of our application's pipelining (e.g., see [3]) better than general-purpose models based on regression or machine-learning [7]. We derive our objective function using the standard technique of weighted sum of normalized cost functions; constraints may stem from the design space, QN models, resource availability, performance requirements, and the optimization problem formulation itself.

Such optimization problems tend to be NP-hard because: (1) most design parameters are integer-valued, (2) QN expressions are nonlinear, and (3) the nonlinear functions are not convex, differentiable, or even continuous. Worse, state-of-the-art solvers such as Bonmin or FilMINT [8] often fail to find even a *feasible* solution. Hence, we have developed a search framework, based on *branch and bound* (B&B) principle, that systematically finds the global optimum given adequate time. The efficiency of B&B depends on how the branching variables (BVs) are selected and how many branches are evaluated. To help us with these, we exploit the topology of the application pipelining.

We thank NSF (grants CNS-0905368 and CNS-0931693), the authors and user groups of AMPL, FilMINT, RCS package for R, You Xu of Washington University, and the referees of the symposium.

Our contributions are: (1) We identify and **categorize** domain-specific topological information. (2) Using the categories, we develop a heuristic that **orders** the BVs such that each branching leads to maximum decomposition of the search space. We introduce a way to identify decomposability of our applications. (3) To improve our search efficiency further, we originate a branching principle that we name **convex branching**, which when present obviates evaluating every value of the BV. (4) Our framework supports **anytime** solutions if application developers need a suboptimal solution fast. (5) We illustrate the application and benefit of our framework using a common and prototypical real-world streaming application.

II. Domain-specific optimization

Our optimization problems are highly nonlinear and mostly discrete and hence none of the standard *relaxation* techniques [2] work, as we verified during our experiments. This implies we cannot bound and hence cannot prune branches. Hence, efficiency of our search depends on the ordering of BVs. Branching on a variable *decomposes* the search space so that the resulting *subproblems* are easier. Therefore, we aim to order the BVs such that each branching leads to maximum decomposition of the search space.

A. Domain-specific topological info

To identify decomposability from variables, we check if a matrix showing the presence of variables in the constraints and the objective function has the canonical *Jordan block* form. In this form, the objective function and the constraints separate into blocks that can be solved in parallel *without* loosing optimality. A variable that prevents such decomposition is traditionally called a **complicating variable (CV)** (Figure 1). We define *degree of complication* as the number of constraints a CV appears in so that the *most complicating variable* has the highest $degree$. For our applications, we group variables and constraints concerning a single queueing station (QS) as a Jordan block. Accordingly, in the categorization below, variables in top are the most complicating, followed by those in mq, whereas sq variables are not complicating. If $\lambda_{in} \in var$, by **l**, λ_{in} has the highest degree of complication in mq.

$var = \{var_i | var_i \in \mathbb{R}_+ \cup \{0\}\}$ is the set of **design** variables in the optimization problem that correspond to the user-identified design parameters. $n_v \doteq |var|$ and $\mathbf{v} \in (\mathbb{R}_+ \cup \{0\})^{n_v}$

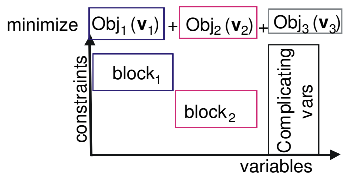

Fig. 1: Jordan block form with complicating vars.

is the vector of the variables in var. $var = top + mq + sq$. $top \subseteq var$ is the set of **topological** variables that result in distinct *alternative* QN topologies. $n_t \doteq |top|$ and **t** is the vector formed by the variables. We discovered the presence of top during our experimentation (i.e.) they are not traditional. $mq \subseteq var$ is the set of **Multi-QS** (MQ) variables, with each element in the domain of more than one $u_j(\cdot)$. $sq \subseteq var$ is the set of **Single-QS** (SQ) vars. Each element of sq is in the domain of only one $u_j(\cdot)$.

$der = \{der_i | der_i \in \mathbb{R}_+ \cup \{0\}\}$ is the set of **derived** variables that depend on one or more elements in var. $der \bigcap var = \emptyset$. $met \subseteq der$ is the set of performance **metrics**. $n_m \doteq |met|$ and **met** is the vector formed by the variables in *met*. $met_k = o_k(\mathbf{v}) : (\mathbb{R}_+ \cup \{0\})^{n_v} \rightarrow \mathbb{R}_+ \cup \{0\}$ where **o** is the vector of functions that define **met**. $z \in der$ is the **cost function** (the **objective** function). $z = \sum_{k=1}^{n_m} W_k \times met_k : (\mathbb{R}_+ \cup \{0\})^{n_m} \rightarrow \mathbb{R}_+, \sum_{k=1}^{n_m} W_k = 1$, W_k are the weights.

$ivar \subseteq der$ is the set of **intermediary** variables (IVs). IVs may arise because: (1) application developers are interested in them (e.g., for debugging) (2) to codify abstractions such as QNs in the performance models and (3) to help optimization solvers (cutting-plane based solvers such as FilMINT [1] tend to work better with linear cost functions while solvers using the interior-point algorithm work better with linear constraints). $mu \subseteq ivar$ is the set of mean *service* rates at each QS. μ is the vector formed by the variables in *mu*. $\mu_j = u_j(\mathbf{v}) : (\mathbb{Z}_+ \cup \{0\})^{n_v} \rightarrow \mathbb{R}_+$ where **u** is the vector of functions that define μ. $lam \subseteq ivar$ is the set of mean *job arrival* rates at each QS. λ is the vector formed by the variables in *lam*. λ are related by a system of linear equations with a unique solution (in terms of input mean job arrival rate denoted by $\lambda_{in} \in \mathbb{R}_+$). $\lambda_j = l_j(\lambda_{in}, \mathbf{t}) : \mathbb{R}_+ \cup \{0\} \times \mathbb{Z}_+^{n_t} \rightarrow \mathbb{R}_+ \cup \{0\}$ where **l** is the vector of functions that define λ. **ul** is a vector of *QN constraints* that restricts every $\lambda_j < \mu_j$ for the system to be *stable*.

Topologies of our streaming applications are restricted to be directed acyclic graphs (DAGs) while QN topologies are annotated digraphs. Annotations include, at the minimum, expressions for each of **u**, **l**, **o**, and z; each node represents a QS which is a service facility with its queue and each edge, the communication link between the two connected nodes. We identify the general form of our optimization problem to be the following. **g** and **h** are vectors of the general constraints.

$$\begin{aligned} \min_{\mathbf{v}} \quad & z = \Sigma_{k=1}^{n_m} W_k \times o_k(\mathbf{v}), \Sigma_{k=1}^{n_m} W_k = 1 \\ \textit{subject to} \quad & \mu = \mathbf{u}(\mathbf{v}), \quad \lambda = \mathbf{l}(\lambda_{in}, \mathbf{t}), \quad \mathbf{ul}(\lambda, \mu), \\ & \mathbf{h}(\mathbf{v}) = 0, \quad \text{and} \quad \mathbf{g}(\mathbf{v}) \leq 0 \end{aligned}$$

B. Domain-specific branch & bound

We order the BVs by decreasing degree of complication, first across variable categories and then within a category. We break ties within a category in favor of the variable with the largest domain. The resulting ordering is: (1) Branch on top variables. *After branching on all top variables, each branch will evaluate only one topology*, by definition of top. (2) Branch on λ_{in} if it is in var. (3) Branch on the remaining mq variables. *After branching on all mq, each subproblem concerns only one queueing station* because $var - top - mq = sq$. (4) If the subproblems are still unsolvable (e.g., u is not convex), branch on sq variables. *The solution after branching on all of sq is the global optimum* since $var = top + mq + sq$.

Based on our experiments, we discovered a **convexity** property which when present obviates evaluating every value of a BV, making it more efficient than exhaustive searching: *if the objective function of the branching variable is convex, we can solve for that variable analytically* (by setting the first derivative of z w.r.t. the BV = 0). We name this branching **convex branching** and the branching variable **con-BV**. For the QN topologies shown in Section III and for the common performance metrics of application latency and throughout, $z(\lambda_{in})$ is convex and hence λ_{in} is a con-BV.

Our search procedure is: (1) choose and branch on the next BV; (2) solve the resulting subproblems (exploiting the convexity property if present) and update the incumbent solution (initialized to ∞ for a minimization problem); and (3) terminate if applicable or go back to (1). We compare the incumbent solution against the objective function value in every subproblem resulting from the current branching. If a subproblem is *convex* and its solution is worse than the incumbent solution, we stop branching on that subproblem. If every subproblem from a branching is convex, we stop branching, and the incumbent solution, updated as applicable, is the global optimum. A termination condition may be to stop with a feasible (suboptimal) solution.

III. Empirical results

We illustrate our search procedure using the the well-known class of **data-decomposing** applications in which incoming data is divided, processed in parallel for accelerated application performance and the results combined (Figure 2). Streaming sort is one such application and is part of many real-world high-performance applications [4], [6]. We refer to computational elements as *blocks*, communication channels as *links*, and all elements at a given level of the application DAG collectively as a *column*. We use the FilMINT optimization solver as it best handles formulations such as ours [1].

QN expressions are provided (and validated) by application developers. Below is a sample for illustrative purposes from a deployment on our Auto-pipe platform [4]. Every column in Figure 2 is modeled as a queueing station (Figure 3). If the elements in a column are not identical, effect of the dominant element w.r.t. the cost function is modeled.

var includes N (2^N = number of parallel processing blocks), four mappings ($map_0, \ldots$), number of resources per

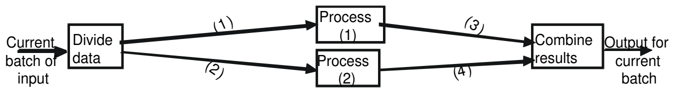

Fig. 2: Data-decomposing streaming applications.

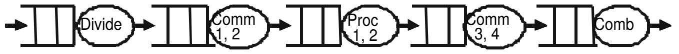

Fig. 3: A queueing model for the topology in Figure 2.

column ($nRes_j$, j denotes the column), resource type per column (e.g., processor vs. FPGA: sw_j, hw_j, ...), λ_{in}, algorithmic choices, and communication message size (msg). An example mapping is map_3 when links into and out of the process blocks are shared on one channel. Variables are constrained to yield a feasible solution. $top = \{N, map_0, \dots\}$. As N increases, the number of divide and combine columns also increases, increasing the length of the tandem QN. map_3 transforms the tandem QN (Figure 3) to Figure 4. $mq = \{\lambda_{in}$, msg, sw_j, $hw_j\}$. msg is set system wide. The resource types of links are selected according to that of the connected blocks.

$met = \{Latency, Throughput\}$. For e.g.:
$Latency = map_0 \left(\sum_{j=0}^{4N} \frac{1}{\mu_j - \lambda_j} \right) + map_1(\cdot) + \dots.$
$Throughput = 1/\lambda_{in}$.

$z = 0.5(Latency) + 0.5(1/Throughput)$.

$ivar = \{\mu_j, \lambda_j\}$. **u** may be derived based on first principles or regression-based curve fitting on a sample of experimental observations. As an example, for divide blocks:

$$\begin{aligned} \mu_j &= [(map_1 + map_2)(sw_0 \cdot C_1 + hw_0 \cdot C_2) \\ &\quad +(map_0 + map_3)(sw_j \cdot C_2 + hw_j \cdot C_2) \\ &\quad] \times nRes_j \times (2^{j/2}/2^{NUM\_ELEMENTS}) \end{aligned}$$

C_1, C_2 are base rates calibrated using sample empirical observations or known results; the fraction denotes the job size (number of elements) processed by each block. **l** is derived as: for tandem QNs, each $\lambda_j = \lambda_{in}$; for the QN with branching, $\lambda_{2N-1} = 2\lambda_{in}, \lambda_{2N+1} = 0$, each of the other $\lambda_j = \lambda_{in}$.

For the resulting optimization problem, none of the state-of-the-art solvers find even a feasible solution. Hence, we applied our search procedure by branching on $N \in top$, then on the $map \in top$, and then on λ_{in} (Figure 5). For this application, every element of met moves monotonically with λ_{in} and hence $z(\lambda_{in})$ is **convex**. FilMINT finds a feasible solution for every subproblem resulting from branching on λ_{in}. The solution after these branchings is $z = 0.3$ ms and the corresponding configuration is: 8 processing blocks, map_3, $\lambda_{in} = 143$ 64-bit Melements/s, the fastest and maximum number of resources for every column, and $msg = 4$. Using our application knowledge and by neighborhood search, we found that the selections for $mapping$ and msg are not optimum even locally. To find a local optimum, we need to continue branching on the remaining mq variables and, if needed, on sq variables.

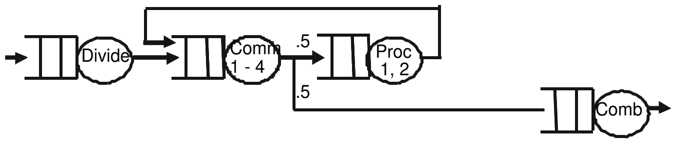

Fig. 4: Queueing network with branching.

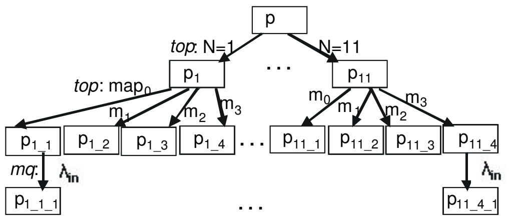

Fig. 5: Branching for our example application.

We verified that the solution from our framework matches the result from searching exhaustively when $N \leq 2$ (takes months beyond that). The overhead due to decomposition is reasonable because the solver runtime is in milliseconds and gets progressively lower with the subproblems. We verified sensitivity of our solution by varying the weights on met. To evaluate the effectiveness of our framework when the original problem can be solved by the solver, we relaxed some highly nonlinear constraints on resource type selection of links and obtained 2-3 orders of magnitude improvement in the cost function value, from 10 s to 2 ms.

IV. Ongoing work

Our investigation includes: (1) Can we generalize that top variables should always be considered first in B&B solvers? (2) Does any connected graph with the met of latency and throughput facilitate our convex branching? (3) How do we handle including power consumption in the objective function, as it does not benefit from higher μ as does latency and throughput? (4) Application of our framework to other classes of real-world streaming applications, in particular, filtering (e.g., biosequence search) applications.

References

[1] K. Abhishek, S. Leyffer, and J. T. Linderoth, "FilMINT: An Outer-Approximation-Based Solver for Nonlinear Mixed Integer Programs," in *ANL/MCS-P1374-0906*, Mar. 2008.

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

[3] G. Casale, M. Ningfang, and E. Smirni, "Versatile models of systems using map queueing networks," in *IEEE Int'l Symp. on Parallel and Distributed Processing*, Apr. 2008.

[4] R. Chamberlain *et al.*, "Auto-Pipe: Streaming applications on architecturally diverse systems," *Computer*, vol. 43, no. 3, pp. 42–49, 2010.

[5] W. J. Dally *et al.*, "Merrimac: Supercomputing with Streams," in *ACM/IEEE Supercomputing Conf.*, 2003.

[6] F. Kulla and P. Sanders, "Scalable Parallel Suffix Array Construction," *High Performance Computing in Science and Engg*, 2006, part 6.

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

[8] "Mixed Integer Nonlinearly Constrained Optimization Solvers," neos.mcs.anl.gov/neos/solvers/index.html.