Automatic Data Layout Using 0-1 Integer Programming

Published in the Proceedings of the International Conference on Parallel Architecture and Compilation Techniques, Montreal, Canada, August 1994

Robert Bixby
Rice University, USA

Ken Kennedy
Rice University, USA

Ulrich Kremer
Rice University, USA

Introduction:

In the optimization world now, even the use of considering the number of processors for parallel programming is carefully considered. The goal of high-level languages is to provide a simple yet efficient machine-independent parallel programming model. The programmer’s data parallel programs should be able to compile and executed with a good performance on many different architectures. However the most sophisticated compiler may not be able to compensate for a poor chosen data layout since many compiler decisions are driven by the data layout specified in the program. Hence the procedure to select a good and efficient data layout is important. The data layout selection problem can be formulated as a 0-1 integer programming problem.

Background for using Integer Programming:

Integer programming is required because the automatic data layout problem is NP-complete. The basic technique for solving integer programs is to apply intelligent branch-and-bound using linear programming at the nodes. There have been remarkable improvements in the area of solving integer programs. LP codes have become much faster then before, there have been major developments in cutting-plane technology and in the area of applications of parallel processing to handle the branching when cutting plans do not succeed in sufficiently strengthening the LP formulation. Hence by using integer programming, the algorithm will determine a good layout for the given compilation system.

Outline and Phase Structure:

The phase structure of the program is represented in the phase control flow graph, an augmented flow graph where each phase is represented by a single node. In the second step, the program alignment space is determined. In the next two steps, data layout search spaces are constructed for each phase. After the generation of the search spaces, a single candidate layout is selected at each phase, resulting in the data layout for the entire program. This step involves the so-called inter-phase data layout problem. The selected candidate layouts have minimal overall cost. The overall cost is determined by the cost of each selected candidate layout, and the required remapping costs between selected candidate layouts.

Inter-Phase Layout as a 0-1 Problem:

An instance of the inter-phase data layout problem is translated into a 0-1 integer-programming problem which can be solved by CPLEX. The problem is modeled as an optimization problem over the data-layout graph. The data layout graph has one node for each candidate data layout. Edges represent possible remappings between candidate data layouts. Nodes and edges have weights representing the overall cost of each layout and remapping, respectively, in terms of execution time. The costs reflect the frequencies or probabilities of phase execution. To solve the inter-phase data layout problem, a single data candidate layout must be chosen for each phase...
such that the overall cost of the selected layouts is minimal. The corresponding data layout graph has \( m_i \) nodes for each phase \( i \), \( 1 < i < n \). Each node represents a particular candidate data layout that specifies the alignment and distribution of all variables referenced in the phase.

An instance of a 0-1 problem consists of a set of variables \( X \), a set of linear constraints over the variables in \( X \), and a linear objective function with domain \( X \). The set \( X \) is the union of two sets of variables, \( X = (X_{\text{layout}} \cup X_{\text{remap}}) \). \( X_{\text{layout}} \) contains a single switch for each node in the data-layout graph, and \( X_{\text{remap}} \) has one switch for each edge. Switch is nothing but being assigning values of 1 if on and 0 if off. The switch \( x_{ik} \in X_{\text{layout}} \) represents the \( k \)-th node of the \( i \)-th phase. The switch \( x_{ik}^{jl} \) represents the remapping edge between the \( l \)-th node of the \( j \)-phase and the \( k \)-th node of the phase \( i \). The integer program is as follows: \( \text{cost}_{\text{layout}} \) & \( \text{cost}_{\text{remap}} \) represent node and edge weights.

Objective Function: \[
\min Z = \sum_{i,k} x_{ik} \in X_{\text{layout}} \text{cost}_{\text{layout}} (x_{ik}) + \sum_{i,k} x_{ik} \in X_{\text{remap}} x_{ik}^{jl} \text{cost}_{\text{remap}} (x_{ik}^{jl})
\]

s.t.

(Layout Constraints) \[
\sum_{k=1}^{m_i} x_{ik} = 1 \quad \text{[Exactly one switch has to be on for each phase and all other switches have to be off]}
\]

( Remapping Constraints )
\[
\sum_{j=1}^{m_j} x_{ik}^{jl} = x_{ik}^{jl} \quad \text{[For the node represented by } x_{ik} \text{, these are the IN-Coming edges, i.e. if switch } x_{ik} \text{ is on then exactly one switch representing an edge from phase } j \text{ must be on]}
\]
\[
\sum_{j'=1}^{m_j} x_{ik}^{j'l'} = x_{ik} \quad \text{[Similarly, for the node represented by } x_{ik} \text{ with nodes in the same } j' \text{ as their sinks, these are the OUT-going edges]}
\]

By solving this minimization problem, we get the optimal layout under the above constraints.

Future Work:

All the work has been carried out considering that there was no replication of data. In the absence of replication, each procedure is represented by a single copy of its data layout graph in the data layout graph for the entire program. Hence further investigation for supporting data replication in the framework is being considered. By adding data replication, multiple copies of an array with distinct owners for each copy should be taken care off. The IN and OUT-constraints of the presented 0-1 formulation will have to be modified to accommodate these new edges and additional constraints can be used to enforce an upper bound on the number of copies of an array. The validation of the technique proposed here will only happen after executing this framework.

Conclusion:

The use of the latest and most powerful general purpose linear programming and integer programming techniques have been used here successfully to present an approach for finding out an optimal and efficient data layout. Even though the problem was NP-hard, exact solutions of the automatic data layout have been attained. So if we can still find out special purpose integer programming techniques that can give us the exact solution faster, we can take advantage of the particular structure of the formulation of the data-layout problem.

By: Shailesh A. Bangale