Decomposition Techniques for Optimal Design-Space Exploration of Streaming Applications

Shobana Padmanabhan
Yixin Chen
Roger D. Chamberlain

Shobana Padmanabhan, Yixin Chen, and Roger D. Chamberlain, “Decomposition Techniques for Optimal Design-Space Exploration of Streaming Applications,” in *Proc. of 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (PPoPP), Feb. 2013, pp. 285-286.

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

Decomposition Techniques for 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

{spadmanabhan, ychen25, roger}@wustl.edu

Abstract

Streaming data programs are an important class of applications, for which queueing network models are frequently available. While the design space can be large, decomposition techniques can be effective at design space reduction. We introduce two decomposition techniques called convex decomposition and unchaining and present implications for a biosequence search application.

Categories and Subject Descriptors B.8.2 [*Performance and Reliability*]: Performance Analysis and Design Aids

General Terms Design, Performance, Theory

Keywords Optimization, Domain-specific branch and bound

1. Introduction

High performance streaming data applications are frequently deployed on architecturally diverse or hybrid systems (employing chip multiprocessors, graphics engines, and reconfigurable logic). Searching the design space of possible configurations, commonly referred to as design-space exploration, is challenging because: a) the number of configurations is exponential in the number of design parameters; b) the design parameters may interact nonlinearly; and c) the goals of the exploration are often multiple and conflicting.

We approach the design-space exploration of streaming applications as an optimization problem that searches for a globally optimal configuration. Cost functions are derived from queueing network (QN) models of the application, in particular, BCMP networks. Consider Figure 1. A four-stage pipelined application is modeled with a QN consisting of four *service centers*. A service center comprises a queue and one or more servers. The application stages represent functions in the biosequence search application BLAST [1]. It is conventional to denote the mean arrival rate at service center j as $\lambda_j \in \mathbb{R}$ and the mean service rate as $\mu_j \in \mathbb{R}$.

The optimization problem formulations tend to be mixed-integer, nonlinear and NP-hard. We have developed domain-specific branch and bound techniques that exploit topological information about the application's pipelining embodied in the QNs [3, 4]. More precisely, we identify Jordan block form and solve the blocks independently. We define a block as the group of variables associated with a single service center. We call these variables *single* variables and the ones associated with more than one block as *complicating* variables.

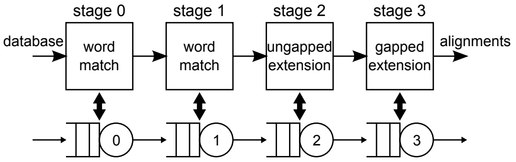

Figure 1. BLAST application and its queueing network model.

After such decomposition, however, the search space size can still be considerable. To reduce the search space further, here we present two decomposition techniques that we call *convex decomposition* (extended from [4]) and *unchaining* (novel here). We analyze the impact of these decomposition techniques by quantifying the reduction in the number of configurations (leaf nodes) to be evaluated during branch and bound search.

2. Convex Decomposition

We define a *convex variable* as a design parameter for which the optimization problem's objective function is convex when all other parameters are held constant (even when the problem itself is nonconvex). In convex decomposition, we decompose the optimization problem to exclude the convex variable and solve for the remaining variables as a new problem (with a new objective function). The solution is substituted back in the original problem which is then solved for the convex variable. We guarantee that the resulting configuration is globally optimum.

Previously [4], we identified sufficient conditions for a convex variable to result in convex decomposition when each normalized performance measure remains the same or improves ("not worsen") with increasing any μ_j in Equation (2) below. Below, we present sufficient conditions for a convex variable to lead to convex decomposition when some normalized measures do not worsen and others remain the same or worsen ("not improve"), as expressed in (1). An increase in the value of a measure is not always the desired goal. An example is power usage. We denote the measures that we wish to decrease by $\vec{\rho}$ and the ones that we wish to increase by $\vec{\tau}$.

Notation. When an optimization problem satisfies the sufficient conditions for a convex variable (an example of which is λ_{in} denoting the application's data ingest rate) to lead to convex decomposition, it takes the form shown below. We denote such a problem as P. n_m denotes the number of measures in the objective function. Let there be n variables in $\vec{v}$ (corresponding to the design parameters); these variables may be integer or real-valued. Let us denote variables other than λ_{in} in $\vec{v}$ by $\vec{v'}$. $\vec{\lambda}$ denotes the vector of the mean job arrival rates at the different stages of the queueing network and $\vec{\mu}$, that of the mean service rates. Functions $\vec{g}$ and $\vec{h}$ are any user-specified constraints; $\vec{\rho}$, $\vec{\tau}$, $\vec{u}$, $\vec{l}$, $\vec{g}$, and $\vec{h}$ may not be convex, differ-

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

PPoPP'13, February 23–27 2013, Shenzhen, China.
ACM 978-1-4503-1922-5/13/02.

entiable, or continuous but are assumed to have closed-form. The problem formulation of P is:

$$\begin{aligned} \min_{\vec{v}} \quad & \sum_{k=1}^{x} W_k \cdot \rho_k \left(\vec{v'}\right) - \sum_{k=x+1}^{n_m} W_k \cdot \tau_k \left(\lambda_{in}\right) & (1) \\ \text{subject to} \quad & \vec{\mu} = \vec{u}\left(\vec{v'}\right) & (2) \\ & \lambda_j = l_j \left(\vec{v'}\right) \cdot \lambda_{in}, \; j = 1, 2, \ldots, n & (3) \\ & \vec{\lambda} < \vec{\mu}; \; \vec{g}\left(\vec{v'}\right) \le 0; \; \vec{h}\left(\vec{v'}\right) = 0 & \end{aligned}$$

The sufficient conditions for a convex variable to result in convex decomposition are: a) each ρ_k in (1) is independent of λ_{in} and $\vec{\lambda}$ while each τ_k depends only on λ_{in}; b) each ρ_k does not improve with increasing any μ_j while each τ_k does not worsen; c) variables in $\vec{\mu}$ are independent of λ_{in} and $\vec{\lambda}$ as shown in (2) whereas variables in $\vec{\lambda}$ depend on $\vec{v'}$ and λ_{in} as shown in (3); and d) variables in λ_{in} and $\vec{\lambda}$ are not involved in the bound of any other design variable, while these variables can be bounded themselves.

Reduction in the number of configurations: If each variable in $\vec{v}$ has d values, the number of leaves in a branch and bound tree is d^n. However, with a convex variable, the number reduces to d^{n-1}.

3. Unchaining

When there is chaining of service centers in a queueing network, the corresponding Jordan blocks (JBs) get chained as well. We define a *chaining variable* as a derived variable (one that depends on one or more of the design parameters) that connects two adjacent JBs that are otherwise independent. A set of $k-1$ chaining variables would then chain k adjacent JBs. Our unchaining solves the blocks independent of each other, while preserving feasibility. We require only a feasible solution here because we expect to have branched on all the parameters not associated with the chained blocks and we expect this to determine the value of the objective function.

To apply unchaining, we identify the following sufficient conditions: a) the only variables remaining in the optimization problem are the single variables associated with each of the chained blocks; and b) the value of the objective function is known.

Notation. Let k adjacent Jordan blocks be *chained*. We index these JBs as $JB_i, i = 1, 2, \cdots, k$. Let P_F be the feasibility (a.k.a. satisfiability) problem in the general form shown below. 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. $\vec{\mu} = (\mu_1, \mu_2, \ldots, \mu_k)$ are derived variables and are real-valued. $\vec{\mu}$, with the exception of μ_1, are the variables that chain the blocks as shown in P_F below. An example of this is illustrated using three blocks in Figure 2 where μ_3 chains JB_3 and JB_2 while μ_2 chains JB_2 and JB_1. Functions $\vec{g}$ and $\vec{h}$ are any user-specified constraints; $\vec{u}, \vec{g}$, and $\vec{h}$ may not be convex, differentiable, or continuous but are assumed to have closed-form. The form of P_F is:

Find a feasible solution for $\vec{v}$

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

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*. E.g., in Figure 2, the right-block of μ_3 is JB_3 and its left-block is JB_2.

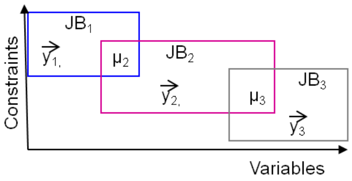

Figure 2. Chaining of blocks.

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. Bounds are determined by formulating a pair of maximization and minimization problems for each of $\mu_k, \mu_{k-1}, \ldots, \mu_2$. Then, we assign feasible values for the single variables of JB_1. We then proceed with assigning values to variables associated with each of $JB_2, JB_3, \ldots, JB_k$, using the solution of the chained variable selected in its left-block to assign feasible values for the variables of its right-block. We guarantee that these solutions together constitute a feasible assignment to all variables $\vec{v}$ in P_F.

Reduction in the number of configurations: When there are chaining variables, the number of subproblems to solve is linear in the number of chained blocks (denoted by k) (i.e., $3k - 1$). Without unchaining, the number of configurations is the product of the number of values of each variable in $\vec{v}$.

4. Empirical Results

For the BLAST models in [2, 3], the number of configurations from the single variables of Jordan blocks 0, 1, 2, and 3, and the complicating variables are $15^5 \times 100 \times 100 \times 4 \times 72 \times 10^6 \approx 2 \times 10^{18}$. Our earlier techniques [3] reduce the number to $15 \times 5 \times (100 + 100 + 4) \times 72 \times 10^6 \approx 10^{12}$. The reduction in JB_0 comes from the assumption that the substages have infinite buffers making the servers independent [2].

In this work, convex decomposition of λ_{in} (in the BLAST model of [2]) reduces 72×10^6 configurations to 72×10^4. The total search space size is reduced to about 10^{10} configurations.

Accounting for dependency arising from finite buffering among the service centers within stage 0 [3], the number of configurations reverts to the original product of configurations at each service center rather than their sum. Even using the techniques described in [3], the search space is only reduced to approximately 10^{16} configurations. Unchaining, however, allows the 15^5 configurations within JB_0 to be reduced to $15 \times 5 \times 2$. The multiple of 2 is because at each step one must solve for both a minimum and a maximum value of the chaining variable. This results in a search space size of approximately 2×10^{12}, and we have almost returned to the search space size of the model in [2].

References

[1] S. F. Altschul and W. Gish. Local alignment statistics. *Methods: a Companion to Methods in Enzymology*, 266:460–80, 1996.

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

[3] 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*, Sept. 2011.

[4] S. Padmanabhan, Y. Chen, and R. D. Chamberlain. Convexity in Non-convex Optimizations of Streaming Applications. In *IEEE Int'l Conf. on Parallel and Distributed Systems*, Dec. 2012.