LABORATORY FOR
CONCURRENT COMPUTING SYSTEMS
COMPUTER SYSTEMS ENGINEERING
School of Electrical Engineering
Swinburne Institute of Technology
John Street, Hawthorn 3122, Victoria, Australia.

Proposals for SISAL and OSC

Technical Report 31-014

Pau S. Chang
Greg K. Egan

Version 1.0	Original Document 31/01/90
Version 1.1	Original Document 22/02/90
Version 1.2	Original Document 01/03/90
Version 1.3	Original Document 26/04/90

ABSTRACT

SISAL and its compiler for conventional multiprocessors OSC are relatively new. Documented in this memo are the proposals of some of the improvements necessary for OSC and SISAL which otherwise will keep posing as potential drawbacks of the compiler and the language. They arise from the our experience in the implementations of a two dimensional FFT model and a spectral weather simulation model.

Introduction

SISAL is a relatively new functional language whose efficacy in expressing the potential concurrency of scientific computational models is yet to be judged by practical application studies. Although it was originally targeted as a dataflow language, programs written in SISAL have also been successfully compiled and run with good speedup on multiprocessors based on conventional architecture. Nonetheless, some features still need to be added to the language to improve its expressive capability.

Many optimisation stages have been added in the first released Optimising SISAL Compiler OSC received by us in early 1989. Nevertheless, given the newness of the compiler, there are still a number of improvements necessary to make the compiler more reliable and effective. The known features are the need to adopt the FORTRAN-like expression of multiple dimensional array construct which is closer to the mapping of the physical memory rather than the present SISAL expression of multiple arrays of arrays, and the need to have only one form of loop construct instead of the present sequential and parallel loop constructs.

Presented in note form in the following sections are the proposals for additional improvements in the compiler (sections 1 to 6) and the language (sections 7, 8 and 9). They arise from our experience in the implementations of a two dimensional FFT model and a spectral weather simulation model.

1. Starting Index of "FOR array RETURNS VALUE OF CATENATE"

If we write

```
FOR i IN 0, bound
RETURNS VALUE OF CATENATE i
END FOR
```

we would expect the results to be an array with a starting index of 0. However, the front end SISAL compiler generates IF1 graphs which have a starting index of 1. Additionally, both the Dataflow Interpreter and the C code generated by OSC give the results with a starting index of 1 even if the lower bound in the IF1 graphs is manually set to 0. This is potentially disastrous for computations which habitually consist of arrays whose intended starting indices are 0, such as FFT.

The case study as elaborated in Figure 1 shows that the IF1 code generated by SISAL frontend sets the lower bound of the concatenation result to 1 regardless. Further, even if the low bound is altered to 0 in the IF1 code, both DI and OSC do not check this lower bound given in the IF1 code, but rather simply set it, again regardless, to 1.

We are forced to always use **array_setl** to set the desired lower bound when any loop returning 'value of catenate' is used.

```
FOR i IN 0, 10
RETURNS VALUE OF CATENATE
    FOR j IN 0,10
    RETURNS VALUE OF j
    END FOR
END FOR
```

(i) The SISAL code

```
T 1 1 0   %na=Boolean
T 2 1 1   %na=Character
T 3 1 2   %na=Double
T 4 1 3   %na=Integer
T 5 1 4   %na=Null
T 6 1 5   %na=Real
T 7 1 6   %na=WildBasic
T 8 10
T 9 0        4
T 10 8       9        0
T 11 3       0        10
T 12 4       4
T 13 8       9        10
T 14 3       13       10
T 15 4       9
C$  C Faked IF1CHECK
C$  D Nodes are DFOrdered
C$  E Common Subs Removed
C$  F Livermore Frontend   Version1.8
C$  G Constant Folding Completed
C$  L Loop Invars Removed
C$  O Offsets Assigned
X    11       "main"  %ar=13  %sl=3
E    4 1      0 1     9       %of=1   %mk=V
{ Compound  1  0
G    0        %fq= 0.00000000000000e+00     %ep=0
E    1 1      0 1     12      %na=j  %of=2   %mk=V
N 1  142
L             1 1     4 "0"   %of=3   %mk=V
L             1 2     4 "10"  %of=4   %mk=V
G    0        %fq= 0.00000000000000e+00     %ep=0
G    0        %fq= 0.00000000000000e+00     %ep=0
E    1 1      0 1     9       %of=5   %mk=V
N 1  107
L             1 1     4 "1"   %of=6   %mk=V
E    0 1      1 2     12      %na=j  %of=2   %mk=V          %sl=7
} 1 0 3 0 1 2
N 2  103
L             2 1     4 "1"   %of=7   %mk=V
N 3  115
E    1 1      3 1     9       %of=5   %mk=V
L             3 2     4 "0"   %of=8   %mk=V
{ Compound  4  0
G    0        %sl=5
E    1 1      0 3     12      %na=i  %of=11         %mk=V
N 1  142      %sl=5
L             1 1     4 "0"   %of=12          %mk=V
L             1 2     4 "10"  %of=13          %mk=V
G    0        %sl=5
E    0 2      0 4     9       %of=10          %mk=V
G    0        %sl=5
E    1 1      0 1     9       %of=1   %mk=V
N 1  149      %sl=9
L             1 1     14 "CATENATE"         %mk=V
E    0 1      1 2     9       %of=9   %mk=V
E    0 4      1 3     15      %of=10          %mk=V
} 4 0 3 0 1 2         %sl=5
E    2 1      4 1     9       %of=9   %mk=V
E    3 1      4 2     9       %of=10          %mk=V
```

(ii) The corresponding IF1 code

```
-[ 1: 0 1 2 3 4 5 6 7 8 9 10 0 1 2 3 4 5 6 7 8 9 10
      0 1 2 3 4 5 6 7 8 9 10 0 1 2 3 4 5 6 7 8 9 10
      0 1 2 3 4 5 6 7 8 9 10 0 1 2 3 4 5 6 7 8 9 10
      0 1 2 3 4 5 6 7 8 9 10 0 1 2 3 4 5 6 7 8 9 10
      0 1 2 3 4 5 6 7 8 9 10 0 1 2 3 4 5 6 7 8 9 10
      0 1 2 3 4 5 6 7 8 9 10 ]
```

(iii) The same results produced by DI and OSC

Figure 1: A parallel loop returning value of catenate with intended starting index 0

2. FOR array RETURNS VALUE OF CATENATE of concatenations of vectors

This example arises from coding a two dimensional FFT in SISAL. At compilation time, the process passes through SISAL frontend compiler and the optimisation stages without any indication of problems, but during CC, the CGEN generates several errors regarding the need to use pointers. The problem is shown in Figure 2.

The compilation of the code passes through the SISAL frontend compiler and all of the optimisation stages, but during CC, it is terminated due to some "struct/union" errors generated by CGEN. The problem embeds in:

```
FOR loop
RETURNS ARRAY OF
    FOR index IN lowerbound, upperbound
    vecvec := vector || vector              % concatenation
    RETURNS VALUE OF CATENATE vecvec
    END FOR
END FOR
```

```
osc chip.sis -v
sisal -noopt -nooff -dir /usr/local/sisal chip.sis
 LL Parse, using binary files
* Reading file: chip.sis...

version 1.8     (Mar 28, 1989)

 accepted
   81 lines in program
   0 errors ( calls to corrector)
   0 tokens inserted;   0 tokens deleted.
   0 semantic errors

if1ld -o chip.mono -e main chip.if1
if1opt chip.mono chip.opt -I -e
unlink chip.mono
if2mem chip.opt chip.mem
unlink chip.opt
if2up chip.mem chip.up
unlink chip.mem
if2part /y/rco/rcodf/sisal/release/OSC_csu/bin/s.costs chip.up chip.part -L0
unlink chip.up
if2gen chip.part chip.c -b
unlink chip.part
cc -I/y/rco/rcodf/sisal/release/OSC_csu/bin -DSUN3 -f68881 -O -S chip.c
%"chip.c", line 229: nonunique name demands struct/union or struct/union pointer
%"chip.c", line 230: nonunique name demands struct/union or struct/union pointer
%"chip.c", line 262: nonunique name demands struct/union or struct/union pointer
%"chip.c", line 264: nonunique name demands struct/union or struct/union pointer
 ** COMPILATION ABORTED **
```

(i) Error messages given at compile time

```
define main
type ArrInt1 = ARRAY [integer];
type ArrReal = ARRAY [real];
type ArrReal2 = ARRAY [ArrReal]
GLOBAL SIN(num: real returns real)
GLOBAL COS(num: real returns real)
GLOBAL ATAN(num: real returns real)
GLOBAL SQRT(num: real returns real)

FUNCTION main(RETURNS ArrReal,ArrReal)
LET
n := 4; pi := 3.141593;
twopow :=   for initial      i:=0; pow:=1; two := array_fill(0,n,1);
            while i<n repeat        i:=old i+1; pow := old pow*2;
                                    two := old two[i: pow];
            returns value of two
            end for;
Areal,Aimag:=
    for row in 0, twopow[n] - 1 CROSS col in 0, twopow[n] - 1
    returns array of if row<twopow[n]/2 then 5.0 else 0.0 end if
             array of if row<twopow[n]/2 then 5.0 else 0.0 end if
    end for;
IN
  LET
  stage := 2; off := twopow[n - stage];
  upperboundjump := twopow[stage - 1] - 1; jumpby := twopow[n - stage + 1];
  AR1, AI1:=
    FOR indexjump IN 0, upperboundjump
    jump := indexjump * jumpby;
    Rwing1R, Rwing1I, Rwing2R, Rwing2I :=
      FOR x IN 0, off - 1
      p1 := x + jump; p2 := p1 + off;
      W := pi * REAL(x) / REAL(off); cosine, sine := COS(W), SIN(W);
       Lwing1R, Lwing1I, Lwing2R, Lwing2I :=
              Areal[1, p1], Aimag[1, p1], Areal[1, p2], Aimag[1, p2];
      realm, imagm := Lwing1R - Lwing2R, Lwing1I - Lwing2I;
      RETURNS     ARRAY OF Lwing1R + Lwing2R
                  ARRAY OF Lwing1I + Lwing2I
                  ARRAY OF realm*cosine + imagm*sine
                  ARRAY OF imagm*cosine - realm*sine
      END FOR;

% Error spot: The focus is on concatenations
    groupR := Rwing1R || Rwing2R;    % This creates error in cc
    groupI := Rwing1I || Rwing2I;

    % The inexplicable solution:
    %    groupR,groupI :=  for kk in 0, 2*off - 1
    %                      grR, grI := if kk < off then Rwing1R[kk],Rwing1I[kk]
    %                                  else Rwing2R[kk-off],Rwing2I[kk-off] end if;
    %                      returns array of grR
    %                              array of grI
    %                      end for;
    %
    % The drawback here is that one needs to know the actual array size of
    % "groupR" and "groupI" ie 2*off - 1

    RETURNS       VALUE OF CATENATE groupR
                  VALUE OF CATENATE groupI
    END FOR;
  IN AR1, AI1
  END LET
end let
end function
```

(ii) The SISAL code

Figure 2: A bug in OSC

3. Normalisation of Parallel Loops

In the initialisation section of the weather simulation implementation in SISAL [1], loops of similar loop bound are forced to be coupled together in order to be accepted and pass through the OSC compiler. The full code, which is available on request, belongs to older versions of the initialisation routines, but adequately exhibits the fault.

The focus of this example is in the calculation of the variance "var" and the average potential height "h". Figure 3(ii) is an extract of the code producing the error .

In attempting to simplify the program in order to narrow the scope to isolate the source of error, the problem disappears. This suggests that the "complexity" of the program could be a factor.

When these two statements are listed separately in the program as shown, IF1OPT fails, giving the error message shown in Figure 3(i). This seems to suggest that "Graph Normalisation" is incomplete within IF1OPT [4].

A way to get around this problem is to "couple" loops of similar loop bound together as shown in Figure 3(iii).

```
osc main.sis -IF1 -double_real
LL Parse, using binary files
* Reading file: main.sis...

version 1.8     (Mar 28, 1989)

 accepted
  226 lines in program
 0 errors ( calls to corrector)
 0 tokens inserted;   0 tokens deleted.
 0 semantic errors
osc -v -o prefft main.if1 IntrFuncs.if1 complex.if1 Inital.if1
                    InitFFT.if1 gaussg.if1 legendre.if1 SasAlfa.if1
if1ld -o main.mono -e main main.if1 IntrFuncs.if1 complex.if1
        Inital.if1 InitFFT.if1 gaussg.if1 legendre.if1 SasAlfa.if1
if1opt main.mono main.opt -l -e

if1opt: E - FORALL RETURN SUBGRAPHS NOT NORMALIZED

** COMPILATION ABORTED **

*** Error code 1
stop.
```

(i) Error message for the subgraph normalisation error

```
var :=  FOR diffindex IN 2, jxmx
        RETURNS VALUE OF SUM CabsSqr(zt_mountain[diffindex])
        END FOR;
h :=    FOR index IN 1, jxmx
        RETURNS ARRAY OF Crmul(constant, zt_mountain[index])
        END FOR;
```

(ii) The error producing region in the initialisation section

```
var, h :=   FOR index IN 1, jxmx
            RETURNS VALUE OF SUM IF index = 1 THEN 0.0
                                 ELSE CabsSqr(zt_mountain[index])
                                 END IF
                    ARRAY OF Crmul(constant, zt_mountain[index])
            END FOR;
```

(iii) The immediate solution

Figure 3: Subgraph Normalisation error

4. Exploitation of Parallelism for Conventional Multiprocessors

OSC only exploits parallelism from parallel FOR loops. There are some instances in a program where two big blocks of mutually independent sequential loops should be (able to be) processed concurrently. This occurs in the SISAL implementation of a two dimensional FFT [3]. Unfortunately, owing to the inability of OSC to identify the data independency of the two loops, the result is a much degraded speedup. The problem appears to be trivial but is not.

5. Cost Estimation Routine

The cost estimation routine of SISAL fails to identify the critical path significance of certain parallel loops, as a results these loops are not sliced accordingly. The problem and a quick solution using a mickey mouse Quasi Doubly Nested technique are elaborated in [2].

In the issue of cost estimation of the OSC, there are a few points that need to be raised. The initial findings from the implementation of the weather simulation model indicate that the compiler fails to slice low complexity singly nested parallel loops which reside in the highly parallel critical path of the program i.e. the timeloop. A quick solution using the QDN technique to "trick" the cost estimator is:

```
FOR array RETURNS VALUE OF CATENATE
    FOR array RETURNS ARRAY OF
            xxxxxxx
    END FOR
END FOR.
```

Our initial arguments in [2] were not complete due firstly to the lack of knowledge on how the cost estimates were performed at compilation time. Additionally, at that stage we were not aware of the significant inefficiency of concatenation operations in a parallel loop and hence we skipped commenting on the incomplete parallelism shown on the QDN concurrency profile, and the incompatibility between the concurrency profile obtained (~60%) with 16 processors and the achieved speedup (~3) over the single processor performance.

The cost estimates are performed relying on the number of loop iterations, I, and the complexity of the loop body. The H cost parameter instructed at compile time is the total cost of the loop, below which loop slicing will not be performed. The parameter L is the depth of the nested loops that the compiler is instructed to consider slicing. So only these factors are known to the compiler to estimate the costs of slicing. Once the slicing has been performed, the slice templates are *superficially* fixed. It is then up to the application users to increase the problem size to stuff the templates full to maximise the "actual work performed in a task"/"work required to create the task" ratio if the user finds that good speedup can be obtained using multiple processors to share the workload. We presently do not know how to determine and parameterise the overheads imposed by the OSC runtime system in making decisions relying on the other parameter i.e. the number of processors sharing the work which is specified at the beginning of the run. But our experience in implementing the weather model and the two dimensional FFT model shows that the overheads may be significant.

Presently we also do not know if problem size, known at compile time, which indirectly determines the number of loop iterations, I, has been employed effectively as a parameter for cost estimates. Nonetheless, it is definitely cost saving if the cost estimation is performed by also considering the number of processors used, since in practice one may like to use a fix number of processors. This parameter could be usefully included as a pragma at compile time. Hopefully the cost saving from subsequent reduction of runtime overhead will result in significant performance improvement in large application programs of the types we are studying.

6. Eager Memory Deallocation Routine

The runtime system allocates storage for the initialised data at the beginning of a sequential loop, but it also *eagerly* deallocates, not concurrently with the main computation of the loop body, the storage at the end of each loop before the loop repeats itself. For the weather simulation model, the deallocation time constitutes approximately 28% of the total loop time. The elaboration of this problem and a brief proposal of a solution can be found in [2]. It should be possible using code motion and data structure pointer reassignment to remove the allocation and deallocation of fixed size data structures from within iterative loops [3]; the appropriate optimisation by hand at the C level is relatively easy to perform for simple examples. Where the data structure size cannot be determined statically, data deallocation should be overlapped with the main computation of the loop body i.e. *lazily* [2].

Also documented in [3] is a mathematical analysis for the upper bound performance of, seen from the source level, a supposedly parallel SISAL code:

```
FOR row IN 0, totalN
RETURNS ARRAY OF FFT[row]
END FOR          % where each FFT is potentially sequential
```

Proven by experimental results, the analysis shows that depending on the size of the loop body relative to those of the allocation and deallocation routines, the speedup curve for the code can saturate dramatically at an unexpectedly low value. This further presents the need for improvement in the memory allocation and deallocation scheme implemented in OSC on the ENCORE.

7. Debugging SISAL Programs

As well as having to rely on the FORTRAN weather code, which was not well written, the immediate problem in the direct transliteration process was the lack of debugging support at the SISAL source level. It is to date impossible to debug a SISAL program at its source level. The best possible debugging tool available is DI, the Dataflow Interpreter, which interprets IF1 graphs. Unfortunately, even DI as a debugger had bugs which created problems in producing results from multiple-nested sequential loops (from loop forms A and B) [5]. Program debugging at the C code level is sometimes useful too except that the C code generated from SISAL must be assumed as perfectly correct, which is not always true!

The correctness of a focused variable, whose value alters as it undergoes changes in different program state, can only be checked by making it a function parameter or result. While results of functions are readily available using DI, intermediate values are extremely difficult to obtain without compromising the structure integrity of the SISAL source; it is necessary to create a function boundary around the variable to be investigated. The values are then compared with the output for the changes in state of the variable which were effortlessly obtained from FORTRAN by an additional "print *, *variable name*" statement in the FORTRAN code. This *indirect* debugging in DI is both difficult and unreliable and requires additional lengthy, tedious and error prompt efforts. One not only has to investigate program correctness as originally intended, but also has to deal with correctness of the additional functions created and always beware of the integrity of the interpreter for complicated programs (Hiesenberg effect). This is the most serious drawback which discourages anyone from doing serious programming in SISAL. Research into source level debugging aids for SISAL is therefore needed. This research may not be attractive, yet the reality is that few large application codes are correct by design and even less codes work first time.

8. Language Support for Complex Numbers

Computations involving complex numbers are common in scientific applications; FORTRAN recognises this. As SISAL presently does not provide an implicit structure for complex numbers, they are usually expressed as a record of two numbers representing the real and imaginary parts, and an array of complex numbers is expressed as an array of records. Even though OSC performs a record fission optimisation at compile time, the additional subgraphs of functions for arithmetic on complex numbers serve as a complication which might have contributed to the "Normalisation" error described above. In most cases, particularly when complex arithmetic constitutes a major part of a program, the explicit tasks in the treatment of complex numbers as records may result in an additional execution cost. The alternative representation is to express a complex number as two separate numbers, and then an array of complex numbers as two separate arrays of numbers. This too may result in an additional execution cost.

The remedy is to implement a SISAL language support for complex numbers similar to FORTRAN's, making treatment of complex numbers implicit. This will remove the necessity of building records, extracting elements from records and calling functions for complex arithmetic which serve only to obscure the underlying algorithm.

9. OLD Statements: an Easy Mistake

Sequential loop constructs are associated with the use of **OLD** statements. As **OLD** is used on the right hand side, multiple accesses of an **OLD** variable are common place. So errors due to the coexistence of the variables evaluated in the present iteration and the **OLD** variables evaluated in the previous iteration can occur easily in multiple nested sequential loops. A particular example is in the sequential loops in the function LEGENDRE, where once the **OLD** statement is missed out, the error is very difficult to detect.

Conclusions

In this document we have presented some of the issues associated with the SISAL and OSC from a user's view point. While a number of these problems are newly discovered, it is possible that others may have been solved in the new release of OSC. Many of the problems associated with these issues may be resolved readily while others require substantial effort such as debugging tools.

Acknowledgements

We would like to thank all members of the project and in particular Warwick Heath, for contributing to some of the issues presented here. The research was supported in part by the CSIRO Division of Information Technology and the Royal Melbourne Institute of Technology.

References

[1] Pau S. Chang and Greg K Egan, "An Parallel Implementation of a Barotopic Spectral Numerical Weather Prediction Model in the Functional Language SISAL", SIGPLAN Notices, Vol. 25, No. 3, March, 1990, pp. 109-117, Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming PPoPP, Seattle, Washington, March 14-16, 1990.

[2] Pau S. Chang and Greg K. Egan, "Performance Evaluation of a Parallel Implementation of Spectral Barotropic Numerical Weather Prediction Model in the Functional Dataflow Language SISAL", (TR118 091 R), Technical Report 31-006, Laboratory for Concurrent Computing Systems, School of Electrical Engineering, Swinburne Institute of Technology, Version 1.0, 2/10/89.

[3] Pau S. Chang and Greg K. Egan, "Analysis of a Two Dimensional FFT Implementation in SISAL", Technical Report 31-015, Laboratory for Concurrent Computing Systems, School of Electrical Engineering, Swinburne Institute of Technology, 1990.

[4] David C. Cann, "Compilation Techniques for High Performance Applicative Computation", Technical Report CS-89-108, Colorado State University, May 10, 1989.

[5] Steve Skedzielewski and John Glauert, "IF1 An Intermediate Form for Applicative Languages", M-170, Lawrence Livermore National Laboratory, July 1985.