Program Translation Tools for Systolic Arrays
N00014-87-K-0385
Final Report

Thomas Gross *and* H. T. Kung

School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

June 30, 1991

1 Summary

The work under this contract has concentrated on *Parallel Program Generators* for a systolic array (the Warp machine). A Parallel Program Generator (PPG) translates a program description for a single address space and a single thread of control into code for each of the nodes in parallel distributed-memory system. We investigated three different approaches, each is discussed in more detail in a separate section:

- Use of data parallelism to execute independent iterations on different cells
- Transformation of nested loops to systolic programs
- Multi-model code mapping for a side-effect free program.

All PPGs produce code that is then translated into Warp microcode by the W2 compiler developed with funding from ONR and DARPA.

In addition, we build a debugger for Warp. Most of this work was reported on in the last interim report and is not repeated here.

These PPGs have simplified the programming of systolic arrays significantly. They have given us the ability to generate a large number of programs for parallel machines with minimal effort. This allowed us to use these machines in ways we never expected (today, 5 years after the Warp machine was put in service, it is still working at Carnegie Mellon, and there is a secondary market for used Warp machines!). It allowed us to build more machines than we ever planed (General Electric delivered 18 machines to government and commercial customers). And it provided the seeds for the integrated iWarp system (the "integrated Warp" designed by Carnegie Mellon University and its industrial partner, Intel Corp).

2 Data parallel programs on Warp

A large number of scientific programs contain data parallelism, but at the time we started our research, few if any compilers were able to take advantage of this parallelism. For any computation, there were numerous ways to map the computation onto a parallel machine. We discovered that some aspects of the mapping process are easy to control for a user (like deciding if a matrix should be partioned by row or by column), while others are easy to handle by a programming tool (like keeping track of the details of inserting "send" and "receive" statements).

One of our Ph. D. students (P. S. Tseng, Ph. D. '89) investigated this area in more detail and developed the *AL* (for Array Language) programming model. In the AL model, a computation is described for a single thread of control. There are two types of variables: global and private. The programmer is responsible to move data from global to private name space (a simple copy or assignment operation is sufficient). Thus a user hint determines when a variable is private and when it is in global, and this information is sufficient to generate efficient parallel programs. The 24 Livermore Loops can serve as an example to show how widely this approach is applicable. These loops were translated by hand to the input language accepted by AL; these changes were syntactic (except that the computation was changed from double-precision to single-precision since the Warp hardware does not support double-precision floating point operations). Then hints were added to allow the AL parallel program generator to distribute the computation over 10 cells. Figure 1 shows the speedup over the 1-cell version, which is free of any parallelization overhead. As can be seen from this Figure, there are some loops that are inherently sequential, but for a number of loops, the AL program generator delivers an impressive speedup.

3 Systolic programs from nested loops

The programs produces with the AL program generator use the high bandwidth of the Warp machine but use only seldomly the systolic gates. That is, when receiving an item from another processor, the item is first stored in local memory before it is operated upon. The systolic program generator (researched by H. Ribas, Ph.D. '90) produces programs that operate directly on the items received from another cell. That is, these programs are systolic programs.

To produce such code, the PPG must control the details of mapping computation and communication. That is, the PPG needs exact dependence information, which is not available for all possible programs. For example, if a program includes an update of an array with a variable stride, then the PPG cannot obtain exact dependence information. However, for a large class of computations, the information can be made available; each of these computations can be described as a set of nested loops, *with some constraints on the loop indices.* Such loop nests are the domain of this parallel program generator, and Figure 2 shows the speedup for a *subset* of the Livermore Loops (using single-precision arithmetic). Since the programs are systolic, the first and last cell of the array must be used as I/O staging processors, and this limits the maximum number of processors that can be applied to one task to eight.

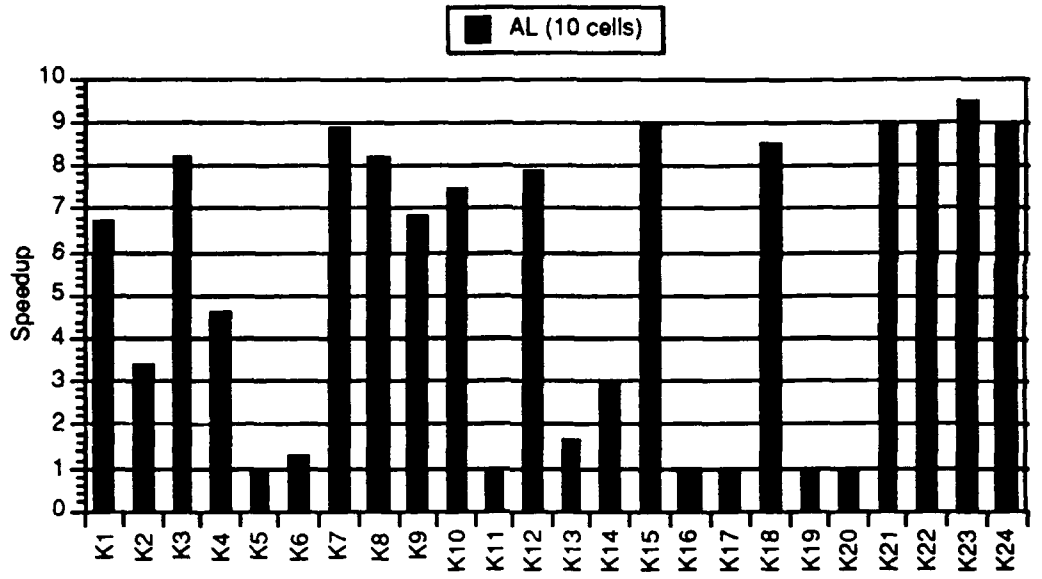

Figure 1: Speedup for Livermore Loop Kernels (AL) on 10-cell Warp machine

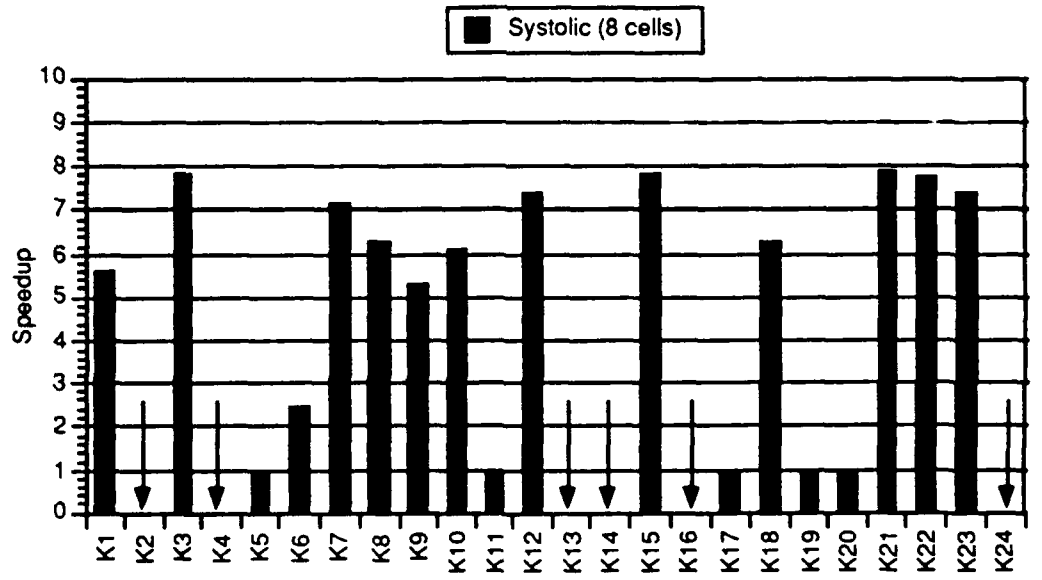

Figure 2: Speedup for Livermore Loop Kernels (Loop Nest) on 8-cell Warp machine

An arrow in Figure 2 indicates that this kernel cannot be translated using the systolic PPG for nested loops. Either the program is not a perfect loop nest with a loop body of a single basic block, or the loop counters assume values that make it impossible to obtain exact dependence information.

4 Comparison

Since a different number of cells can be used by the two parallel program generators, a direct comparison is difficult. Figure 3 shows how efficiently each system uses the cells that are at its disposal. We define $efficiency = (speedup * 100)/(numberofcells)$. [This measure is not completely perfect either; it is easier to obtain high efficiency for a small number of cells than for a large number, but we feel it is a reasonable measure in this case here.]

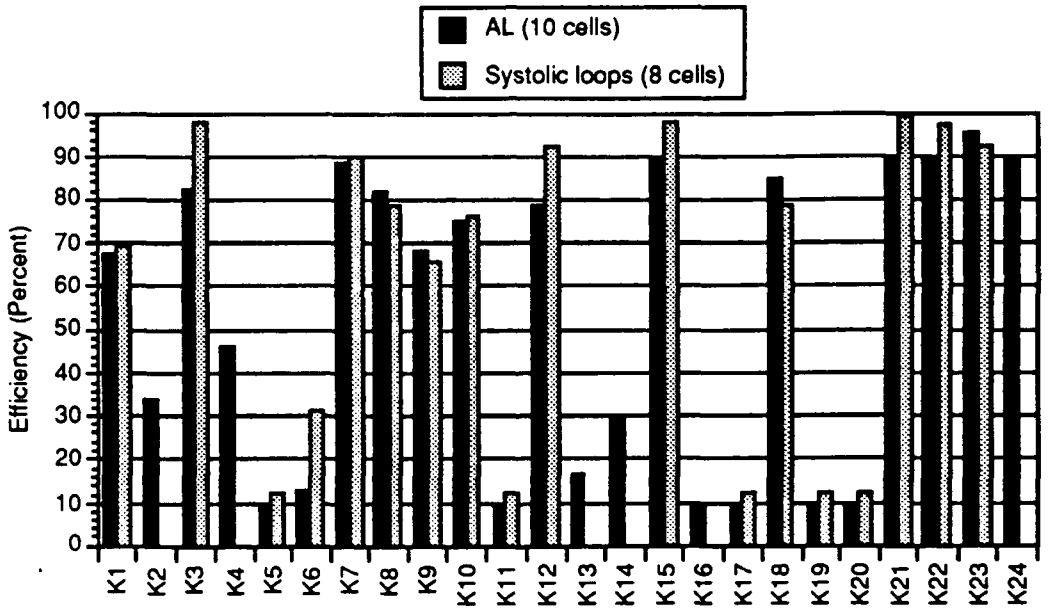

Figure 3: Efficiency for Livermore Loop Kernels (AL, systolic PPG)

It is not surprising to see that the systolic parallel program generator outperforms the AL parallel program generator. The systolic programs include fewer memory references and make better use of the features of the Warp target machine. However, it is also important to note that there are programs that can only be mapped with the AL program generator.

5 (

Multi-model mapping) All attempts to map a computation are influenced by the quality of dependence information. If the parallel program generator cannot determine that two operations

are independent, it cannot execute these operations in parallel. Often a software tool cannot safely determine that two operations are independent although they really are. So it is hard to distinguish failure to parallelize due to resource constraints, communication overhead, or other factors under control of the program generator from a failure to identify parallelism. Furthermore, both AL PPG and systolic PPG for nested loops use a single mapping paradigm. To understand the effect of these constraints, we investigated the use of a executiong model-driven parallel program generator that maps SISAL programs onto the Warp machine. A SISAL program is side-effect free, which means that the programmer has removed all non-essential dependences. This study then confirmed that simple performance model can predict the execution behavior sufficiently well to choose an appropriate mapping strategy; this means that we can automatically make a choice between different mapping strategies. (Further details are available in A. Sussman's thesis (Ph.D. '91).)