Abstract. To optimize programs for parallel computers with distributed shared memory two main problems need to be solved: load balance between the processors and minimization of interprocessor communication. This article describes a new technique called data-driven scheduling which can be used on sequentially iterated program regions on parallel computers with a distributed shared memory. During the first execution of the program region, statistical data on execution times of tasks and memory access behaviour are gathered. Based on this data, a special graph is generated to which graph partitioning techniques are applied. The resulting partitioning is stored in a template which is used in subsequent executions of the program region to efficiently schedule the parallel tasks of that region. Data-driven scheduling is integrated into the SVM-Fortran compiler. Performance results are shown for the Intel Paragon XP/S with the DSM-extension ASVM and for the SGI Origin2000.

1 Introduction

Parallel computers with a global address space share an important abstraction appreciated by programmers as well as compiler writers: the global, linear address space seen by all processors. To build such a computer in a scalable and economical way, such systems usually distribute the memory with the processors. Parallel computers with physically distributed memory but a global address space are termed distributed shared memory machines (DSM). Examples are SGI Origin2000, KSR-1, and Intel Paragon XP/S with ASVM [3]). To implement the global address space on top of a distributed memory, techniques for multi-cache systems are used which distinguish between read and write operations. If processors read from a memory location, the data is copied to the local memory of that processor where it is cached. On a write operation of a processor, this processor gets exclusive ownership of this location and all read copies get invalidated. The unit of coherence (and therefore the unit of communication) is a cache line or a page of the virtual memory system. For this reason, care has to be taken to avoid false sharing (independent data objects are mapped to the same page).
There are two main problems to be solved for parallel computers with a distributed shared memory and a large number of processors: load balancing and minimization of interprocessor communication. Data-driven scheduling is a new approach to solve both problems. The compiler modifies the code such that at run-time data on task times and memory access behaviour of the tasks is gathered. With this data a special graph is generated and partitioned for communication minimization and load balance. The partitioning result is stored in a template and it is used in subsequent executions of the program region to efficiently schedule the parallel tasks to the processors.

The paper is organized as follows. After giving an overview of related work in section 2, section 3 gives an introduction to SVM-Fortran. In section 4 the concept of data-driven scheduling is discussed and in section 5 performance results are shown for an application executed on two different machines. Section 6 concludes and gives a short outlook of further work.

2 Related Work

There are a number of techniques known for parallel machines which try to balance the load, or to minimize the interprocessor communication, or both.

Dynamic scheduling methods are well known as an attempt to balance the load on parallel computers, usually for parallel computers with a physically shared memory. The most rigid approach is self scheduling [10] where each idle processor requests only one task to be executed. With Factoring [7]) each idle processor requests at the beginning of the scheduling process larger chunks of tasks to reduce the synchronization overhead. All of these dynamic scheduling techniques take in some sense a greedy approach and therefore they have problems if the tasks at the end of the scheduling process have significantly larger execution times than tasks scheduled earlier. Another disadvantage is the local scheduling aspect with respect to one parallel loop only and the fact that the data locality aspect is not taken into account.

There are several scheduling methods known which have a main objective in generating data locality. Execute-on-Home [6] uses the information of a data distribution to execute tasks on that processor to which the accessed data is assigned (i.e. the processor which owns the data). An efficient implementation of the execute-on-home scheduling based on data distributions is often difficult if the data is accessed in an indirect way. In that case, run-time support is necessary. CHAOS/PARTI [13] (and in a similar manner RAPID [4]) is an approach to handle indirect accesses as it is for example common in sparse matrix problems. In an inspector phase the indices for indirection are examined and a graph is generated which includes through the edges the relationship between the data. Then the graph is partitioned and the data is redistributed according to the partitioning.

Many problems in the field of technical computing are modeled by the technique of finite elements. The original (physical) domain is partitioned into discrete elements connected through nodes. A usual approach of mapping such