Data Redistribution
in an Automatic Data Distribution Tool *

Eduard Ayguadé, Jordi Garcia, Mercè Gironès,
M. Luz Grande and Jesús Labarta

Computer Architecture Department, Polytechnic University of Catalunya
cr. Gran Capità s/núm, Mòdul D6, 08071 - Barcelona, Spain

Abstract. Data distribution is one of the key aspects to consider in a parallelizing environment for Massive Parallel Processors. Automatic data distribution proposals may be categorized as either static or dynamic, depending on whether the distribution of the arrays is allowed to change throughout the execution of the program. This paper describes the features and implementation of the intra-procedural data remapping module implemented in our automatic data distribution research tool. The solution to the remapping problem for a sequence of computational phases consists in selecting a mapping among the possible candidates for each phase and introduce remapping actions between consecutive phases. Control flow information is used to identify how phases are sequenced during the execution of the application.

1 Introduction

Data distribution is one of the key aspects to consider in a parallelizing environment for Massive Parallel Processors. In these systems, data distribution dramatically affects performance because of the non-uniformity of the memory system. The cost of accessing a local (or close) memory location can be more than one order of magnitude lower than the cost of accessing a remote memory location.

Mapping data into the distributed memory has to be done according to the access patterns within computational intensive phases and parallelism exploitation out of them. There has been a significant amount of work concerning static mappings, where the mapping of each array remains fixed along the execution of the whole program ([LC90], [KLS90], [LC91], [Gup92], [Who92], [CGSS94b], [AGG+94]). Our work focuses on dynamic mappings in which the mapping of an array may change over its lifetime. Data remapping is one of the topics in this area subject of current research ([CP93], [BKK94], [CGSS94a], [PB95]). The main objective of this work has been to devise an algorithm to automatically detect points in the code where to realign or redistribute arrays in order to reduce the total data movement and thus improve performance of the application. Deciding the granularity of the computational phases executed with a static mapping, and among which remapping may be done is also one of the aspects to consider.

* This research was partially supported by Convex Computer Corporation, CONVEX Supercomputers S.A.E, CEPBA (European Center for Parallelism of Barcelona) and by the Ministry of Education of Spain under contracts TIC-880/92 and TIC-429/95.
[BKK94] considers the profitability of data remapping between computational phases. Each phase has a set of candidate mapping schemes. Selecting a mapping scheme for each phase in the entire program is done by representing the problem with the Data Layout Graph. Each possible mapping for a phase is represented with a node. Edges between two nodes in different phases represent the remapping that has to be carried out to execute each phase with the associated mapping. Nodes and edges have weights representing the overall cost of executing a phase with a mapping and remapping costs respectively, in terms of execution time. The problem is translated into a 0-1 integer programming problem suitable to be solved by a state-of-the-art general purpose integer programming solver.

The FCS system [CP93] considers the problem in the framework of a data distribution tool for Fortran90 source codes. In this scope, array-syntax assignment statements and WHERE masks are examined to determine candidate data mappings. A phase is basically a DO-loop containing array-syntax assignment statements or WHERE masks in its body. It uses a tree-exhaustive algorithm with some heuristics to prune the search space. A Conflict Table storing the conflicts between the mappings of the arrays from one phase to the other is the basis of the remapping algorithm. This table determines which remapping options are worth considering at each transition. From this information, a tree showing all the different alternatives of remapping is built. The aim is to determine the path in the tree with the lowest cost. The full remapping tree can easily grow to intractable proportions.

[CGSS94b] represent the problem as an alignment-distribution graph and use a divide-and-conquer approach to the dynamic mapping problem [CGSS94a]. It initially assigns a static mapping to all the nodes and then recursively divides it into regions which are assigned different mappings. Two regions are merged when the cost of the dynamic mapping is worse than the static mapping taking computation, data movement and remapping costs into account. [PB95] also use a divide-and-conquer approach in which the program is recursively decomposed into a hierarchy of candidate phases. Then, taking into account the cost of remapping between the different phases, the sequence of phases and phase transitions with the lowest cost is selected. It uses [Gup92] to assign mappings to the phases generated.

2 Overview of Our Approach

In this section we outline the major aspects of the intra-procedural remapping module implemented in DDT and introduce the working example that is used along the paper. For this example and for simplicity, we only consider one-dimensional distributions; the algorithm in Section 3 deals with the general case.

The intra-procedural data remapping module groups, for each routine, the statements in the original source code into a collection of phases. A phase is either the outermost non-iterative loop in a nest or a call to a routine. For each phase, a set of candidate mappings is obtained for it: if the phase is a loop nest, the candidate mappings are obtained by performing an analysis of