# Optimal Design-Space Exploration of Streaming Applications

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

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

# Optimal Design-Space Exploration of Streaming Applications

Shobana Padmanabhan, Yixin Chen, and Roger D. Chamberlain

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

*Abstract*—**Many embedded and scientific applications are pipelined (i.e., streaming) and deployed on application-specific systems. Typically, there are several design parameters in the algorithms and architectures used that impact the tradeoff between different metrics of application performance as well as resource utilization. Efficient automatic exploration of this design space is the goal of our research.**

**We present a global optimization framework comprising a domain-specific variation of branch-and-bound that reduces search complexity by exploiting the topology of the application's pipelining. We exploit the topological information to discover decomposability through the canonical Jordan block form. The reduction in search complexity for four real-world streaming applications (drawn from the literature) is significant, ranging from a million-fold reduction in search space size to a reduction factor of 10 billion. All four optimization problems are thereby solvable in reasonable time.**

## I. INTRODUCTION

High performance streaming applications are frequently deployed on architecturally diverse systems employing chip multiprocessors, graphics engines, and reconfigurable logic. The number of options available to the designer increases 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.).

We approach design space exploration as an optimization problem in which we seek a globally optimal configuration. We exploit queueing network (QN) models [1] because they embody the queueing and topology of applications' pipelining. 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 (objective function or constraints) are typically not convex, differentiable, or even continuous. Worse, in practice, state-of-the-art solvers such as Bonmin or FilMINT [7] often fail to find even a feasible solution for this class of mixed-integer nonlinear problems (MINLP). We have developed a search framework, based on the branch and bound principle, that systematically finds the global optimum.

With standard branch and bound, efficient bounding helps to prune branches which makes the search more efficient. However, the optimization problems in the domain of streaming applications tend to be highly nonlinear (remaining non-convex through several levels of branching) and mostly discrete, so the standard relaxation techniques tend not to work. This implies we cannot bound and therefore cannot prune branches.
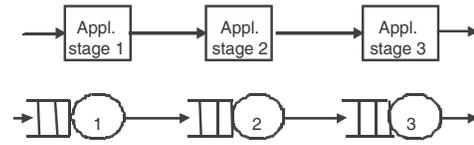
Fig. 1: Example streaming application and associated queueing network model.

In such cases, the search efficiency depends on the ordering of branching variables and identifying conditions that obviate having to evaluate every branch.

Consider the example of Figure 1. 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 [1]. In such a network, it is conventional to define $\lambda_j$ as the mean job arrival rate and $\mu_j$ as the mean service rate for station $j$. Here, we restrict ourselves to $M/M/1$ BCMP networks with infinite queues and FIFO queueing.

Analytic models for the service rates and network branching probabilities are assumed to come from the user. For the application of Figure 1, we assume that each $\mu_j$ corresponds to the clock frequency (denoted by $f_{CLKj}$) of the processor or the custom hardware to which it is mapped, scaled by a constant. For the tandem pipeline in our example application, each $\lambda_j = \lambda_{in}$. For the sake of simplicity, we assume that the only objective is to maximize throughput, given by $\lambda_{in}$.

In the context of an optimization problem, the topology of the queueing network can give us significant clues as to how to efficiently search the design space. In this paper, we describe techniques to do precisely that.

## II. DOMAIN-SPECIFIC OPTIMIZATION

Branching on a variable decomposes the search space, resulting in subproblems that are less complex. Our aim then is to order the branching variables such that the current branching leads to more decomposition of the search space than the subsequent one. We identify decomposability from variables by checking if a matrix showing the presence of variables in the constraints and the objective function has the canonical Jordan block form [9]. In this form, the objective function and the constraints separate into blocks that can be solved independent of each other without loosing optimality.

We identify Jordan block form using the BCMP equivalence property described above. The stations in a BCMP network

are related only through job arrival rates (denoted by $\lambda_j$ in the example model). In this case, we group variables and constraints associated with only a single queueing station as a Jordan block (JB). We call the variables concerning a single JB **Single-JB** variables, denoted by the set $sj$. There are 3 JBs for our example application and the corresponding Single-JB variables are $f_{CLK1}$, $f_{CLK2}$, and $f_{CLK3}$.

In real-world applications, there are usually variables that prevent such decomposition. Such variables are traditionally called complicating variables (CVs). Variables concerning more than a single queueing station would be CVs. We define the *degree of complication* as the number of queueing stations a CV concerns so that the *most complicating variable* has the highest degree. We refer to CVs that concern more than one queueing station as **Multi-JB** variables, denoted by the set $mj$. For the example application, $\lambda_{in}$ is in $mj$.

We observe that there are sometimes distinct alternative queueing network topologies for a single application. We call the complicating design variables that result in multiple topologies **topological** variables, denoted by the set $top$.

Sometimes there are variables that are expressed in terms of design variables. We call such variables **derived** variables. Examples of this kind are performance metrics and the value of the cost function. Sometimes there are abstractions introduced by queueing theory, such as $\mu_j$ and $\lambda_j$. We call such derived variables **intermediary** variables.

We denote the set of design variables in the optimization problem that correspond to the user-identified design parameters by $var$. Formal definitions of the variable categories as well as our formulation of the optimization problems are provided in [8].

*Heuristic to order branching variables*: We order the branching variables in the order of 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 $\lambda_{in}$ if it is in $var$. (3) Branch on the remaining $mj$ variables. After branching on all $mj$, each subproblem concerns only one queueing station because $var - top - mj = sj$. (4) If the subproblems are still unsolvable (e.g., not convex), branch on $sj$ variables. The solution after branching on all of $sj$ is the global optimum since $var = top \cup mj \cup sj$.

*Reduction in number of branch evaluations*: If there is more than one element in $sj$ per Jordan block, then their combinations inside a JB need to be evaluated. However, by our definition of $sj$ variables, $sj$ variables across JBs are independent of each other and hence their combinations need not be evaluated (i.e., the subproblems corresponding to the combinations need not be solved). This leads to a significant reduction in the number of branches that are to be evaluated. For example, consider $n_v$ variables with each variable having $k$ values, the total number of nodes in the branch and bound tree, including the root, is $\frac{1-k^{n_v+1}}{1-k}$. On the other hand, by our

definition of $sj$ variables, the complete enumeration need only happen for $n_t + n_{mj}$ variables ($n_v = n_t + n_{mj} + n_{sj}$, where $n_v = |var|$, $n_t = |top|$, $n_{mj} = |mj|$ and $n_{sj} = |sj|$). To count the number of branch evaluations involving $sj$ variables, let the number of distinct JBs be denoted by $J$ and let the variables in $sj$ be evenly divided among the JBs. Then, the number of variables in each JB is given by $n_{sj}/J$, and the total number of nodes in the reduced branch and bound tree is:

$$\frac{1-k^{n_t+n_{mj}+1}}{1-k} + k^{n_t+n_{mj}} \left( \frac{1-k^{n_{sj}/J+1}}{1-k} - 1 \right).$$

This represents a significant savings anytime that $J >> 1$.

*Search procedure*: (1) choose and branch on the next branching variable; (2) solve the resulting subproblems and update the incumbent solution (initialized to $\infty$ 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. The search procedure above supports anytime solutions, in which early termination can still yield one (or more) feasible configurations.

## III. EMPIRICAL RESULTS

### A. BLASTN

BLAST, the Basic Local Alignment Search Tool, is the leading algorithm for searching genomic and proteomic sequence data. BLASTN, the variant that focuses on genomic data, has been accelerated using special-purpose hardware [2]. BLASTN is an example of a data-filtering application. A queueing network-based performance model of Buhler et al.'s FPGA-accelerated BLASTN [2] was validated by Dor et al. [4]. The queueing network is illustrated in Figure 2. The set of design variables are enumerated in Figure 3. The goal is:

$$maximize \quad W_1 \times \lambda_{in} - W_2 \times P, \quad \sum_1^2 W_i = 1$$

where $W_1$ and $W_2$ encode both the weights and normalization factors. Throughput is given by $\lambda_{in}$. $P$ is the FPGA power consumption modeled in the traditional manner as a combination of dynamic power (linearly related to clock frequency) and static power (a constant, independent of clock frequency):

$$P = m_{1a} \cdot f_{1a} + m_{1b} \cdot f_{1b} + m_2 \cdot f_2 + P_{static}$$

where the values for $m_{1a}$, $m_{1b}$, $m_2$, and $P_{static}$ are derived from Xilinx Power Estimator, ISE v11.5i. Within stage 1a, which requires a bounded buffer size model, the mean service rate of an upstream node is scaled by the queue occupancy of the downstream node, e.g.,

$$\mu_{1a1} = \left( 1 - \left( \frac{\lambda_{1a2}}{\mu_{1a2}} \right)^{b_2} \right) \cdot f_{1a}$$
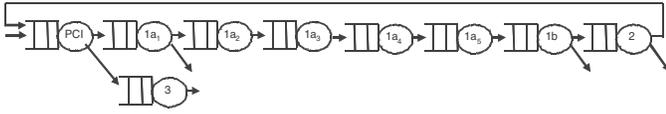
Fig. 2: Queueing network model of FPGA-accelerated BLASTN.



Fig. 4: Streaming sort application.

| Variable | Symbol | Ranges and Constraints | Category |
|---|---|---|---|
| Reduction tree size | $r$ | 5 or 6 | $top$ |
| Arrival rate | $\lambda_{in} \in \mathbb{R}_+$ | Each $\lambda_j < \mu_j$ | $mj$ |
| Bloom filter (BF) hash functions | $k \in \mathbb{Z}$ | $2 \leq k \leq 10$ | $mj$ |
| BF memory size | $m \in \mathbb{Z}$ | $1000 \leq m \leq 2000$ | $mj$ |
| Query length | $q \in \mathbb{Z}$ | $40,000 \leq q \leq 65,000$ | $mj$ |
| Word size | $w \in \mathbb{Z}$ | $10 \leq w \leq 13$ | $mj$ |
| Stage 2 threshold | $p_2$ | $10^{-8} \leq p_2 \leq .005$ | $mj$ |
| Clock freq. | $f_{1a}, f_{1b}, f_2$ | $10 \leq f \leq 133.3$ MHz | $sj$ |
| Buffer size | $b_i \in \mathbb{Z}$ | $2 \leq b_i \leq 16, 1 \leq i \leq r$ | $sj$ |
| Processor cores | $c \in \mathbb{Z}$ | $1 \leq c \leq 4$ | $sj$ |

Fig. 3: Design variables of BLASTN application.

| Variable | Symbol | Category |
|---|---|---|
| Number of parallel sorts | $2^N$ | $top$ |
| Mapping choices | binary: $m_0, m_1, m_{CR}, m_{IR}$ | $top$ |
| Input mean job arrival rate | $\lambda_{in} \in \mathbb{R}_+$ | $mj$ |
| Compute resource type | binary: $cpu_j$ or $fpga_j$ | $mj$ |
| Comm. message size | $2^M$ | $mj$ |
| Comm. resource type | binary: $smem_j$ or $gige_j$ | $sj$ |
| Number of resources | $nRes_j$ | $sj$ |
| Sort algorithm | binary: $alg_1$ or $alg_2$ | $sj$ |

Fig. 5: Design variables of streaming sort application.

Mean service rates for the remaining queueing stations are expressed similarly. With $p_i$ denoting the probability that a stage passes an input to its downstream neighbor, the relationship between the arrival rates at each stage are below.

$$
\begin{aligned}
\lambda_{1a1} &= \lambda_{in} \\
\lambda_{1a2} &= 2 \cdot p_{1a} \cdot \lambda_{1a1} \\
\lambda_{1a,j} &= 2 \cdot \lambda_{1a,j-1} \quad for \quad 3 \leq j \leq r \\
\lambda_{1b} &= \lambda_{1a,r} \\
\lambda_2 &= p_{1b} \cdot \lambda_{1b} \\
\lambda_3 &= p_2 \cdot \lambda_2 \\
\lambda_0 &= \lambda_{in} + \lambda_3
\end{aligned}
$$

The resulting optimization problem is MINLP, and state-of-the-art solvers (FilMINT) fail to find even a feasible solution.

We begin with branching on $r \in top$, which results in 2 subproblems. Next, we branch on $\lambda_{in} \in mj$. Although $\lambda_{in}$ is real-valued, there are only a finite number of discrete values that can be realized in practice. For this problem, we evaluate 100 of its values. $\lambda_{in}$ is upper-bounded by the possible service rates. Next, we branch on $k \in mj$. Each branch here yields 9 subproblems. We then branch successively on $q \in mj$, $w \in mj$, $m \in mj$, and $p_2 \in mj$. All the remaining variables at this level are in $sj$ and they include $f_{1b} \in \mathbb{R}_+$, $f_2 \in \mathbb{R}_+$, and $c$. If we evaluate only 100 discrete values for each of $f_{1b}$ and $f_2$, the number of branches evaluated will be 204 rather than the otherwise required $40,000$.

In total, a completely enumerated branch and bound search of the design space requires $\approx 2 \times 10^{18}$ evaluations (assuming only 100 evaluations for each real variable, a very conservative assumption). Our techniques reduce this to $\approx 2 \times 10^{12}$ evaluations, a reduction of a million fold.

### B. Streaming sort

Our second application is a streaming implementation of sorting. Input data is *split* into parts, each part is *sorted* in parallel, the results from all the sorts are *merged*, and the
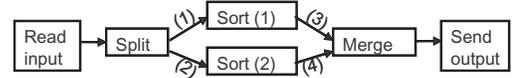
sorted data is output. This application topology is shown in Figure 4. The streaming sort is an example of a data decomposition application. For the sorting application, the queueing model was introduced in [8]. For the application topology shown in Figure 4, the queueing network model is shown in Figure 6. The set of design variables is enumerated in Figure 5. The performance objectives for the streaming sort application include both minimizing latency and maximizing throughput which are combined using the standard technique of weighted sums, similar to how it was done for the BLASTN application. Note that the number of parallel sort blocks, denoted by $2^N$, controls the degree of parallelization which in turn impacts the tradeoff between application latency and throughput. As with BLASTN, the optimization problem formulation for the streaming sort is also mixed-integer and nonlinear and the state-of-art-solvers we mentioned earlier are unable to find even a feasible solution. The number of variables and constraints in the original problem range from (50, 30) to (399, 3077) as we increase $N$ from 1 to 13. This corresponds to $2^N = 2$ to 8192 parallel sorts.

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 1 to 3 (i.e., 2 to 8 parallel sorts), the size of the completely enumerated branch and bound search space is approximately $6 \times 10^{20}$ possible configurations. Using our techniques presented here, this search space decreases to approximately $3 \times 10^{10}$ configurations, reducing by a factor of 20 million. As the range of $N$ increases, the size of the completely enumerated search space simply explodes, reaching $10^{260}$ for $N$ ranging from 1 to 10. At this range of $N$, our techniques yield a search space of size $5 \times 10^{12}$.



Fig. 6: Queueing network model of streaming sort.

| Variable | Symbol | Number of values | Category |
|---|---|---:|---|
| Number of processors | $p$ | 9 | *top* |
| Communication link contention level | $w$ | 1000 | *top* |
| No. of processors in comm. synchronization | $c$ | $1 \leq c \leq p$ | *top* |
| No. of disks | $d$ | $1, 2, \cdots, p$ | *top* |
| Parallel computation service rate | $\mu_{par}^{CPU}$ | 100,000 | *sj* |
| Transfer service rate | $\mu_R^{COM}$ | 1000 | *sj* |
| I/O transfer service rate | $\mu_R^{I/O}$ | 12 | *sj* |

Fig. 7: Design variables of cyclic SPMD applications.

| Variable | Symbol | Number of values | Category |
|---|---|---:|---|
| Number of query processors | $P$ | 100 | *top* |
| Number of busses | $B$ | 100 | *top* |
| Number of file servers | $F$ | 150 | *top* |
| System load | $N$ | 46 | *mj* |
| LLT size | $M$ | 18 | *mj* |
| Input transaction rate | $\lambda$ | 10 | *mj* |
| Service rate of $P_j$ | $\mu_{P_j}$ | 100,000 | *sj* |
| Service rate of $B_j$ | $\mu_{B_j}$ | 12 | *sj* |
| Service rate of $F_j$ | $\mu_{F_j}$ | 10 | *sj* |
| Abort probability at stage $j$ | $\delta_j$ | 200 | *sj* |
| Rollback prob. at stage $j$ | $\gamma_j$ | 100 | *sj* |

Fig. 8: Design variables of LLT processing system.

## C. Cyclic SPMD applications

Cremonesi and Gennaro [3] present a queueing network model for a family of cyclic SPMD applications executing on MIMD platforms. They validate their model on the high-performance applications of a PDE solver and a quantum chemical reaction dynamics code. The set of design variables is summarized in Figure 7. Computation service rates depend on architectural choices, which (in an application-specific systems context) typically include varying cache sizes, cache line sizes, instruction sets, register counts, and so on, which can easily result in $100,000$ or more possible configurations [5]. I/O service rates typically range over at least a dozen choices such as PCIe bus, Infiniband switch, Gigabit Ethernet, etc.

A conservative number of potential values considered for these variables is listed in Figure 7. These choices imply that in the worst case, standard branch and bound needs to search $\approx 5 \times 10^{16}$ possible configurations. However, if we categorize the design variables as in Figure 7 and use our heuristic of ordering variables during branch and bound, one only needs to search $\approx 4 \times 10^7$ possible configurations. The reduction in size of the search space is on the order of $10^9$ configurations.

## D. Long-lived Transaction Processing

The final application we present is a long-lived transaction (LLT) processing system for database management systems (DBMS) modeled by Liang and Tripathi [6]. Here, the performance of the overall transaction processing system being modeled includes the effects of data locking, resource contention, and failure recovery. The design variables of this application are recapped in Figure 8. The choices for the service rate of the above-mentioned components are derived following similar reasoning as that for the cyclic SPMD application above.

In the worst case, standard branch and bound needs to search through $\approx 3 \times 10^{21}$ possible configurations. However, if we categorize the design variables according to the topological information (as shown in Figure 8) and use the heuristic ordering of variables during branch and bound, we only need to search $\approx 10^{10}$ possible configurations. The reduction in the search space is on the order of $10^{11}$ configurations.

## E. Discussion

For the four real-world applications presented, the search space reductions are of the order of $10^6$, $2 \times 10^7$ to $10^{248}$ depending on the parallelism used, $10^9$, and $10^{11}$, respectively.

While in the general case the search space size is still exponential in the number of parameters, for each of these applications the reduced size of the search space is $\leq 10^{12}$ configurations, which can be searched in under 3 hours on 100 cores if each evaluation takes no more than $1~\mu$s.

## IV. CONCLUSIONS

Our domain-specific branch and bound framework is motivated by the fact that none of the state-of-the-art MINLP solvers are able to find even a feasible solution for our streaming applications. In particular: (1) We identified and categorized domain-specific topological information embodied in queueing network models. (2) Using the categories, we developed a heuristic that orders the branching variables such that each branching operation leads to more decomposition of the search space than the subsequent branching. We introduced a way to identify decomposability of our applications based on the canonical Jordan block form. (3) Our framework supports anytime solutions if application developers need a suboptimal solution fast. (4) We exemplified the benefit of using our techniques using four real-world applications drawn from the literature. The time required to find the optimal configuration for these applications is reduced to a reasonable level.

## REFERENCES

[1] 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.

[2] J. Buhler, J. Lancaster, A. Jacob, and R. D. Chamberlain, "Mercury BLASTN: Faster DNA sequence comparison using a streaming hardware architecture," in *Reconfigurable Systems Summer Institute*, Jul. 2007.

[3] P. Cremonesi and C. Gennaro, "Integrated Performance Models for SPMD Applications and MIMD Architectures," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 12, pp. 1320–1332, Dec. 2002.

[4] R. Dor, J. M. Lancaster, M. A. Franklin, J. Buhler, and R. D. Chamberlain, "Using queuing theory to model streaming applications," in *Symp. on Application Accelerators in High Performance Computing*, Jul. 2010.

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

[6] D. Liang and S. K. Tripathi, "Performance Analysis of Long-lived Transaction Processing Systems with Rollbacks and Aborts," *IEEE Trans. Knowl. Data Eng.*, vol. 8, no. 5, pp. 802–815, Oct. 1996.

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

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

[9] G. Strang, *Intro. to Linear Algebra*. Wellesley-Cambridge Press, 1980.