Automatic Parallelization of \textit{while}-Loops Using Speculative Execution$^1$

Jean-François Collard$^2$

\textit{Received} March 28, 1994

Automatic parallelization of imperative sequential programs has focused on nests of \textit{for}-loops. The most recent of them consist in finding an affine mapping with respect to the loop indices to simultaneously capture the temporal and spatial properties of the parallelized program. Such a mapping is usually called a “space-time transformation.”

This work describes an extension of these techniques to \textit{while}-loops using speculative execution. We show that space-time transformations are a good framework for summing up previous restructuration techniques of \textit{while}-loops, such as pipelining. Moreover, we show that these transformations can be derived and applied automatically.

\textbf{KEY WORDS:} Automatic parallelization; space-time transformation; speculative execution; \textit{while}-loop.

1. INTRODUCTION

Many recent parallelization techniques for \textit{for}-loop nests are based on loop transformation.$^{(1,2)}$ These transformations generally boil down to the compile-time determination of a scheduling function (or just “schedule”)$^{(3,4)}$ and a processor mapping for every statement in the source program. These functions assign an execution data and processor coordinates to every statement instance. They are affine functions with respect to loop indices, and build up what is usually called a “space-time mapping” or “space-time transformation.”$^{(5)}$

---

$^1$ Partly supported by the French CNRS Coordinated Research Program on Concurrency, Communication and Cooperation C3, PRC/MRE contract ParaDigne and DRET contract 91/1180.

$^2$ Laboratoire de l’Informatique du Parallélisme (LIP), ENS Lyon, 46 Allée d’Italie, 69364 Lyon Cedex 07, France. E-mail: Jean-Francois.Collard@ens-lyon.fr.
On the other hand, potential parallelism in while-loop has been pointed out for a long time. Pipelining of iterative methods was described in Ref. 6 for instance. But few methods have been proposed to automatically parallelize such loops. These methods can roughly be broken into two classes. The first one tries to transform the while-loop into a for-loop without conditional branching by extracting a variable suitable to be a for-loop index. The second one deals with “true” while-loops, i.e. such while-loops that cannot be transformed into “pure” for-loops. For an analysis of both methods, see Ref. 7. The rest of this paper will only deal with the second kind of while-loops.

Automatic parallelization\(^{(8,9)}\) and pipelining\(^{(7)}\) of loops with tests, including whiles as well as fors with conditional branching, have already been addressed, but space-time transformations of general nests of fors and whiles have not been studied until Griebl and Lengauer.\(^{(10,11)}\) This paper also extends the space-time framework to while loops, but differs from their work in three points: 1) Our model is more restricted, since we restrict ourselves to a single while surrounding for-loops. (Note that this kind of program is common in scientific applications, where, for instance, a while iterates a convergent computation on a discrete space scanned by a nest of fors.) On the other hand, we are then able to use an array dataflow analysis. 2) This paper uses speculative (or eager) execution to speed computations up. 3) We propose a method to derive scheduling functions for speculative operations, enhancing an already existing algorithm.\(^{(12)}\) Our main contribution is to show that the space-time transformation framework is adequate to express the speculative parallelism found in programs conforming to our model, and that high speed-ups can be achieved in this way.

Speculative execution first appeared in the context of Instruction-Level Parallelism (ILP)\(^{(13)}\) and in first parallel implementations of logic and functional languages.\(^{(14-16)}\)

**Speculation in ILP:** Loops with tests can be executed “eagerly”\(^{(9)}\) to obtain “saturation” of the processor. Branches are “bypassed,” and the corresponding path tree is pruned dynamically when the results of the tests are known. Eager execution may be used to generate optimized code on dataflow machines.\(^{(17)}\) Boosting is an architectural mechanism to support speculative execution.\(^{(18)}\) A comprehensive and abstract study of control flow analysis, multiple control flow executions and speculative execution has been proposed by Lam and Wilson.\(^{(19)}\)

**Speculation in logic and functional languages:** OR-parallelism was invented in PROLOG interpreters to process in parallel the literals of a clause, provided enough free processors were available.\(^{(20)}\) The purpose was