A Scalable Method for Run-Time Loop Parallelization

Lawrence Rauchwerger, Nancy M. Amato, and David A. Padua

Current parallelizing compilers do a reasonable job of extracting parallelism from programs with regular, well behaved, statically analyzable access patterns. However, they cannot extract a significant fraction of the available parallelism if the program has a complex and/or statically insufficiently defined access pattern, e.g., simulation programs with irregular domains and/or dynamically changing interactions. Since such programs represent a large fraction of all applications, techniques are needed for extracting their inherent parallelism at run-time. In this paper we give a new run-time technique for finding an optimal parallel execution schedule for a partially parallel loop, i.e., a loop whose parallelization requires synchronization to ensure that the iterations are executed in the correct order. Given the original loop, the compiler generates inspector code that performs run-time preprocessing of the loop's access pattern, and scheduler code that schedules (and executes) the loop iterations. The inspector is fully parallel, uses no synchronization, and can be applied to any loop (from which an inspector can be extracted). In addition, it can implement at run-time the two most effective transformations for increasing the amount of parallelism in a loop: array privatization and reduction parallelization (element-wise). The ability to identify privatizable and reduction variables is very powerful since it eliminates the data dependences involving these variables and...
thereby potentially increases the overall parallelism of the loop. We also
describe a new scheme for constructing an optimal parallel execution schedule
for the iterations of the loop. The schedule produced is a partition of the set of
iterations into subsets called *wavefronts* so that there are no data dependences
between iterations in a wavefront. Although the wavefronts themselves are con-
structed one after another, the computation of each wavefront is fully parallel
and requires no synchronization. This new method has advantages over all
previous run-time techniques for analyzing and scheduling partially parallel
loops since none of them simultaneously has all these features.

**KEY WORDS:** Run-time; parallelization; schedule; doall; wavefront.

1. **INTRODUCTION**

To achieve a high level of performance for a particular program on today's
supercomputers, software developers are often forced to tediously hand-
code optimizations tailored to a specific machine. Such hand-coding is diff-
ficult, error-prone, and often not portable to different machines. Restruct-
turing, or parallelizing, compilers address these problems by detecting and
exploiting parallelism in sequential programs written in conventional
languages. Although compiler techniques for the automatic detection of
parallelism have been studied extensively over the last two decades, current parallelizing compilers cannot extract a significant fraction of the
available parallelism in a loop if it has a complex and/or statically insuf-
iciently defined access pattern. This is an extremely important issue
because a large class of complex simulations used in industry today have
irregular domains and/or dynamically changing interactions. For example,
SPICE for circuit simulation, DYNA-3D and PRONTO-3D for structural
mechanics modeling, GAUSSIAN and DMOL for quantum mechanical
simulation of molecules, CHARMM and DISCOVER for molecular
dynamics simulation of organic systems, and FIDAP for modeling complex
fluid flows.

Thus, since the available parallelism in theses types of applications
cannot be determined statically by present parallelizing compilers, compile-time analysis must be complemented by new methods capable of
automatically extraction parallelism at run-time. The reason that run-time
techniques are needed is that the access pattern of some programs cannot
be determined statically, either because of limitations of the current
analysis algorithms or because the access pattern is a function of the input
data. For example, most dependence analysis algorithms can only deal with
subscript expressions that are linear in the loop indices. In the presence of
nonlinear expressions, a dependence is usually assumed. Compilers usually
also conservatively assume data dependences in the presence of subscripted
subscripts. More powerful analysis techniques could remove this last