ARO 23221.18-EL

(2)

Parallel Functional Computation

Final Report

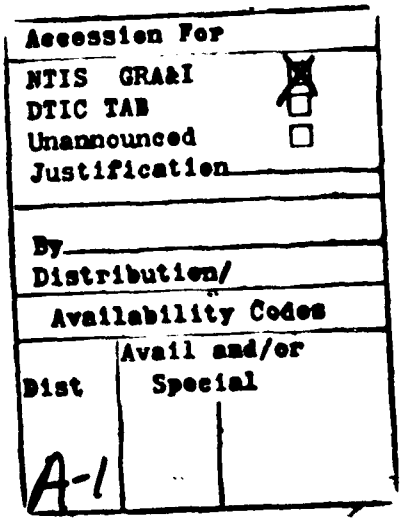
R. R. Oldehoeft

November 9, 1989

U.S. Army Research Office

Contract Number DAAL03-86-K-0101

Colorado State University



The view, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

MASTER COPY - FOR REPRODUCTION PURPOSES

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S) ARO 23221.18-EL
6a. NAME OF PERFORMING ORGANIZATION Colorado State University 6b. OFFICE SYMBOL (If applicable) ------	7a. NAME OF MONITORING ORGANIZATION U. S. Army Research Office
6c. ADDRESS (City, State, and ZIP Code) Fort Collins, CO 80523	7b. ADDRESS (City, State, and ZIP Code) P. O. Box 12211 Research Triangle Park, NC 27709-2211
8a. NAME OF FUNDING / SPONSORING ORGANIZATION U. S. Army Research Office 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER DAAL03-86-K-0101
8c. ADDRESS (City, State, and ZIP Code) P. O. Box 12211 Research Triangle Park, NC 27709-2211	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Parallel Functional Computation

12. PERSONAL AUTHOR(S)
Rodney R. Oldehoeft

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM 6/15/86 TO 8/14/89	1989/November/9	10

16. SUPPLEMENTARY NOTATION
The view, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Parallel computation, functional languages, SISAL

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This is the final report for ARO Contract DAAL03-86-K-0101 titled "Parallel Functional Computation" at Colorado State University. The research described here has developed the functional language SISAL to be a highly efficient mechanism for automatically exploiting shared-memory parallel computers.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION	
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	Unclassified	
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

This is the final report for project DAAL03-86-K-0101 titled "Parallel Functional Computation" and funded by the U. S. Army Research Office begween 15 June, 1986 and 14 August, 1989 at Colorado State University.

1 Problem Statement

The magnitude of problems that people need to solve using automatic computation often exceeds the processing power of available sequential computers. However, most algorithms contain components that can execute concurrently instead of serially, and vendors have provided a variety of machines capable of parallel processing.

But expected reductions in execution times did not always occur owing to ineffective utilization of available parallel hardware or excessive interprocessor communication. Programmers could sometimes manually optimize program structures to solve these problems, but this work is extraordinarily difficult, and not practical for large programs. Research into automatic parallelization follows two paths. First, because of the large investment in extant software, attempts continue toward compilers for conventional languages that map ordinary programs onto parallel architectures efficiently. Systems of this kind work best when programmers use certain easily recognizable forms in source programs and understand underlying parallel hardware.

Second, research continues in this and other projects to design, implement, and evaluate new languages for which automatic parallel execution is more easily attainable. Programs written in functional languages have been known to be easily parallelizable because they possess these features:

- Programs are true functions.
- Assignments to names occur only once.
- Function application does not include side effects.
- Iteration occurs only in special forms.
- Only data dependencies sequentialize evaluation.

One or more factors have hindered the potential for efficiently exploiting parallel architectures via programs written in functional languages:

- Functional programs must do excessive data copying in simple implementations to preserve semantics.
- Researchers have designed special architectures along with functional languages, limiting the applicability of results.
- The theoretical bases have been explored exclusively, so the possible is better understood than the practical.

- A gap exists between practitioners with large problems to solve and the functional language research community.
- Language design has been based on obscure academic systems instead of the more familiar applications languages.
- Producing highly efficient software is not in the scope of academic research projects.

In some cases recent developments have ameliorated some of these problems.

At the time the U. S. Army Research Office funded the work summarized here, the SISAL project had designed a functional language and showed its utility for implicitly parallel execution.

- The language syntax resembled that of conventional, Pascal-like languages [18].
- An implementation on the Denelcor HEP showed that automatic parallel execution was possible without hardware designed just for SISAL [1, 19].
- Performance data showed some successes, and located the sources of additional improvement [21].

The proposal had the following objectives:

- *Performance Enhancement:* Attack the special performance problems of functional languages, generate better object code, reduce run-time overhead.
- *Portability Studies:* Dissasociate SISAL software from all but the most widely available UNIX features, show that portability is possible widely.
- *Production Language Features:* Develop modular language features and other enhancements.
- *Conventional Language Compatibility:* Allow programs to be written partly in SISAL, and partly in, e.g., FORTRAN.
- *Array Operations:* Develop better multi-dimensional array facilities, including array operations in the language.

The next section addresses the success at meeting these objectives.

2 Summary of Project Accomplishments

Dr. Rodney R. Oldehoeft and his research group at Colorado State University, and the Computing Research Group at LLNL, have evolved the SISAL project to its current state:

an efficient testbed for additional research in parallel functional computing. LLNL personnel have written the current front-end parser for SISAL 1.2, designed the intermediate languages IF1 and IF2 [26, 29], written an interpreter for IF1 that also gives graphs of potential parallelism [28], developed several optimizers that work with IF1 [27], written many SISAL benchmarks [11], and continue to collaborate with the CSU group in many ways.

Representatives from LLNL, CSU, the University of Manchester (England) and Digital Equipment Corporation cooperatively defined the original SISAL language. Version 1.1 followed quickly, and in 1985 LLNL, CSU, and UM defined Version 1.2 [18].

2.1 Efficient Processing

The CSU group have concentrated on the efficient native execution of SISAL 1.2 programs. While native code translations of SISAL programs available on a variety of systems [6] execute much faster than any interpreter could run, the compiler was not competitive with those for conventional languages on sequential machines, or with hand-parallelized programs on multiprocessors [20]. The group set two goals:

- SISAL programs should run competitively with those written in conventional languages on a single processor.
- SISAL programs should show efficient use of multiple processors when they are available and the algorithms themselves have inherent parallelism.

These goals may seem contradictory because the easiest way to show good parallel speedup is to generate inefficient code that occupies all available processor resources. This, however, precludes competitive sequential execution. To pursue these goals the CSU group began several optimization projects.

- Run-time support overhead in thread management, dynamic storage allocation, lock code, and processor assignment have been dramatically improved [14, 24]. Code is much more parallel, and simultaneously more efficient, than before.
- The group extended IF1 optimizers developed at LLNL to have a more global effect across entire programs, and developed several new optimizations.
- Staff at LLNL[23] designed "build-in-place" optimization, but did not implement it. This enables, for example, an array that is built as the concatenation of smaller arrays to be placed in its final storage from the beginning, eliminating all sub-array copying from concatenation operations. The CSU group implemented the design, proving the concept. This optimization, and all following, involve expressing SISAL programs in the extended intermediate form IF2.
- They adapted a prototype optimizer of reference count operations for dynamically allocated storage, begun at LLNL, to work globally [8]. In its current implementation

it removes most operations from typical programs. This is important because the operations take time, and updates must take place in critical sections for correct results. Removing operations improves speed and parallel efficiency simultaneously.

- The "update-in-place" problem was attacked. In single assignment languages, an aggregate data structure derived from another by changing one component must, in principle, be copied before modification. Analysis of when the data may be changed without copying is not always possible at compile time. Based on the reference count optimizer, this optimization determines that more efficient code can often be produced, and uses reference counting to handle other cases during execution. The result is execution time reduced by two decimal orders of magnitude for problems such as sequential sorting. The CSU SISAL group have effectively solved this problem for functional languages in a general setting [3, 9, 10].

- The group developed a new code generator that produces C from IF2. The quality of the machine code resulting from compiling this C is often better than that resulting from a handwritten C program.

The performance goals have been met: SISAL programs compiled and run using CSU software are often competitive with their C and FORTRAN counterparts when executing sequentially, and show good parallel speedup and efficiency on shared memory multiprocessors [10].

Members of the group have built two tools to help in analyzing performance problems and ameliorating them. First, one can obtain a histogram across SISAL program text giving the relative amounts of time spent in each function (or in sections within functions). This helps to identify bottlenecks in run-time routines or translations of SISAL functions. Second, an instrumented run-time library emits data reflecting events of interest during parallel execution. Later, one can examine this data interactively on a Sun workstation to identify parallel bottlenecks and other hard-to-obtain behaviors [16].

2.2 Software Portability

The new software discussed above no longer relies on the intermediate forms of any other compiler. By developing all software in C [2, 21], and producing C from the SISAL intermediate form [4], the widest possible portability has been achieved [7]. The compiler-produced C can use structures not generally used by human programmers, and therefore sequential SISAL program performance generally is better than hand-written C programs. The machine-dependent facilities necessary for parallel synchronization are carefully encapsulated for easy porting to new systems. The CSU group have produced SISAL implementations for sequential execution on Vaxes, Suns, and other similar systems, and on commercially available parallel machines such as the Sequent, Encore, and Alliant. A prototype Cray version has been produced [17], a version has been ported to the CMU Warp [13], and a Connection Machine implementation is underway.

2.3 Language Redesign

Based on experience with writing SISAL programs [11, 12, 15, 25] and observations about the possible optimizations, LLNL and CSU SISAL groups cooperatively began a revision of the language. Although the language specification is incomplete [5], the improved version comprises these features:

- *Arrays and associated operators:* Arrays and their manipulation are common in scientific programs, and SISAL 2.0 provides advanced features in this area. Arrays, and sub-arrays defined by subscript expressions, are first-class objects that can be combined with infix arithmetic operators. True multidimensional arrays are available to complement current one-dimensional arrays with arrays as components. Multidimensional arrays are constructable by describing arbitrary sub-array expressions that can be evaluated in parallel. A simple syntax for describing diagonals in multidimensional arrays with arbitrary orientation provides improved expressive power.

- *Modules:* The design for a comprehensive interface between SISAL and other, conventional languages is complete [22]. It adds a modern modular structure to the language that separates specification from implementation, allows a simple form for small programs, and enables SISAL code to use extant subprogram libraries or large applications written in conventional languages to reference SISAL code for subsystems where high performance, parallel execution is necessary.

- *Parameterized types and inference:* One can omit the explicit declaration of some type names and use these names to declare functions. When a function reference is processed, the types of actual parameters and results are known and an instance of the function declaration is automatically instantiated in response. This is a powerful facility for declaring a "generic" function with several instantiations for the various usages of it—smaller programs often result. Parameterized types and inference cause no degradation in execution performance.

- *Higher-order functions:* A programmer can declare function types, pass functions as actual parameters, and obtain values of function types as the result of expression evaluation. Partial evaluation or "Currying" is possible to produce efficient instances of generalized functions. A function definition capability that fixes non-parameter values at definition time is included. Greatly enhanced expressive power can result, but these features are limited to actions during compilation so execution performance is not degraded.

- *More compact syntax:* SISAL is an expressive language with much familiar syntax, and is easy to learn for that reason. Certain economies are introduced to shorten the expression of some common structures.

The language redesign has achieved the remaining project objectives by enabling large program construction, providing general interfaces with conventional languages, and incorporating comprehensive array construction and manipulation facilities.

3 List of Publications

Cann, D. C., Ching-Cheng Lee, R. R. Oldehoeft and S. K. Skedzielewski. *SISAL Multiprocessing Support*, Technical Report UCID-21115, Lawrence Livermore National Laboratory, Livermore, CA, July, 1987.

Cann, D. C. and R. R. Oldehoeft. *Porting Multiprocessor SISAL Software*, Technical Report CS-88-104, Computer Science Department, Colorado State University, Fort Collins, CO, February, 1988.

Cann, D. C. Cann and R. R. Oldehoeft. *Reference Count and Copy Elimination for Parallel Applicative Computing*, Technical Report CS-88-129, Computer Science Department, Colorado State University, Fort Collins, CO, November, 1988.

Cann, D. C. and R. R. Oldehoeft. *High Performance Parallel Applicative Computing*, Technical Report CS-89-104, Computer Science Department, Colorado State University, Fort Collins, CO, February 1989.

Cann, D. C. *Compilation Techniques for High Performance Applicative Computation*, Ph.D. dissertation, Computer Science Department, Colorado State University, Fort Collins, CO, May, 1989.

Hanson, T., S. Harikrishnan, T. Richert and R. Oldehoeft. *The Purdue Parallel Benchmarks in SISAL*, Technical Report CS-88-114, Computer Science Department, Colorado State University, Fort Collins, CO, September, 1988.

Hanson, T. and R. R. Oldehoeft. *Parallel Performance Measurement: Instrumentation and Examples*, Technical Report CS-88-115, Computer Science Department, Colorado State University, Fort Collins, CO, September, 1988.

Hanson, T. *SISAL Runtime Parallel Performance Instrumentation*, Technical Report CS-89-109, Computer Science Department, Colorado State University, Fort Collins, CO, May, 1989.

Harikrishnan, S. *A Type Inferencing Linkage Editor for Data Flow Graphs Generated from SISAL*, Technical Report CS-89-110, Computer Science Department, Colorado State University, Fort Collins, CO, June, 1989.

Oldehoeft, R. R., D. C. Cann and S. J. Allan. "SISAL: Initial MIMD Performance Results," *Proceedings of the 1986 Conference on Algorithms and Hardware for Parallel Processing*, Aachen, Federal Republic of Germany, September, 1986: 120-127.

Oldehoeft, R. R. and J. R. McGraw. "Mixed Applicative and Imperative Programs." To appear in *Parallel Computing*.

Oldehoeft, R. R. and D. C. Cann. "Applicative Parallelism on a Shared-Memory Multiprocessor," *IEEE Software*, January,1988: 62-70.

Oldehoeft, R. R. "Parallelism Granularity in SISAL," *Proceedings of the SRC Parallelism Packaging Workshop*, Supercomputing Research Center, April, 1988: 5.9.1–5.9.13.

Richert, T. R. *Efficient Task Management for SISAL*, Technical Report CS-89-111, Computer Science Department, Colorado State University, July, 1989.

Sieker, Fritz. *A Study of Rendering Graphical Images in SISAL*, Technical Report CS-89-102, Computer Science Department, Colorado State University, Fort Collins, CO, February, 1989.

Skedzielewski, S. K., R. K. Yates and R. R. Oldehoeft. "DI: An Interactive Debugging Interpreter for Applicative Languages," *Proceedings of the ACM SIGPLAN 87 Symposium on Interpreters and Interpretive Techniques*, June, 1987: 102-112.

4 Participating Scientific Personnel

Principal investigator:	Rodney R. Oldehoeft
Ph.D. graduate:	David C. Cann
M.S. graduates:	Thomas C. Hanson Seetharaman Harikrishnan Tamara R. Richert Fritz Sieker
Continuing students:	Khalid Aziz Steven Dominic Matthew Haines Daniel Sweeney Tom Walley

References

[1] S. J. Allan and R. R. Oldehoeft. HEP SISAL: Parallel functional programming. In J. Kowalik, editor, *Parallel MIMD Computation: The HEP Supercomputer and Its Applications*, pages 123–150. MIT Press, Cambridge, MA, 1985.

[2] S. J. Allan and R. R. Oldehoeft. Parallelism in SISAL: Exploiting the HEP architecture. In *19th Hawaii International Conference on System Sciences*, pages 538–548, January 1986.

[3] D. C. Cann. *Compilation Techniques for High Performance Applicative Computation.* PhD thesis, Colorado State University, Computer Science Department, Fort Collins, CO, 1989.

[4] D. C. Cann, S. J. Allan, and R. R. Oldehoeft. An IF1 driven portable code generator. Technical Report CS-84-15, Colorado State University Computer Science Department, Fort Collins, CO, December 1984.

[5] D. C. Cann, J. T. Feo, and R. R. Oldehoeft. *SISAL 2.0 Reference Manual.* In preparation.

[6] D. C. Cann, Ching-Cheng Lee, R. R. Oldehoeft, and S. K. Skedzielewski. SISAL multiprocessing support. Technical report, Lawrence Livermore National Laboratory, *Livermore*, CA, 1987.

[7] D. C. Cann and R. R. Oldehoeft. Porting multiprocessor SISAL software. Technical Report CS-88-104, Computer Science Department, Colorado State University, Fort Collins, CO, February 1988.

[8] D. C. Cann and R. R. Oldehoeft. Reference count and copy elimination for parallel applicative computing. Technical Report CS-88-129, Computer Science Department, Colorado State University, Fort Collins, CO, November 1988.

[9] D. C. Cann and R. R. Oldehoeft. Compiling techniques for high performance parallel applicative computing. *Concurrency Practice and Experience*, 1989. Submitted for publication.

[10] D. C. Cann and R. R. Oldehoeft. High performance parallel applicative computing. Technical Report CS-89-104, Computer Science Department, Colorado State University, Fort Collins, CO, February 1989.

[11] J. T. Feo. The Livermore Loops in SISAL. Technical Report UCID-21159, Lawrence Livermore National Laboratory, Livermore, CA, August 1987.

[12] J. T. Feo. An analysis of the computational and parallel complexity of the Livermore Loops. *Parallel Computing*, 7:163–185, July 1988.

[13] Thomas Gross and Alan Sussmann. Mapping a single-assignment language onto the WARP systolic array. In *Proceedings of the ACM Conference on Lisp and Functional Programming*, October 1987.

[14] T. Hanson. Sisal runtime parallel performance instrumentation. Technical Report CS-89-109, Colorado State University Computer Science Department, Fort Collins, CO, May 1989.

[15] T. Hanson, S. Harikrishnan, T. Richert, and R. Oldehoeft. The purdue parallel benchmarks in SISAL. Technical Report CS-88-114, Colorado State University Computer Science Department, Fort Collins, CO, September 1988.

[16] T. Hanson and R. R. Oldehoeft. Parallel performance measurement: Instrumentation and examples. Technical Report CS-88-115, Colorado State University Computer Science Department, Fort Collins, CO, September 1988.

[17] C. Lee. Experience of implementing applicative parallelism on cray x-mp. Technical Report UCRL-98303, Lawrence Livermore National Laboratory, May 1988.

[18] J. R. McGraw, S. K. Skedzielewski, S. J. Allan, R. R. Oldehoeft, J. Glauert, C. Kirkham, W. Noyce, and R. Thomas. SISAL: Streams and iteration in a single assignment language: Reference manual version 1.2. Manual M-146, Rev. 1, Lawrence Livermore National Laboratory, Livermore, CA, March 1985.

[19] R. R. Oldehoeft and S. J. Allan. Execution support for HEP SISAL. In J. Kowalik, editor, *Parallel MIMD Computation: The HEP Supercomputer and Its Applications*, pages 151–180. MIT Press, Cambridge, MA, 1985.

[20] R. R. Oldehoeft and D. C. Cann. Applicative parallelism on a shared-memory multiprocessor. *IEEE Software*, pages 62–70, January 1988.

[21] R. R. Oldehoeft, D. C. Cann, and S. J. Allan. SISAL: Initial MIMD performance results. In *Proceedings of the 1986 Conference on Algorithms and Hardware for Parallel Processing*, pages 120–127, Aachen, Federal Republic of Germany, September 1986.

[22] R. R. Oldehoeft and J. R. McGraw. Mixed applicative and imperative programs. *Parallel Computing*, 1990. To appear.

[23] J. E. Ranelletti. *Graph Transformation Algorithms for Array Memory Optimization in Applicative Languages.* PhD thesis, University of California at Davis, Computer Science Department, Davis, California, 1987.

[24] T. R. Richert. Efficient task management for SISAL. Technical Report 89-111, Computer Science Department, Colorado State University, July 1989.

[25] Fritz Sieker. A study of rendering graphical images in sisal. Technical Report CS-89-102, Computer Science Department, Colorado State University, Fort Collins, CO, February 1989.

[26] S. K. Skedzielewski and J. Glauert. IF1—an intermediate form for applicative languages. Manual M-170, Lawrence Livermore National Laboratory, Livermore, CA, July 1985.

[27] S. K. Skedzielewski and M. L. Welcome. Data flow graph optimization in IF1. In Jean-Pierre Jouannaud, editor, *Functional Programming Languages and Computer Architecture*, pages 17–34. Springer-Verlag, New York, NY, September 1985.

[28] S. K. Skedzielewski, R. K. Yates, and R. R. Oldehoeft. DI: An interactive debugging interpreter for applicative languages. In *Proceedings of the ACM SIGPLAN 87 Symposium on Interpreters and Interpretive Techniques*, pages 102–112, June 1987.

[29] M. L. Welcome, S. K. Skedzielewski, R. K. Yates, and J. E. Ranelletti. IF2: an applicative language intermediate form with explicit memory management. Manual M-195, University of California Lawrence Livermore National Laboratory, November 1986.