Abstract

Graph Traverse Scheduling is a loop partitioning method for shared memory multiprocessors that achieves minimum execution time of the parallel code generated assuming that a sufficient number of processors are available and synchronization cost is negligible. The method considers the set of statements in the loop body in the partitioning process.

In this paper we study how static schedules can be generated analyzing the compromise between number of processors, load balance and execution time. The method is presented in a descriptive way based on synthetic examples.

1 Introduction

Many methods have been presented in the literature for partitioning DO loops into computations that can be executed in parallel. It is possible to obtain parallel independent computations when there are not cyclic dependence chains in the dependence graph. Problems arise when recurrences or cycles appear. In this case, some loop parallelizing methods try to obtain fully independent partitions [6], [7], [8]. Other methods try to obtain more parallelism by synchronizing dependent computations assigned to different processors [4], [9]. Graph Traverse Scheduling (GTS) falls into the latter group of loop partitioning methods.

GTS is a static partitioning method conceived for parallelizing loops with tight recurrences and constant distance dependences known at compile-time. The method is based on a Hamiltonian dependence cycle that is used for scheduling purposes. This recurrence is
called Scheduling Recurrence Rsch. The basic idea of the method is to obtain a schedule in which dependences imposed by the original sequential execution are preserved either by internalizing them in the computations assigned to each processor (dependences included in Rsch) or by synchronizing them using some inter-task synchronization mechanism.

The method was presented in [1] for single-nested loops and extended in [2], [3] for multiple-nested loops. The scheduling method goes together with a static evaluation of some loop characteristics which in fact are used as a guidance in the process of determining the efficiency of the parallelization process.

Table 1 introduces terminology used along this paper. The most important concept from the point of view of GTS is the concept of thread. A thread is considered to be a set of computations among which an execution order is imposed by the Rsch. In the single-nested loop case, the parallelism of the loop is constant leading to equally sized threads. However, in the multiple-nested loop case, the instantaneous parallelism varies along the execution of the loop. As a result, some of the threads determine the execution time of the whole parallel set of threads whereas others take less time to complete their execution.

A static scheduling is obtained by assigning the execution of threads to processors, avoiding run-time scheduling overheads. A multiprogrammed execution of those threads is also possible when fewer processors than the number of threads generated are allocated. In the general case of loops with other dependences apart from those in Rsch, a preemptive run-time scheduling would be necessary in order to avoid deadlocks.

GTS, as presented in [3], determines from the Rsch and the limits of the iteration space the number of processors that can be used to execute the loop in minimum time. In this paper we describe how to reduce the number of processors by statically grouping the execution of several threads in a task. We present a grouping strategy and study the compromise between number of processors, load balance and execution time.

### Basic Concepts and Definitions

- **Normalized Loop L**: its a perfect nested-loop structure that iterates in each loop dimension \( k \) from 0 to some upper bound \( N^k - 1 \) in steps of 1.
- **Iteration Space IS**: the IS of a d-nested loop is a d-dimensional discrete Cartesian space in which each point represents the execution of all the statements in one iteration of the loop. Each point is identified by the values of the loop control variables.
- **Statement per Iteration Space SIS**: it is a (d+1) dimensional discrete space defined by the Cartesian product IS x V, being V the set of statements in the body of the innermost loop. Each point represents an instance of a given statement.
- **Free-Point set FP**: given a set of dependence relations \( E \), the FP set is defined as the set of points in SIS which do not depend on any previous execution.
- **Thread**: it is a set of points in SIS among which an execution order is implied by a dependence chain of the dependence graph.
- **Thread Set TS**: subset of points of the unbounded SIS from which the whole set of threads generated by a dependence chain can be obtained.
- **Task**: collection of threads statically assigned for being executed on a given processor.
- **Task Set TAS**: subset of points of the TS from which the whole set of tasks generated by a dependence chain and a grouping arc can be obtained.

Table 1: Basic Concepts and Definitions