LABORATORY FOR CONCURRENT COMPUTING SYSTEMS

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

Proceedings of the Australian SISAL Workshop 1990

Technical Report 31-016

Compiled by G.K. Egan

Abstract

A Workshop was held in May 1990 to establish an Australian position on the future development of SISAL. This report contains the material tabled and presented at the workshop and the issues arising from the open session at the end of the workshop.

SWINBURNE INSTITUTE OF TECHNOLOGY

LABORATORY FOR CONCURRENT COMPUTING SYSTEMS
SCHOOL OF ELECTRICAL ENGINEERING

SISAL WORKSHOP
The Australian Position on Sisal Futures

Purpose:

To establish an Australian position for the future development of Sisal to be presented at the Asilomar International Sisal Workshop June 1990. Researchers interested in the development of concurrent computing systems and their associated languages are welcome to attend.

Venue:

Swinburne Institute of Technology
Council Chambers
5th Floor Library Building
Tuesday 8th May 1990

Laboratory for Concurrent Computing Systems at Swinburne

9.30 - Directions (Prof. Greg Egan) short statement
9.45 - CSIRAC II (Prof. Greg Egan/Dr. David Abramson) short statement
10.00 - IF1/IF2 translators for CSIRAC II (Neil Webb)
10.45 *Morning Tea*
11.00 - User experiences : weather and fft codes (Pau Chang)

High Performance Computer Systems in CSIRO

11.45 - Directions (Dr. David Abramson) short statement
12.00 - Sisal related research (Dr. Abramson)
12.45 *Lunch*

University of Adelaide

1.30 - Directions (Dr. Andrew Wendelborn) short statement
1.45 - Sisal on the Encore and Leopard Multiprocessors (Hugh Garsden)
2.30 - Sisal 2.0 Critique (Dr. Andrew Wendelborn)
3.15 *Afternoon Tea*

Open Session

3.30 - ranking of Sisal issues (Dr. Andrew Wendelborn,Chair)
4.30 *close*

Please advise your intention to attend to the School of Electrical Engineering (03) 819 8516 ASAP so that catering may be arranged.

Participants:

Dr D. Abramson

Senior Research Scientist
CSIRO
55 Barry St.,
Carlton
phone: (03) 660-2095 fax: (03) 662-1060

rcode@koel.co.rmit.oz.au

Pau Chang

Swinburne Institute of Technology
Laboratory for Concurrent Computing Systems
P O Box 218
Hawthorn 3122

pau@stan.xx.swin.oz
rcocp@koel.co.rmit.oz.au

Grant Colling

Computer Systems Officer
Computer Science Department
Swinburne Institute of Technology
P O Box 218
Hawthorn 3122
phone: (03) 819-8670 fax: (03) 818 3645

grant@saturn.cs.swin.oz.au

Antonio Cricenti

Lecturer
Swinburne Institute of Technology
P O Box 218
Hawthorn 3122
phone: (03) 819-8322

Russell Dawe

Faculty of Engineering
Laboratory for Concurrent Computing Systems
Swinburne Institute of Technology
P O Box 218
Hawthorn 3122
phone: (03) 819-8733 fax: (03) 818-3657

russell@saturn.cs.swin.oz.au

Greg Egan

Professor of Computing Systems Engineering
and
Director
Laboratory for Concurrent Computer Systems
Swinburne Institute of Technology
P O Box 218
Hawthorn 3122
phone: (03) 819-8167

gke@stan.xx.swin.oz.au

Lindsay Errington

Department of Computer Science
University of Adelaide
GPO Box 498
Adelaide 5001

lindsay@cs.ua.oz.au

Rhys Francis

La Trobe University
Bundoora 3083
phone: (03) 479-2504

rhys@latcsl.oz.au

Ivan Francis

The Murdoch Institute
Royal Children's Hospital
Flemington Rd.,
Parkville 3058
phone: (03) 345-5045

Hugh Garsden

Computing Officer
University of Adelaide
North Terrace
Adelaide 5000
Phone: 228-5763

hugh@cs.ua.oz.au

Michael Klein

Swinburne Institute of Technology
School of Electrical and Electronic Engineering
P O Box 218
Hawthorn 31222
phone: (03) 819 8612

Dragi Klimovski

Lecturer
Swinburne Institute of Technology
P O Box 218
Hawthorn 3122
phone: (03) 819-8322

C.S. Lee

Senior Lecturer
Schoolof Electrical Engineering
Swinburne Institute of Technology
P O Box 218
Hawthorn 3124
phone: (03) 819 8316 fax: (03) 819-6443

Adam McKay

Research Student
RMIT / CSIRO
phone: (03) 660-2726

asm@goanna.cs.rmit.oz.au

Mark Rawling

Experimental Scientist
CSIRO
55 Barry St., Carlton
phone: 660 2726

rcomr@koel.co.rmit.oz.au

Simon Wail

Postgraduate student
RMIT
Department of Communications and Electronic Engineering
124 LaTrobe St.,
Melbourne 3000
phone:660 2726 fax: 662 1060

rcosw@koel.co.rmit.oz.au

Neil Webb

Development programmer
Computer Power Group
CP Software
616 St Kilda Rd.,
Melbourne 3004

njw@bohra.cpg.oz.au

Andrew Wendelborn

Senior Lecturer
Department of Computer Science
University of Adelaide
GPO Box 498
Adelaide, 5001

andrew@cs.ua.oz.au

Paul Whiting

Experimental Scientist
CSIRO - DIT
C/o RMIT
Department of Communication Engineering
124 La Trobe St.,
Melbourne 3000
phone: (03) 660-2726 fax: (03) 662-1060

rcopw@koel.co.rmit.oz.au

SISAL Futures

The following issues, set out in point form, were raised in the Open Session at the end of the Workshop; many of these points are elaborated in the presentations.

Infrastructure for Intergroup Communication

Mail group

It is clear that international interest in SISAL is growing. Better communication is needed to facilitate a better view of where SISAL research is going. Suggest a SISAL mailing list with connecting all sites.

Integration of improvements/bug fixes

It is becoming more difficult to identify what the "latest" version of SISAL is. With a proliferation of local versions, not all bug fixes/enhancements are making it into new releases. Possible solutions are a single site responsible for integration of releases or ftp access to local versions.

Documentation

There is a general view that our best source of documentation ends up being the source code. The view is that IF1 documentation is old and imprecise and that the specification of IF2 does not reflect current usage. IF1/2 pragmas, while being regarded as necessary, are often undocumented and possibly have different interpretations at different sites. Good documentation is a necessary springboard for future SISAL research.

Sisal 1.2/OSC

Maintenance

Recognition should be given to the existing user community otherwise they may lost as users of Sisal 2.0.

OSC is seen as the best current tool for doing realistic applications work, and provisions should be made to maintain Sisal 1.2/OSC support.

Efficiency

The current mechanism whereby structures are allocated and deallocated wthin while loops may be replaced by double allocation outside the loop and a pointer exchange within the loop for fixed size structures (the most common case). The current scheme leads to unacceptable performance on real codes with large structures [Chang, Garsden].

Overlapping deallocation with execution may give improvement where the structures are not of fixed size but cache cycling and tag pool fragmentation needs further study [Garsden].

OSC cost estimation does not currently detect critical path regions and unravel them; changing the cost pragma globally causes over slicing. Although there are philosophical arguments against it, a compile time pragma specifying the number of processors may permit finer tuning [Chang].

Functionality

The partial implementation of streams in Sisal 1.2 is hindering some studies where streams would be the data type of choice including applications involving continuous processes (signal processing) or applications where pipelining is a principal source of concurrency.

Known bugs

e.g. coalescing of loops of different bounds leads to normalisation errors. If the same loops are merged at source level the problem does not occur [Chang].

Sisal 2.0

Timetable for implementation is important

Recognition of Sisal 1.2 users
- release subset of Sisal 2.0 early
- provide mappings from Sisal 1.2 to Sisal 2.
- module compatibility with Sisal 1.2 (in addition to FORTRAN etc.)
- some caution on language extensions is suggested as over imbelishment may intimidate prospective users.

Impact on IF1/IF2
- will redefinition be necessary?
- Multiples (definition)
  - useful abstraction at IF1 level, but do they hinder implementation?

Function/region level concurrency
- e.g. 2 large, independent serial loops which could be executed in parallel do not

Streams (confidence of implementation given 1.2)

Complex numbers

Vector/Matrix arithmetic

I/O
- real input/output interface required
- fibre is inadequate
- file support
- hooks to real-time I/O devices
- multiple I/O sources resource managers

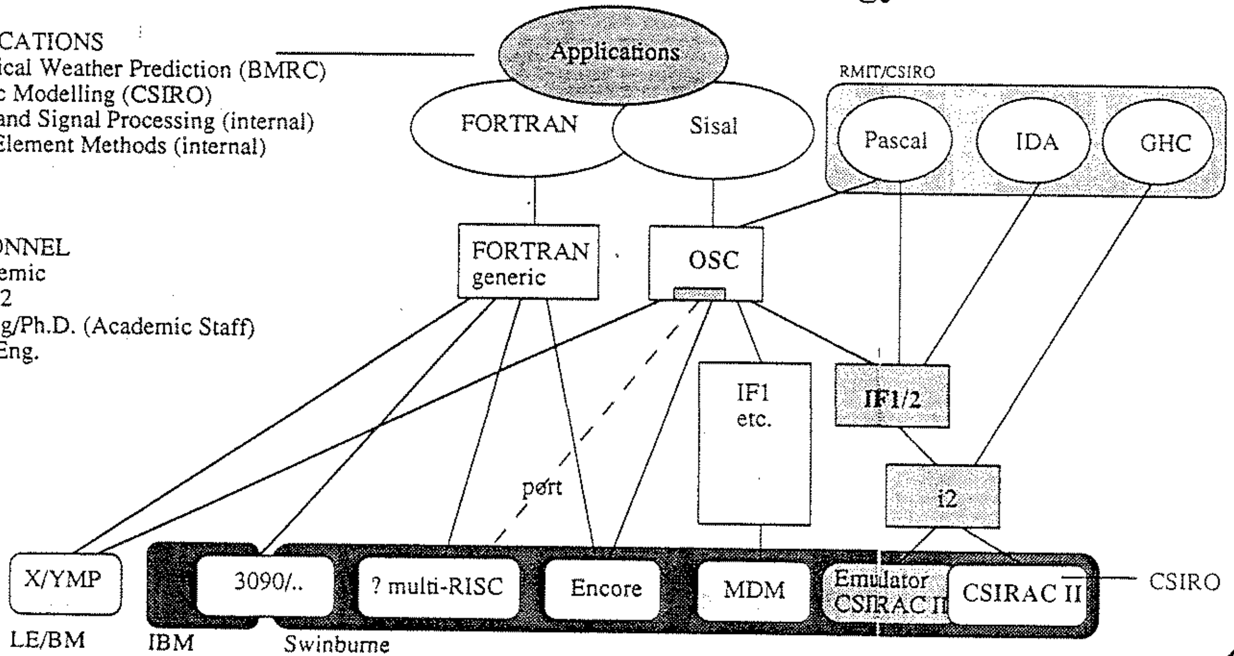
Compiled from open session notes by G.K. Egan & A.L. Wendelborn

Laboratory for

Concurrent Computing Systems at Swinburne

Directions - a short statement

Prof. Greg Egan


Laboratory for Concurrent Computing Systems
Swinburne Institute of Technology
APPLICATIONS
Numerical Weather Prediction (BMRC)
Seismic Modelling (CSIRO)
Image and Signal Processing (internal)
Finite Element Methods (internal)
PERSONNEL
2 Academic
3 CSO/2
2 M.Eng/Ph.D. (Academic Staff)
2-4 M.Eng.
Applications
FORTRAN
Sisal
RMIT/CSIRO
Pascal
IDA
GHC
FORTRAN
generic
OSC
IF1
etc.
IF1/2
i2
port
X/YMP
3090/..
? multi-RISC
Encore
MDM
Emulator
CSIRAC II
CSIRAC II
CSIRO
LE/BM
IBM
Swinburne

Notes

CSIRAC II Dataflow Machine

will maintain and distribute the following:

IF1/2 translator
- i2 assembler
- CSIRAC II emulator/simulators

committed to completing CSIRAC II prototype

will maintain Manchester suite for cross comparison

access to other groups welcome

OSC

will port OSC to Laboratory's multi-RISC processor

identify performance limiting factors collaborate with compiler groups

- memory allocation
- runtime overheads

Application Studies

Cross comparisons with FORTRAN including Encore EPF principally:

- Numerical Weather Prediction on next generation operational code with full physics and data under Bureau of Meteorolgy Research Centre collaboration

- seismic modelling (coal mines) under collaboration with CSIRO Geomechanics

- Signal processing internal and with Adelaide

- Finite Element Modelling internal

Laboratory for

Concurrent Computing Systems at Swinburne

IF1/IF2 translators for CSIRAC II

Neil Webb

IF1 / IF2 Translator for the CSIRAC II Dataflow Computer

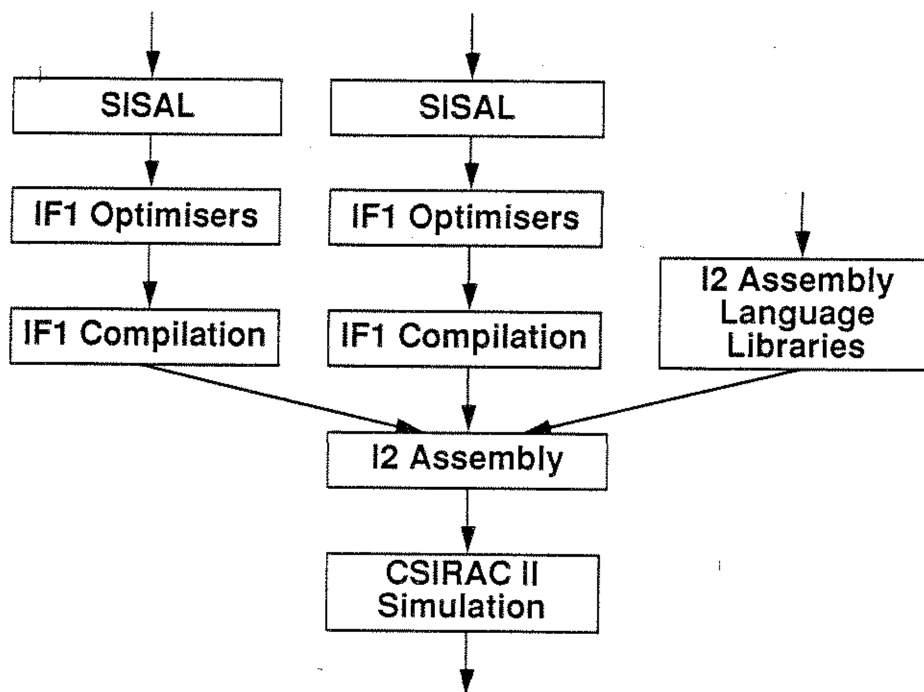
Swinburne Institute of Technology

May 8, 1990

Topics of Interest

- SISAL and CSIRAC II Compilation
- IF1 Compiler Facilities
- Transmitted vs Stored Arrays
- Multi-Language Support
- IF2 Implementation Design
- Some IF1 Difficulties
- Bugs and Beetles
- Operational Comparisons
- CSIRAC II IF1 / IF2 Futures...

SISAL & CSIRAC II Compilation



IF1 Compilation

Execution Facilities:

- Non-strict function and loop boundaries
- Strict evaluation of if-then-else operations
- Serial Loops interate in a single context
- Rapid literal propagation and context creation
- Low parallel loop initiation costs
- Integer, Real, Double Real, Char, Boolean
  scalar data types
- Array and Record data structures
- Transmitted or Stored arrays
- Internal parallel result formatting
- Structural support for error conditions

Transmitted or Stored Arrays

- Compilation directive to select transmitted or stored arrays
- Completely separate execution systems
- Transmitted structures are strict, direct access objects with intrinsic length and lower bound
- Stored structures are non-stict, indirect access objects with explict length and lower bound via a *dope* vector
- Powerful structure initialisation (sbf) and data copy (sbc) instructions are exploited
- Stored structures are *not* disposed

Multi-Language Support

- Support for multiple languages to be used in the same program is provided.
- Supported languages already include:
  - SISAL
  - IdA
  - Pascal
- Other languages that compile to IF1 and do not require special assistance from the IF1 compiler (like IdA) are automatically supported
- The CSIRAC II Pascal compiler's internal support functions (write, "set" operations, etc) are written in SISAL

IF2 Facilities

- Support for IF2 is not yet available
- "D"ependance edges and IF2 data structures are already incorporated
- IF2 function "hooks" are in place
- Preliminary IF2 Node operations design and implementation is proceeding
- IF2 Pragmas are supported (where known)
- New structure stored allocation / disposal required

No difficulties are forseen to hinder a clean implementation

IF1 Difficulties

"Less than Ideal" areas of IF1 include:

- Multiples

  IF1 does not assume sequential operation but other approaches seem to be expensive and strict

- Reductions

  Like multiples, these are serial by nature (see IF1 Futures)

- Vector operations

  Not available with comprehensive analysis

Bugs and Beetles

- The support programs and documentation of SISAL and IF1 are of varying quality.
- Classic examples:
  - SISAL does not use more than 2 arguments to an ACatenate node. Also, SISAL catenates these arguments as a list rather than a tree
  - Code improvers often "eat" correct, non-SISAL originating IF1 programs
  - Documentation is faulty and out-of-date. Over a dozen new IF1 and IF2 nodes have been added since the last IF1 or IF2 document was released

Operational Comparisions

	Manchester DFM			CSIRAC II		
	Critical Path	Aver. Parallel	Total Nodes	Critical Path	Aver. Parallel	Total Nodes
Loop1	48	11.9	573	61	9	571
Loop2	124	6.5	810	66	5	352
Loop3	59	5.1	303	59	5	279
Loop4	102	220.9	22533	52	12	616
Loop5	184	13.8	2537	155	10	1569
Loop6	181	9.6	1737	156	11	1782
Loop7	53	15.8	840	52	14	751
Loop8	113	37.5	4238	100	21	2068
Loop9	78	16.6	1297	40	8	301
Loop10	95	88.1	8374	79	135	10636

CSIRAC II IF1 / IF2 Futures...

There are several major areas of investigation and implemtation for the CSIRAC II IF1 Compiler

These areas include:

- Implement Unions and Tagcase (design complete)
- Implement a tail-recursive Iterate node
- Implement the If-Then-Else node
- Design a new structure store memory manager. Give special consideration to the speed of allocation vs memory fragmentation
- Implement the IF2 nodes and Reference Counting
- Design an error interface for array (vector) operations

CSIRAC II IF1 / IF2 Futures...

(continued)

- Incorporate the new ForAll reductions that exploit the Structure Store. This will remove long "tails" from parallel loops - especially FinalValue operations
- Add automatic throttling codes to ForAll loops and Call operations
- Add AllButLast to IF1 Compiler

Laboratory for
Concurrent Computing Systems at Swinburne

User Experiences: Weather and FFT codes

Pau Chang

Laboratory for Concurrent Computing Systems
Swinburne Institute of Technology

My Experience and Research in SISAL

Pau S. Chang

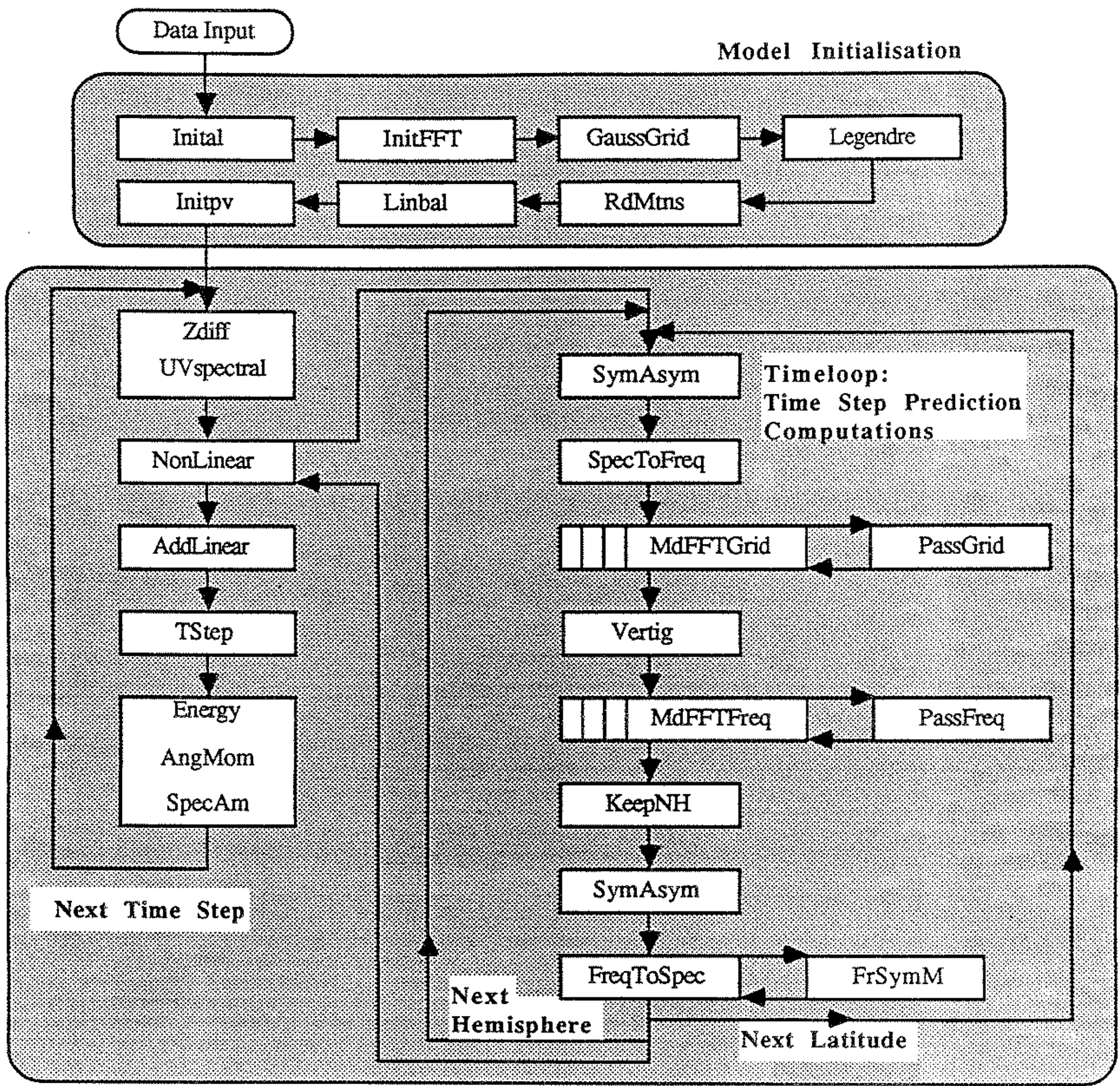


Figure 3: Flow chart for the FORTRAN and sequential SISAL implementations

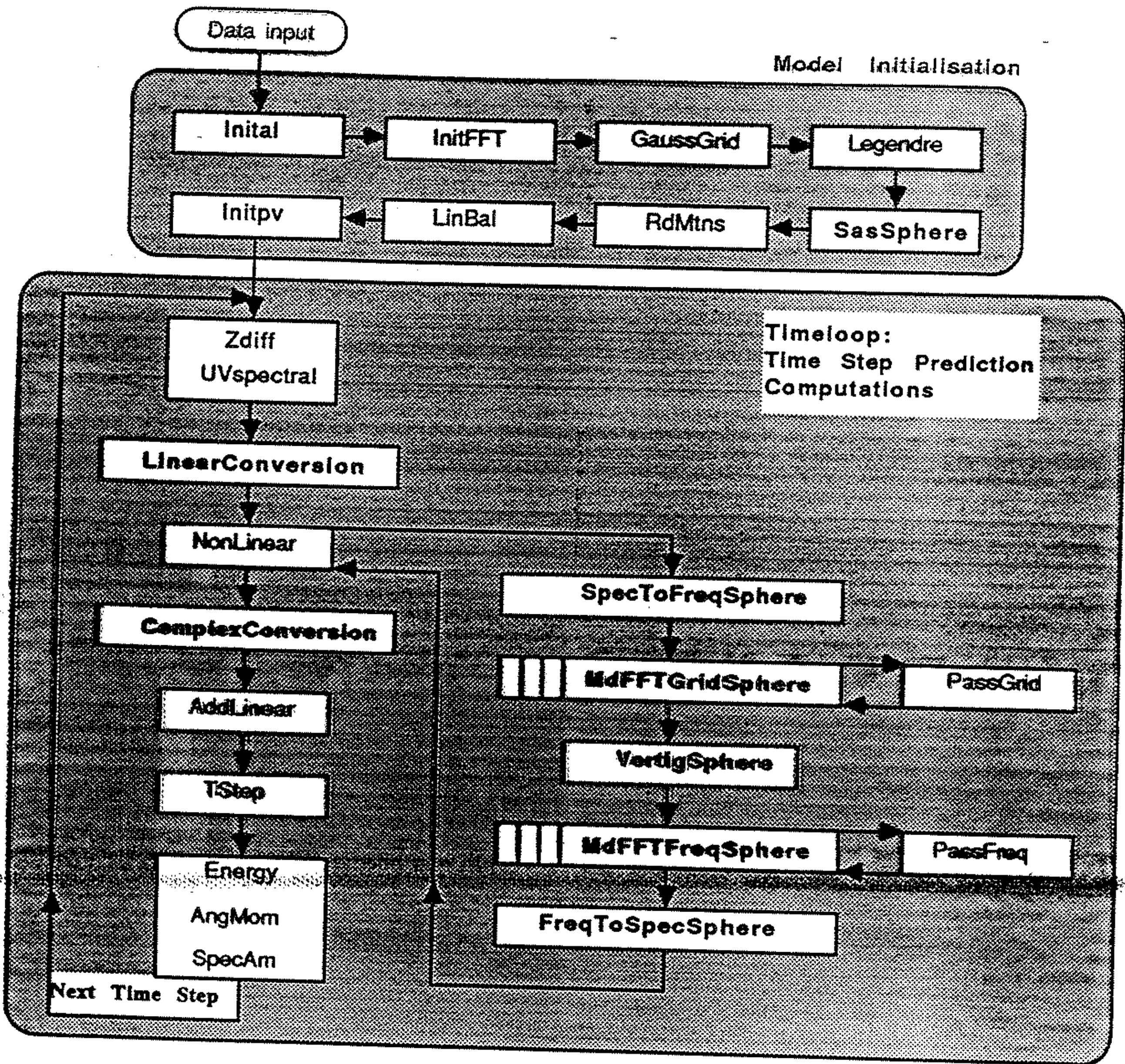


Figure 7: Flow chart for parallel SISAL implementation

Results for the Parallel Implementation

- Single processor runtime: 106.7 seconds

- multiple processors: run time reduced to 13.7 seconds

- Parallelisation of the timeloop body has been successful confirming the feasibility of a parallel implementation of the adopted weather model

Benchmarking of Timeloop

For model size J=30, timestep = 30 minutes; 24-hour forecast needs 48 iterations of timeloop. Timeloop critical and dominant.
----> One iteration of timeloop is sufficient and adequate for benchmarking

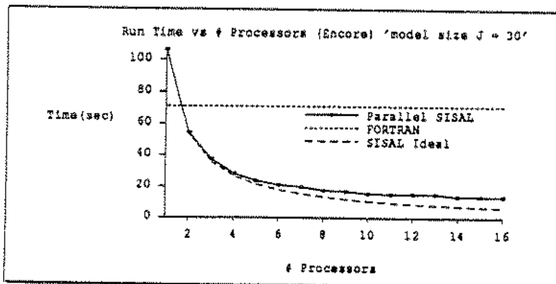


Figure 8a: Execution time profile of the new implemen[illegible]n

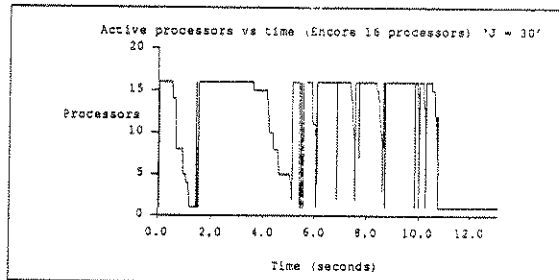


Figure 8b: Concurrency profil[illegible] the new implementation

Performance in terms of Model Sizes

- Curves approximately proportional to $\mathbf{J^2}$

- Single pocessor runtime of parallel implementation in SISAL very close to the sequential implementation in FORTRAN

- SISAL/multiple processors: **the growth in execution time with increasing model size** is much **slower** than that for single processor runtime for either FORTRAN or SISAL

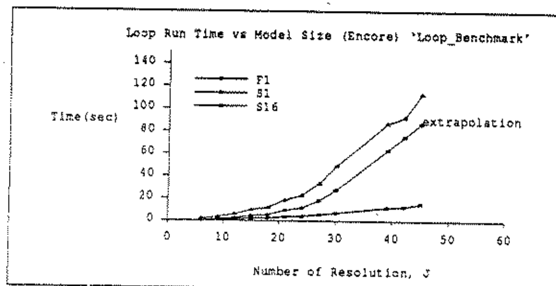


Figure 9: The execution time of the FORTRAN and SISAL implementations as a function of model size.

Speedups from the Benchmark Ratios

- **S1/F1** curve - speedup of the FORTRAN implementation over the SISAL implementation for varying model size

- **F1/S16** curve - speedup of the SISAL implementation in a multiprocessor (16) environment over a sequential implementation in FORTRAN (single processor)

- **S1/S16** curve - speedup of SISAL implementation in multiprocessor (16) environment over the execution time of the same task by a single processor

- **S1/Sn** curves for various model sizes: general view

<u>OSC runtime overheads</u>
actual computation

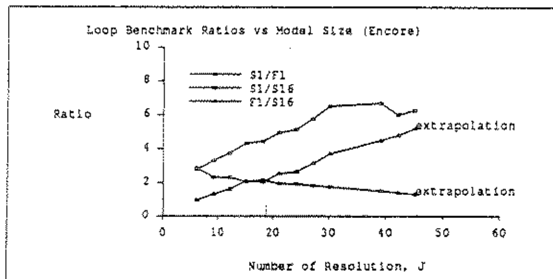


Figure 10: The benchmark ratios as a function of model size.

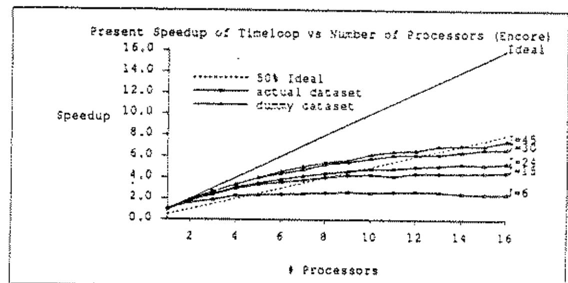


Figure 11: The speedup profile of the timeloop for the present implementation

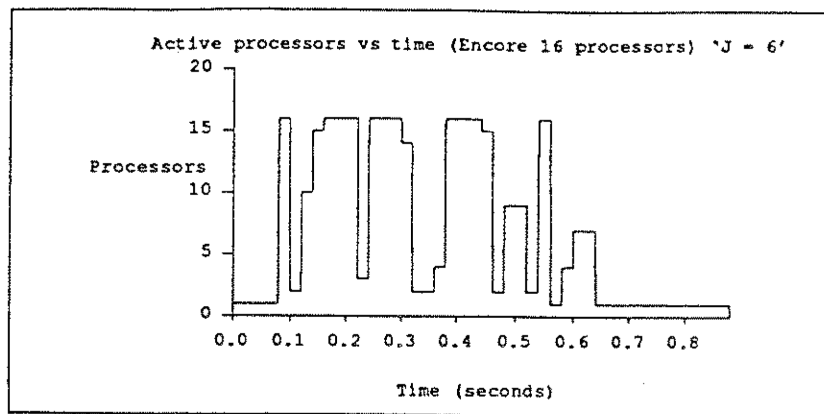


Figure 12a: Concurrency profile of timeloop for J = 6 (small model size)

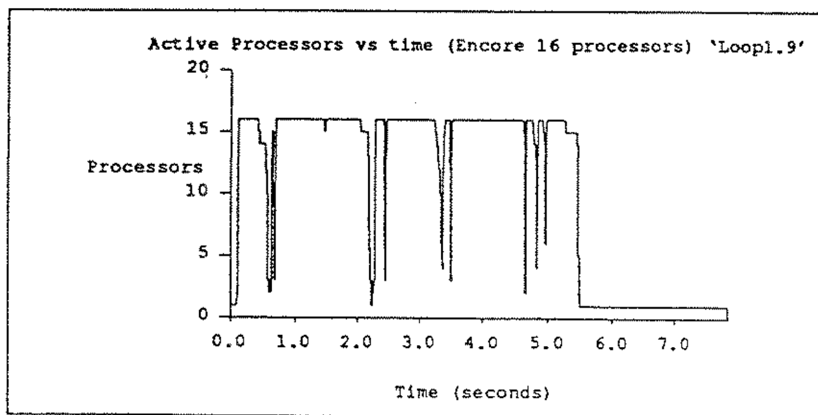


Figure 12b: Concurrency profile of timeloop for J = 30 (large model size)

Effect of Memory Deallocation Overhead

- Significant serial tail section - critical as more parallelism is obtained

- **eager** memory deallocation routine of OSC runtime system

- Storage structures used are automatically but **sequentially deallocated** (**28%** of loop time)

Proposal: static analysis determines that **array sizes are invariant through loop iterations** and may be **re-allocated**

\+

"lazy" deallocation in parallel with main computation only when necessary

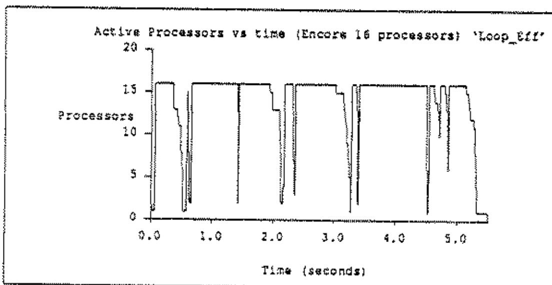


Figure 13a: The achievable concurrency ($J = 30$) of the timeloop with an efficient memory deallocation scheme.

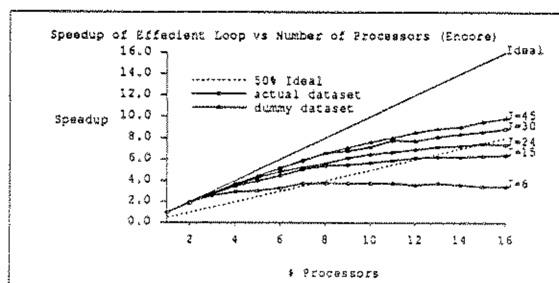


Figure 13b: The would be speedup of the timeloop with an efficient memory deallocation scheme.

SISAL Implementation of a Two Dimensional Fast Fourier Transformation Routine (technical report being prepared)

(1) Direct Transliteration from FFT in C:

```
for row in 0, totalN
FFT(rows) or FFT(columns)      (each FFT is potentially sequential)
        end for
```

SISAL does not exploit efficiently the parallelism offered by many chunks of sequential codes executed concurrently.

(2) Direct Fourier transformation approach, due to the failure in implementing the butterfly transform in SISAL.

(3) Data handbook of Am29540 FFT chip (16 points FFT) as model, I deviced my own routine to determine which points on the left should be chosen as the left wings for each right wing point of a butterfly, and also the routine to determine what W factors and when should be used. For 512 x 512 mesh, the runtime ranges from 414 seconds to 32 seconds (13 times speedup with 16 processors.

-The present one problem is that the results on the SUN are very different fron than that on the Encore. Need debugging.

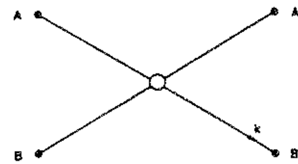
-New bugs of OSC discovered

-Analysis of memory allocation and deallocation

TRANSFORM CHARACTERISTICS

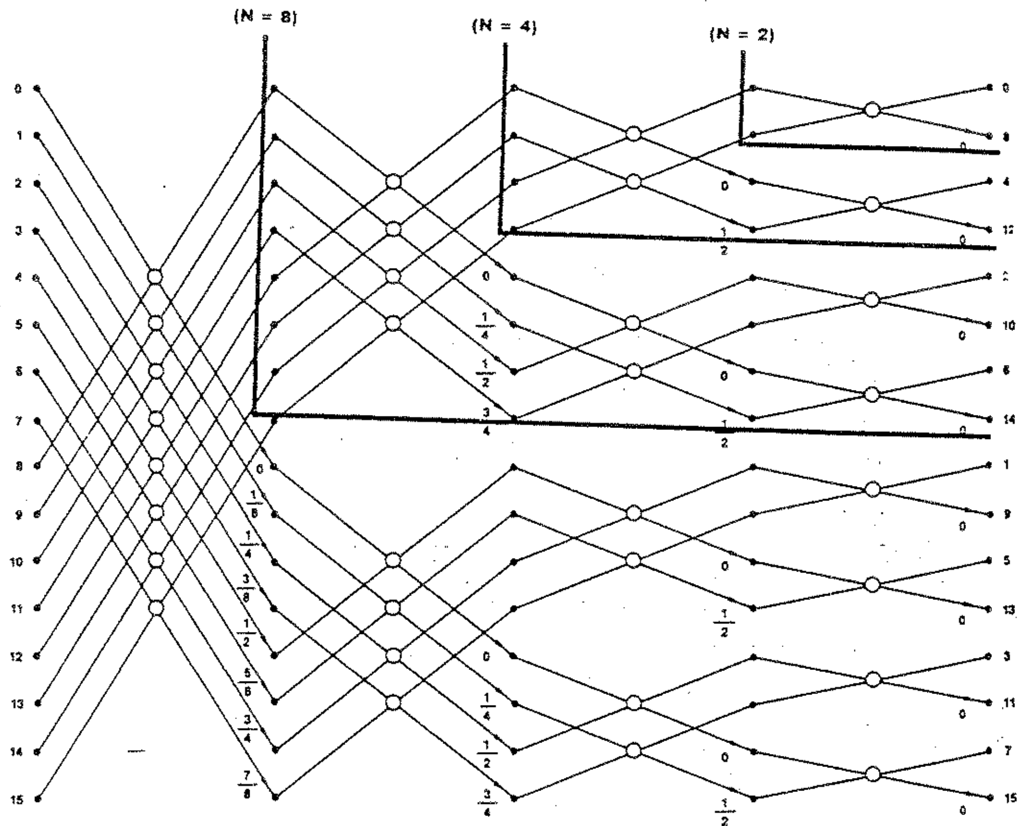
- 16-Point (N = 16)
- RADIX-2
- DIF
- Normally ordered input data (Bit-reversed output data order)
- In-place
- Complex valued input data

TYPICAL BUTTERFLY



FORWARD TRANSFORM	INVERSE TRANSFORM
$A' = A + B$	$A' = A + B$
$B' = (A - B)W^k$	$B' = (A - B)W^{-k}$

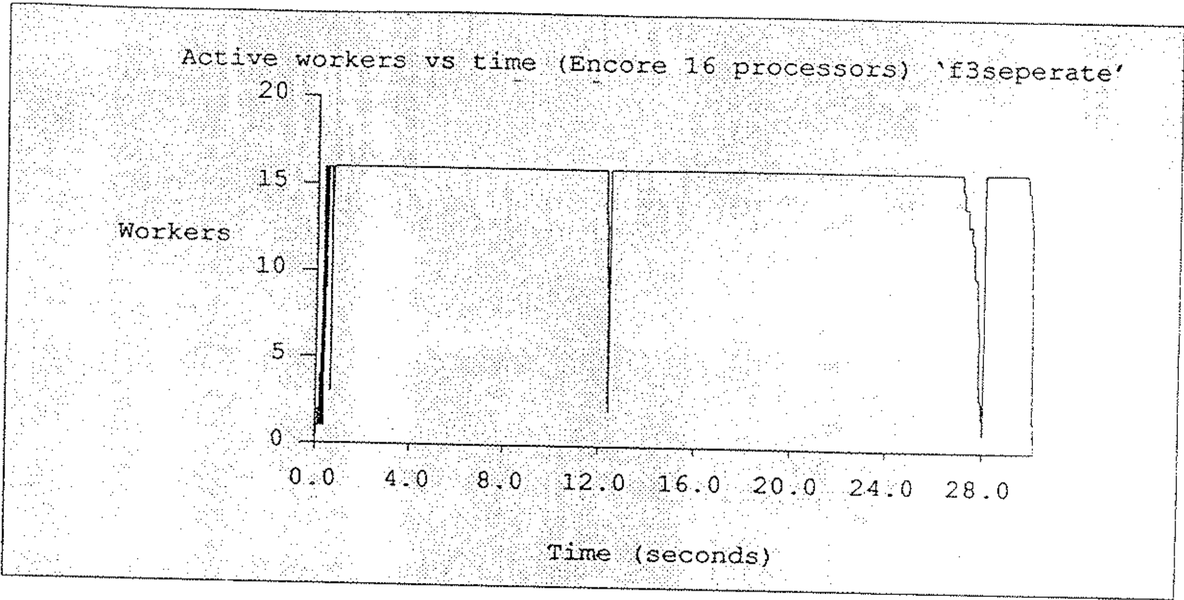
$W = e^{-j\pi}$

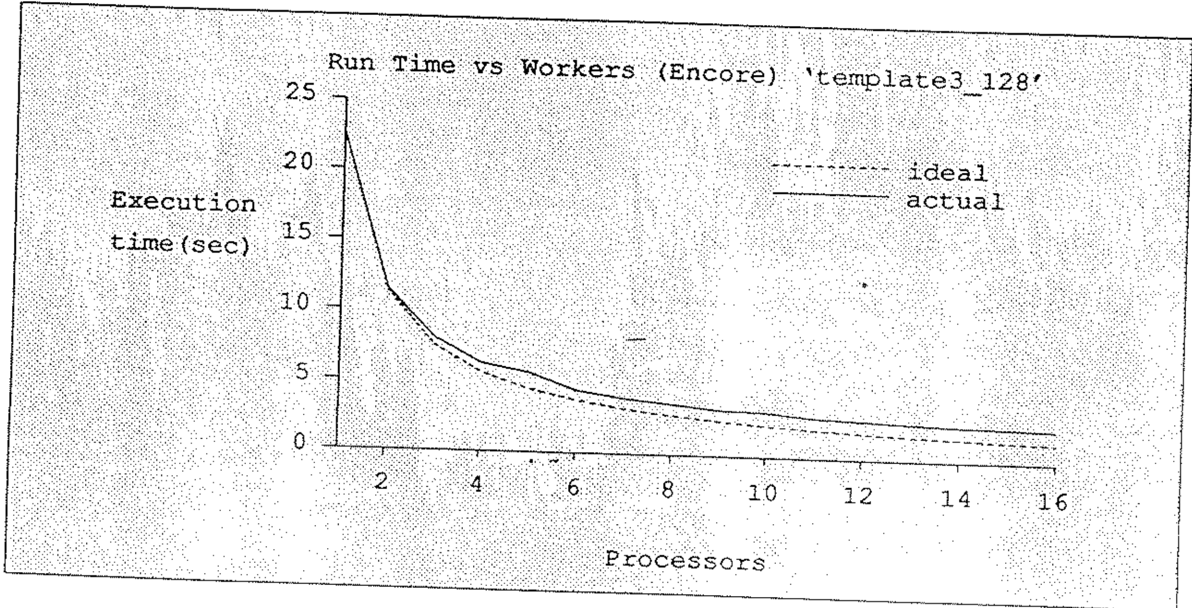


DIT/$\overline{\text{DIF}}$	$\overline{\text{PSD}}$	RADIX 4/$\overline{2}$
L	H	L

Address of	A	B	A'	B'	W^k
AS =	0	1	0	1	8

MPL-047


Active workers vs time (Encore 16 processors) 'f3seperate'
Workers
Time (seconds)
20
15
10
5
0
0.0
4.0
8.0
12.0
16.0
20.0
24.0
28.0


Run Time vs Workers (Encore) 'template3_128'
Execution time(sec)
ideal
actual
Processors
25
20
15
10
5
0
2
4
6
8
10
12
14
16

Proposals for SISAL and OSC

- efficacy in expressing the potential concurrency of scientific computational models is yet to be judged by practical application studies
- some features need added to improve SISAL's expressive capability
- improvements needed for OSC to make it more reliable and effective
-from implementations of a spectral weather simulation model and a two dimensional FFT model

---

Starting Index of "FOR array RETURNS VALUE OF CATENATE"

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

IF1 graphs - 1
Even if IF1 graphs det to 0, DI and the C code generated by OSC - 1
Need **array_setl** to set desired lower bound

FOR array RETURNS VALUE OF CATENATE of concatenations of vectors

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

```
if1ld -o chip.mono -e main chip.if1
if1opt chip.mono chip.opt -l -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 **
```

Error messages given at compile time

Normalisation of Parallel Loops

```
if1opt: E - FORALL RETURN SUBGRAPHS NOT NORMALIZED

** COMPILATION ABORTED **

*** Error code 1
stop.
```

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

Exploitation of Parallelism for Conventional Multiprocessors

Two big blocks of mutually independent sequential loops should be processed concurrently

OSC cannot identify the data independency of the two loops

Cost Estimation Routine

Fails to identify the critical path significance of certain parallel loops so loops are not sliced accordingly.

A quick solution using QDN technique "tricks" the cost estimator:

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

Number of processors sharing the work as compile time pragma - cost saving

---

Eager Memory Deallocation Routine

Single sequential loop:

- Allocate storage at beginning of loop
- *eagerly* deallocates the storage at end of iteration, serial with loop body
- ~28% of the total loop time (weather model)

SISAL Implementation of a Two Dimensional Fast Fourier Transformation Routine (technical report being prepared)

(1) Direct Transliteration from FFT in C:

```
for row in 0, totalN
FFT(rows) or FFT(columns)      (each FFT is potentially sequential)
        end for
```

SISAL does not exploit efficiently the parallelism offered by many chunks of sequential codes executed concurrently.

(2) Direct Fourier transformation approach, due to the failure in implementing the butterfly transform in SISAL.

(3) Data handbook of Am29540 FFT chip (16 points FFT) as model, I deviced my own routine to determine which points on the left should be chosen as the left wings for each right wing point of a butterfly, and also the routine to determine what W factors and when should be used. For 512 x 512 mesh, the runtime ranges from 414 seconds to 32 seconds (13 times speedup with 16 processors.

-The present one problem is that the results on the SUN are very different fron than that on the Encore. Need debugging.

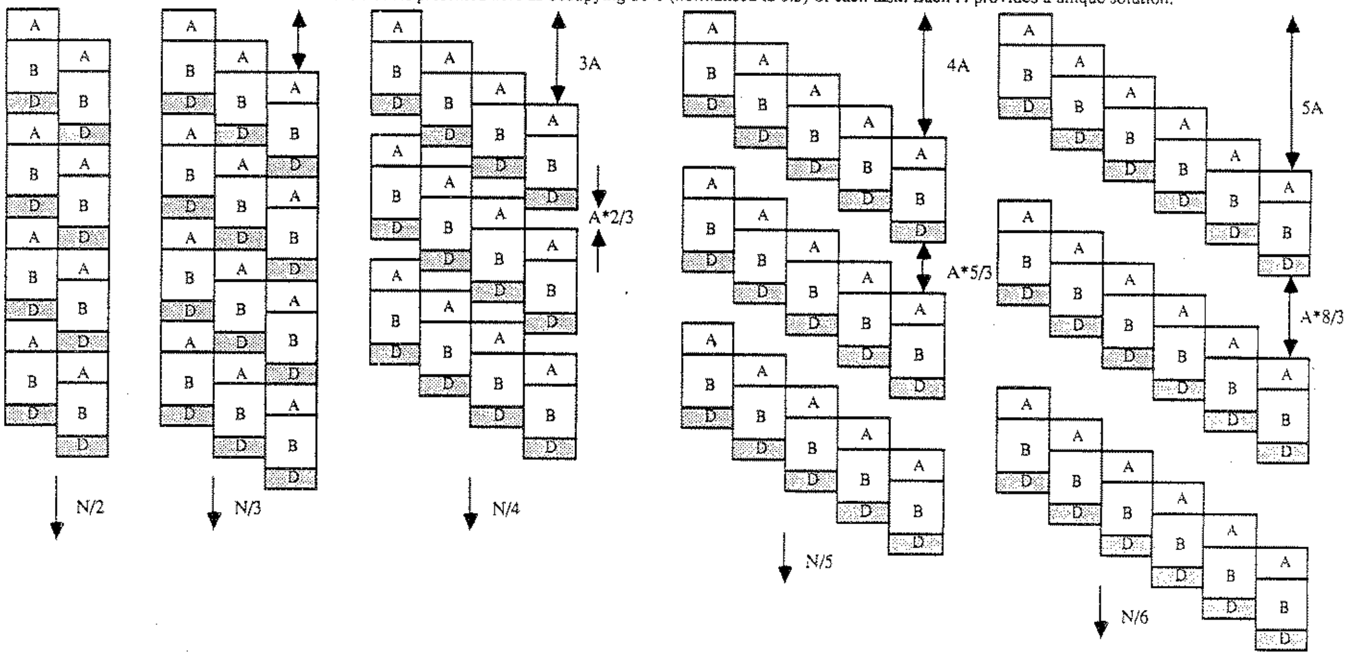
-New bugs of OSC discovered

-Analysis of memory allocation and deallocation

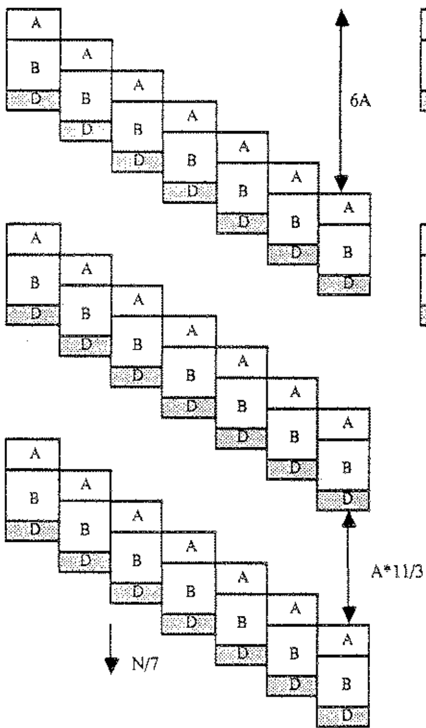
The Analysis of the Effect of Memory Allocation and Deallocation on the Parallelism of a Parallel Code

A - Memory Allocation
B - Slice Body
D - Memory Deallocation

It is expected that for some data structures, the allocation and deallocation operations at runtime overlap each other. The best case is when they run concurrently. The worst case is when they are mutually exclusive and have to run serial to one another due to access to the same data structure. The former is simpler than thelatter to analyse and the analysis is shown below. The execution time of each iteration is normalised to 1 and the total number of tasks is N. A is presented here as occupying 30% (normalised to 0.3) of each task. Each A provides a unique solution.

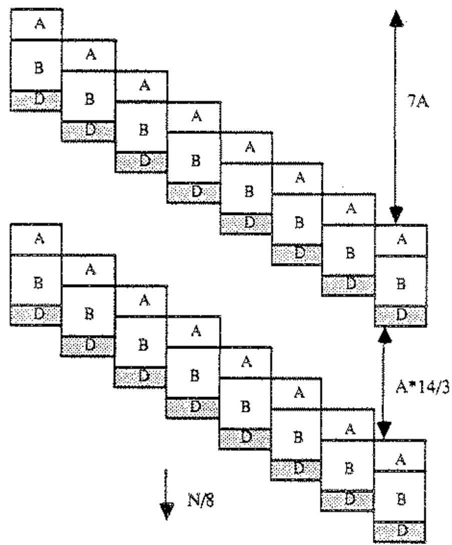


N/2 + A	N/3 + 2A	N/4 + 3A + (N/4-1)A*2/3	N/5 + 4A + (N/5-1)A*5/3	N/6 + 5A + (N/6-1)A*8/3
N=512				
SU=1.998	SU=2.989	SU=3.318	SU=3.318	SU=3.318
t=256.3	t=171.267	t=154.3		



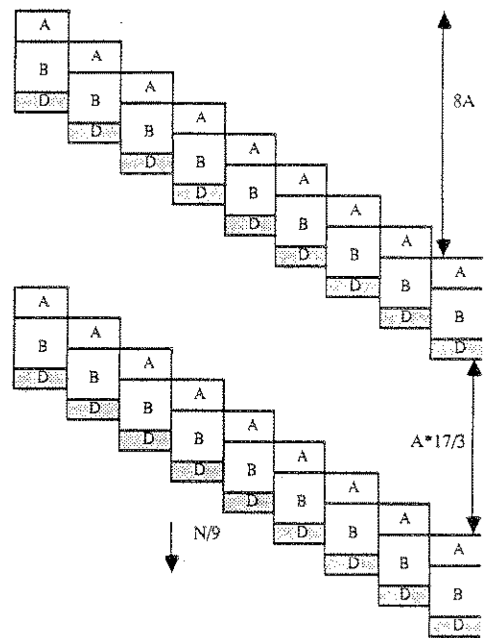
$N/7 + 6A + (N/7-1)A*11/3$

SU=3.318



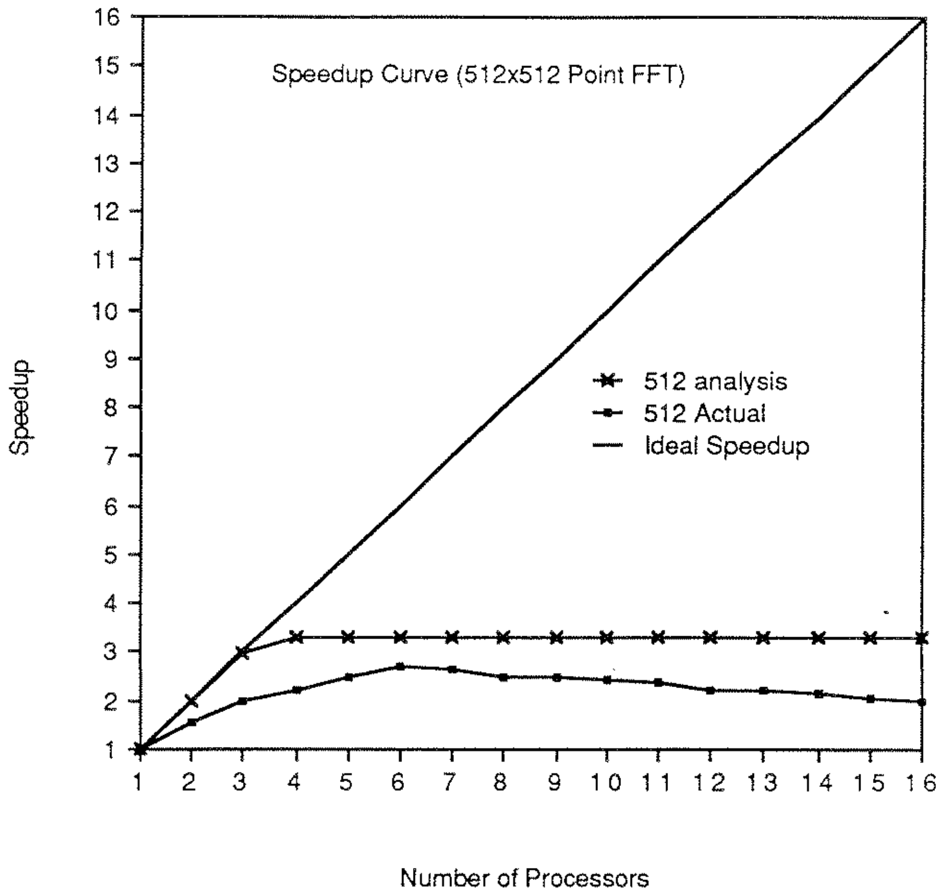
$N/8 + 7A + (N/8-1)A*14/3$

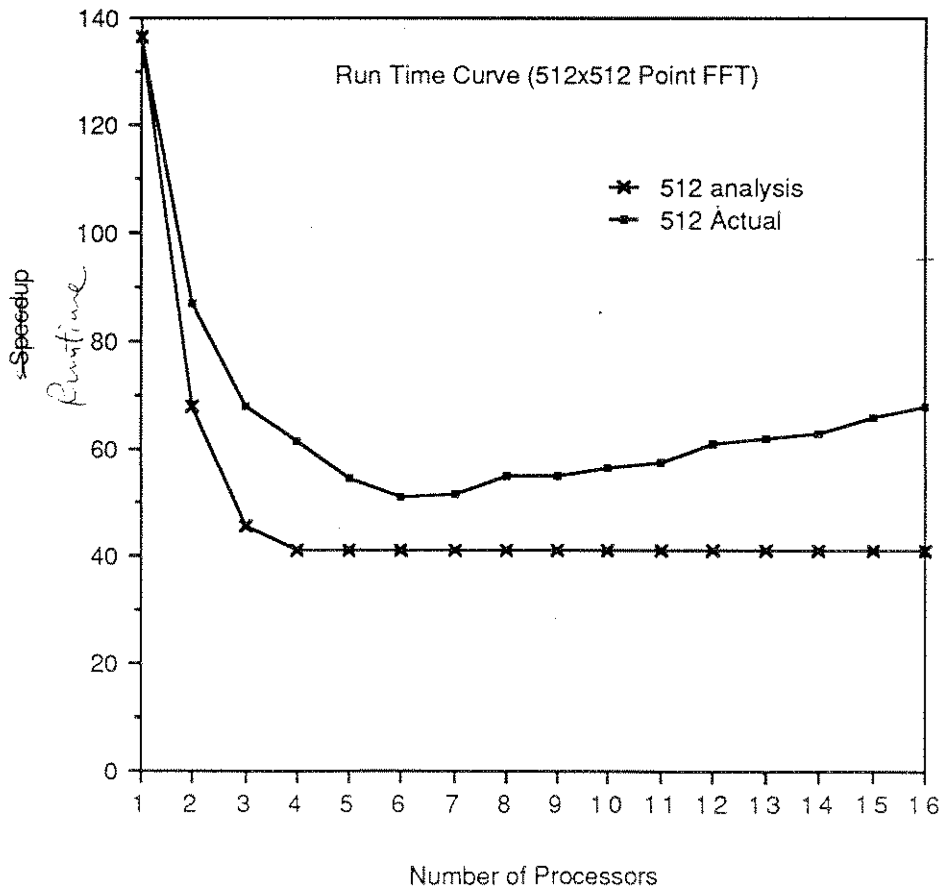
SU=3.318

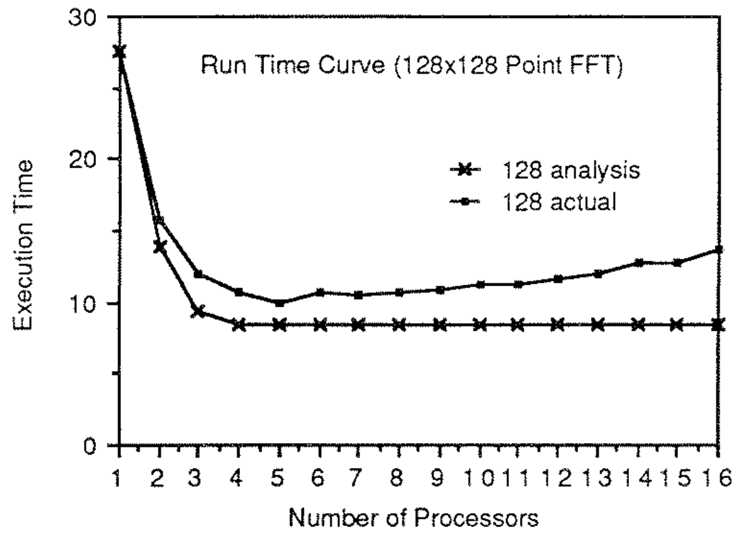


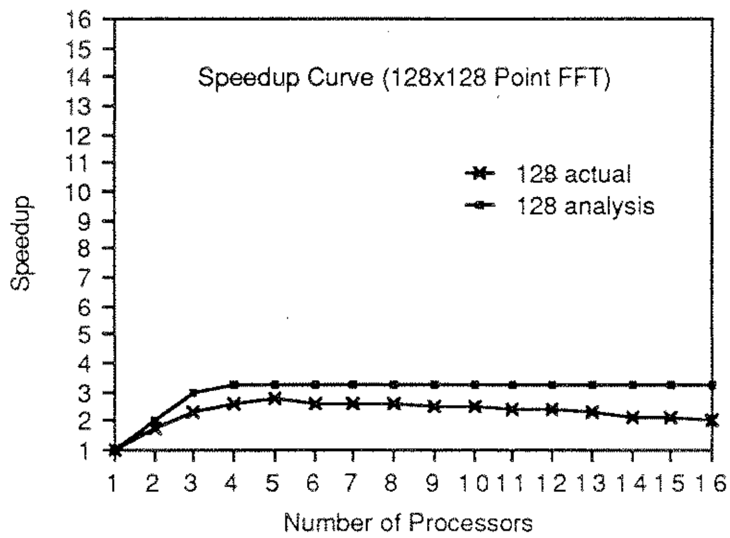
$N/9 + 8A + (N/9-1)A*17/3$

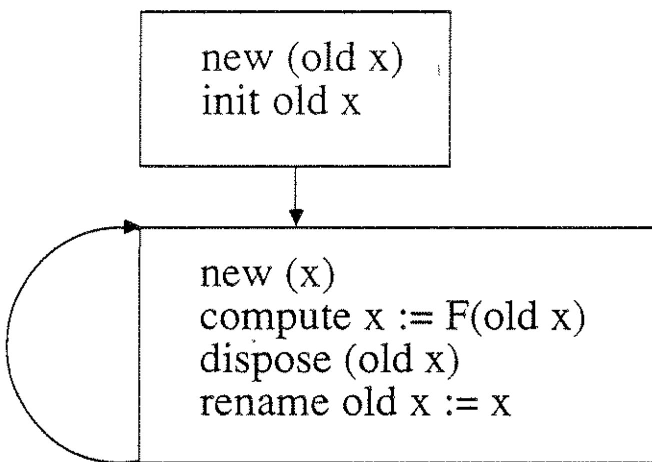
SU=3.318

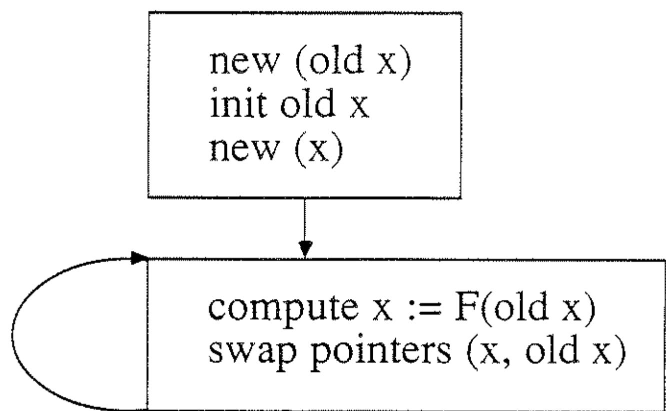

Speedup Curve (512x512 Point FFT)
512 analysis
512 Actual
Ideal Speedup
Speedup
Number of Processors


Run Time Curve (512x512 Point FFT)
512 analysis
512 Actual
Speedup
Runtime
Number of Processors


Run Time Curve (128x128 Point FFT)
128 analysis
128 actual
Execution Time
Number of Processors
30
20
10
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16


Speedup Curve (128x128 Point FFT)
128 actual
128 analysis
Speedup
Number of Processors
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16



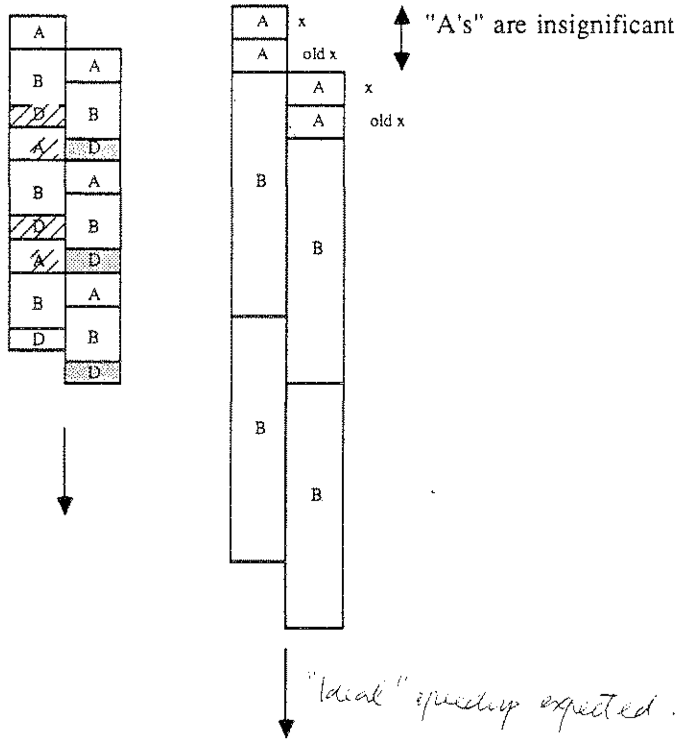


Present Scheme

A Better Scheme

Memory Allocation and Deallocation Scheme of OSC

For the case in which array sizes are invariant through loop iterations.



Effect of implementing do-invariant allocation/deallocation in loop body for cases in which array sizes are invariant through loop iterations.

Possible scheme:
- code motion and data structure pointer reassignment to remove the allocation and deallocation of fixed size data structures from within iterative loops [FFT report]
(appropriate optimisation by hand at C level is relatively easy för simple examples)

- where the data structure size cannot be determined statically, data deallocation should be overlapped with main computation of loop body *lazily* .

Mathematical analysis for upper bound performance of a supposedly parallel SISAL code:

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

- Speedup curve for code saturates dramatically at unexpectedly low value
- Need for improvement in memory allocation and deallocation scheme implemented in OSC on ENCORE

Debugging SISAL Programs

To date impossible to debug SISAL program at source level
Best is DI but as had bugs
Program debugging at C code level is sometimes useful but C code generated from SISAL is not always correct

Indirect debugging in DI is difficult and unreliable, requires additional tedious and error prompt efforts - Hiesenberg effect
Most serious drawback which discourages anyone from doing serious programming in SISAL. Research into source level debugging aids for SISAL is therefore needed

---

Language Support for Complex Numbers

Explicit tasks in the treatment of complex numbers as records may result in an additional execution cost

Remedy: SISAL support for complex numbers similar to FORTRAN's, making treatment of complex numbers implicit
Remove necessity of building records, extracting elements from records and calling functions for complex arithmetic which obscures underlying algorithm

---

OLD Statements: an Easy Mistake

OLD is used on the right hand side
Multiple accesses of an **OLD** variable are common place
Coexistence of the variables evaluated in present iteration and **OLD** variables evaluated in previous iteration an easy mistake
Once **OLD** statement missed out, error is very difficult to detect

Appendix: Code generating "Normalisation Error"

```
% Author: Pau S. Chang
% Revised: 2/2/1990
% Module: The initialisation stage of the Spectral Weather Model.
%
% Problems (highlighted in bold in the main program):
% (i)  Graph Normalisation incomplete inside IF1OPT??
% (ii) Loops of similar loop bound are forced to be "coupled" together
%      in order to pass through the OSC compiler. They are many of such
%      cases here. An example is in the calculation of variance "var"
%      and Average potential height "h":
```

```
---------------------------------------------
filename: Makefile

# makefile for the SISAL codes barotropic model Version 1.8

# Let
ifl files =       main.ifl IntrFuncs.ifl complex.ifl \
                  Inital.ifl InitFFT.ifl gaussg.ifl \
                  legendre.ifl SasAlfa.ifl

.SUFFIXES:
.SUFFIXES:        .sis .ifl

# Compile from .sis files to .ifl files
.sis.ifl:
                  osc $*.sis -IF1 -double_real

prefft:           $(ifl files)
                  osc -v -o prefft $(ifl files)
```

```
--------------------------------
Filename: IntrFuncs.sis

DEFINE ASINR, ACOSR, SQRTR, DSIN, DCOS

% ---------Intrinsic Functions
global SIN(num : real RETURNS real)
global COS(num : real RETURNS real)
global ASIN(num : double_real RETURNS double_real)
global ACOS(num : double_real RETURNS double_real)
global SQRT(num : double_real RETURNS double_real)

% Catering for real operations of Intrinsic functions
function ASINR(num : real RETURNS real)
real(ASIN(double_real(num)))
end function

function ACOSR(num : real RETURNS real)
real(ACOS(double_real(num)))
end function

function SQRTR(num : real RETURNS real)
real(SQRT(double_real(num)))
end function

% Catering for double_real operations of Intrinsic functions
function DSIN(num : double_real RETURNS double_real)
double_real(SIN(real(num)))
end function
```

```
function DCOS(num : double_real RETURNS double_real)
double_real(COS(real(num)))
end function

-------------------------------
Filename: complex.sis

DEFINE Cadd, Csub, Cmul, Cdiv, Crmul, Crsub, Crdiv, Conjg, Cneg,
    Csqrt, Cabs, CabsSqr

type CplexReal = Record[Repart,Impart:real];
type ArrCplexReal = Array[CplexReal];

% ---------Intrinsic Functions
global SIN(num : real RETURNS real)
global COS(num : real RETURNS real)
global ATAN(num : real RETURNS real)
global SQRTR(num : real RETURNS real)

% These subroutines do the arithmatics of complex numbers:

% cnum1 + cnum2
function Cadd(cnum1, cnum2 : CplexReal RETURNS CplexReal)
record CplexReal[ Repart : cnum1.Repart + cnum2.Repart;
                  Impart : cnum1.Impart + cnum2.Impart ]
end function

% cnum1 - cnum2
function Csub(cnum1, cnum2 : CplexReal RETURNS CplexReal)
record CplexReal[ Repart : cnum1.Repart - cnum2.Repart;
                  Impart : cnum1.Impart - cnum2.Impart ]
end function

% cnum1 * cnum2
function Cmul(cnum1, cnum2 : CplexReal RETURNS CplexReal)
record CplexReal[ Repart : cnum1.Repart * cnum2.Repart -
                          cnum1.Impart * cnum2.Impart;
              Impart : cnum1.Repart * cnum2.Impart +
                          cnum1.Impart * cnum2.Repart  ]
end function

% cnum1/cnum2
function Cdiv(cnum1, cnum2 : CplexReal RETURNS CplexReal)
LET dnom:= cnum2.Repart * cnum2.Repart + cnum2.Impart * cnum2.Impart;
IN
record CplexReal[ Repart : (cnum1.Repart*cnum2.Repart +
                 cnum1.Impart*cnum2.Impart) / dnom;
                 Impart : (cnum1.Impart*cnum2.Repart -
                      cnum1.Repart*cnum2.Impart) / dnom ]
end LET
end function

% Real_constant*cnum
function Crmul(cons : real; cnum : CplexReal RETURNS CplexReal)
record CplexReal[ Repart : cons * cnum.Repart; Impart : cons * cnum.Impart ]
end function

% cnum-Real_constant
function Crsub(cnum : CplexReal; cons : real RETURNS CplexReal)
record CplexReal[ Repart : cnum.Repart-cons; Impart : cnum.Impart ]
end function

% cnum/Real_constant
function Crdiv(cnum : CplexReal; cons : real RETURNS CplexReal)
record CplexReal[ Repart : cnum.Repart / cons; Impart : cnum.Impart / cons ]
```

```
end function

% conjugate(cnum)=Repart-Impart
function Conjg(cnum : CplexReal RETURNS CplexReal)
record CplexReal[ Repart : cnum.Repart; Impart : -cnum.Impart ]
end function

% Cneg(cnum)
function Cneg(cnum : CplexReal RETURNS CplexReal)
record CplexReal[ Repart : -cnum.Repart; Impart : -cnum.Impart ]
end function

% Csqrt(cnum)
function Csqrt(cnum:CplexReal RETURNS CplexReal)
LET
RR := cnum.Repart;
II := cnum.Impart;
mag := SQRTR( SQRTR(RR * RR + II * II) );
angle := ATAN(II / RR) / 2.0;
Re, Im := mag * COS(angle), mag * SIN(angle);
IN
record CplexReal[Repart : Re; Impart : Im]
end LET
end function

% Cabs(cnum) refers to the MAGNITUDE of the complex number.
function Cabs(cnum : CplexReal RETURNS real)
SQRTR(cnum.Repart * cnum.Repart + cnum.Impart * cnum.Impart)
end function

% CabsSqr(cnum) refers to the MAGNITUDE Square of the complex number.
function CabsSqr(cnum : CplexReal RETURNS real)
cnum.Repart * cnum.Repart + cnum.Impart * cnum.Impart
end function

---------------------------------------
Filename: Inital.sis

DEFINE Inital

type ArrInt1 = Array[integer];
type ArrReal1 = Array[real];

global SQRTR(num : real RETURNS real)

FUNCTION Inital(ir, ilong, ilat, mx, jx, jxx : integer; zmean1 : real
                RETURNS integer, integer, integer, integer,
                        real, real, real, real, real,
                        arrint1, arrint1, arrint1,
                        arrreal1)
LET
ww := 7.292E-5;
tw := ww * 2.0;
irmax:= ir;

ilath, irmax1, irmax2 := ilat / 2, irmax + 1, irmax + 2;

asq, grav := 6371.22E3 * 6371.22E3, 9.80616;
zmean:= zmean1 * grav / asq;

kmjx, kmjxx := FOR m IN 1, mx
                RETURNS ARRAY of (m - 1) * jx
                        ARRAY of (m - 1) * jxx
                END FOR;
```

```
ksq :=    FOR j IN 1, ir * 2
          RETURNS ARRAY of j * (j + 1)
          END FOR;

epsilon_a :=        FOR mp IN 1, mx
                    RETURNS VALUE of CATENATE
                       FOR j IN 1, jxx    % epsilon_size is 1-272
                       l := j + mp - 2;
                       m := mp - 1;
                       t := real((l + m) * (l - m));
                       b := real(4 * l * l - 1);
                       RETURNS ARRAY of SQRTR(t / b)
                       END FOR
                    END FOR;

epsilon := epsilon_a[1 : 0.0];

IN
ir, ilong, ilath, irmax2, ww, zmean, tw, asq, grav,
kmjx, kmjxx, ksq, epsilon

END LET
END FUNCTION
-------------------------------
Filename: InitFFT.sis

DEFINE InitFFT

type ArrInt1 = Array[integer];
type ArrReal1 = Array[real]

global SIN(num : real RETURNS real)
global COS(num : real RETURNS real)

%----------------factr4/facStep/facRecur
function facRecur(nparti, idiv, ifTi : integer;
                  ifacti : ArrInt1
                  RETURNS integer, integer, integer, ArrInt1)
FOR INITIAL
npart := nparti;
iquot := npart / idiv;
ifT := ifTi;
ifact1 := ifacti;

WHILE npart - idiv * iquot = 0 REPEAT
npart := old iquot;
iquot := npart / idiv;
ifT := old ifT + 1;
ifact1 := old ifact1[ifT : idiv]

RETURNS VALUE of npart
        VALUE of iquot
        VALUE of ifT
        VALUE of ifact1
END FOR
END function    % facRecur

%-----------------factr4/facStep
function Loop_id(n : integer
                 RETURNS integer, integer, ArrInt1)
FOR INITIAL     % loop_id
id := 1;
ifT := 0;
npart := n;
ifact := ARRAY_fill(1, 20, 0)   % NOTE: wild guess
```

```
WHILE id <= n REPEAT
idiv := IF old id - 1 <= 0  THEN 2
                               ELSE old id
    END IF;

npart, iquot, ifT, ifact :=  facRecur(old npart, idiv, old ifT, old ifact);

id := IF  iquot - idiv <= 0 THEN n + 1  % just to make it greater than n
                               ELSE old id + 2
   END IF;

RETURNS VALUE of npart
          VALUE of ifT
          VALUE of ifact
END FOR
END function       % Loop_id

%------------------------
function FACTR4(n : integer RETURNS integer, arrint1)
LET
npart, ifT, ifact1 :=  Loop_id(n);

iff := if npart - 1 > 0  then ifT + 1
                         else ifT
   END if;
ifact2 := if npart - 1 > 0  then ifact1[iff : npart]
                         else ifact1
    END if;

nfactT := iff;
n2 :=     FOR INITIAL
          n2 := 0;
          i := 1;

          % n2 includes case i=nfactT
          WHILE i <= nfactT REPEAT
          i := old i + 1;
          n2 := if ifact2[old i] = 2 then old n2 + 1
                                   else old n2
             END if
          RETURNS VALUE of n2
          END FOR;          % NOTE: very ineffecient!

n4 := n2 / 2;
ifact3 := ARRAY_fill(1, n4, 4)
          ||
          for i in n4 + 1, nfactT - n4 RETURNS
                   ARRAY of ifact2[n4 + i]
          END for
          ||
          ARRAY_fill(nfactT - n4 + 1, nfactT, 0);

nfact := nfactT - n4;

IN nfact, ifact3

END LET
END function       % factr4
%---------------------------------------------------------------

% Subroutine InitFFT does the initialisations necessary so that the
% FFT's can be used. It factorises the number of longitudinal points.
% TRIGF are for forward transforms while TRIGB are for reverse.
```

```
function InitFFT(n : integer
                 RETURNS boolean, boolean, integer, arrint1, ArrReal1, ArrReal1)
LET
Abortinitfft :=    IF (MOD(n, 2) ~= 0 | n > 200)        THEN true ELSE false
                   END IF;
AbortFFT := IF n > 96 THEN true ELSE false END IF;

pi := 3.14159265;

nfax, ifax := FACTR4(n);

trigf, trigb :=
  IF Abortinitfft
  THEN  array ArrReal1 [],
        array ArrReal1 []

  ELSE  FOR Lp IN 1, n
          k := (Lp + 1) / 2;
          Cargument := - 2.0 * pi * real(k - 1)/real(n);
          COStheta := COS(Cargument);      % Repart
          SINtheta := SIN(Cargument);      % Impart

          RETURNS ARRAY of IF MOD(Lp, 2) = 0 THEN SINtheta
                                              ELSE COStheta
                           END IF
                  ARRAY of  IF MOD(Lp, 2) = 0 THEN - SINtheta
                                              ELSE COStheta
                           END IF
        END FOR
  END IF

IN AbortFFT, Abortinitfft, nfax, ifax, trigf, trigb

END LET
END function

------------------------------------
Filename: gaussg.sis

DEFINE gaussg

type ArrReal1 = Array[real];
type ArrDreal1 = Array[double_real]

global ACOS(num : double_real RETURNS double_real)
global SQRT(num : double_real RETURNS double_real)

global SIN(num : double_real RETURNS double_real)
global COS(num : double_real RETURNS double_real)

FUNCTION ORDLEG(ir : integer; coa : double_real
                RETURNS double_real)
LET
irpp, irppm := ir+1, ir;
delta := ACOS(coa);
sqr2 := SQRT(2.0d0);
theta := delta;
c1 := sqr2 *    FOR n IN 1, irppm
                fn := n;
                fn2 := fn * 2;
                fn2sq :=  double_real(fn2 * fn2);
                RETURNS VALUE of product SQRT(1.0d0 - 1.0d0 / fn2sq)
                END FOR;

s1 :=    FOR INITIAL
```

```
                n := irppm;
                fn := double_real(irppm);
                fn2 := fn * 2.0d0;
                ang := fn * theta;
                s1T := 0.0d0;
                c4 := 1.0d0;
                a := -1.0d0;
                b := 0.0d0;
                n1 := n + 1;
                kk := 1;

                WHILE kk <= n1 REPEAT
                kk := old kk + 2;
                k := old kk - 1;
                c4T := IF k=n THEN 0.5d0 * old c4
                              ELSE old c4
                          END IF;
                s1T := old s1T + c4T * COS(old ang);
                a := old a + 2.0d0;
                b := old b + 1.0d0;
                fk := double_real(k);
                ang := theta * (fn - fk - 2.0d0);
                c4 := a * (fn - b + 1.0d0) / (b * (fn2 - a)) * c4T;

                RETURNS VALUE OF s1T
                END FOR;
sx := s1 * c1;
IN sx
END LET
END FUNCTION
%----------------------------------------------------------------

%-----------------gaussg/cycle
FUNCTION CYCLE(ir, irm, irp : integer;
                    ft, a, b, xlim : double_real
                    RETURNS double_real)
LET
g := ORDLEG(ir, ft);
gm := ORDLEG(irm, ft);
gp := ORDLEG(irp, ft);
gt := (ft * ft - 1.0d0)/(a * gp - b * gm);
ftemp := ft - g * gt;
gtemp := ft - ftemp;
ftnew := ftemp;

IN IF ABS(gtemp) - xlim > 0.0d0
          THEN CYCLE(ir, irm, irp, ftnew, a, b, xlim)
          ELSE ftnew
  END IF
END LET
END FUNCTION

%--------------------gaussg
FUNCTION gaussg(nzero : integer
          RETURNS ArrReal1, ArrReal1, ArrReal1, ArrReal1, ArrReal1, ArrDreal1)
LET
xlim := 1.0d-12;
ir := nzero * 2;
fi := double_real(ir);
fi1 := fi + 1.0d0;
pi := 3.141592653589793d0;
piov2 := pi * 0.5d0;
fn := piov2/double_real(nzero);
wt := FOR lat IN 1,nzero RETURNS
          ARRAY of double_real(lat) - 0.5d0
```

```
    END FOR;
f := FOR lat IN 1,nzero RETURNS
          ARRAY of SIN( wt[lat] * fn + piov2 )
   END FOR;

dn := fi/SQRT(4.0d0 * fi * fi - 1.0d0);
dn1 := fi1/SQRT(4.0d0 * fi1 * fi1 - 1.0d0);
a := dn1 * fi;
b := dn * fi1;
irp := ir + 1;
irm := ir - 1;
fnew  := FOR lat IN 1, nzero RETURNS
          ARRAY of CYCLE(ir, irm, irp, f[lat], a, b, xlim)
    END FOR;
wtnew, radnew, coangnew, sianew :=
          FOR lat IN 1, nzero
                    a1 := 2.0d0 * (1.0d0 - fnew[lat] * fnew[lat]);
                    bo := ORDLEG(irm, fnew[lat]);
                    b1 := bo * bo * fi * fi;
                    wtt := a1 * (fi - 0.5d0) / b1;
                    radt := ACOS(fnew[lat]);
                    coangt := radt * 180.0d0 / pi;
                    siat := SIN(radt);
                    RETURNS ARRAY of wtt
                              ARRAY of radt
                              ARRAY of coangt
                              ARRAY of siat
    END FOR;

WORKiyh := fnew || wtnew || sianew || radnew || coangnew;

fs, wts, sias, rads, coangs :=
   FOR lat IN 1, nzero
          RETURNS          ARRAY of real(fnew[lat])
                    ARRAY of real(wtnew[lat])
                    ARRAY of real(sianew[lat])
                    ARRAY of real(radnew[lat])
                    ARRAY of real(coangnew[lat])
   END FOR;

IN fs, wts, sias, rads, coangs, WORKiyh
END LET
END FUNCTION
```

Filename: legendre.sis

```
DEFINE legendre

type ArrDreal1 = Array[Double_real]

global SIN(num : double_real RETURNS double_real)
global COS(num : double_real RETURNS double_real)
global SQRT(num : double_real RETURNS double_real)

FUNCTION legendre(ir, irmax2, jxxmx : integer;
                    coas, sias, deltas : real;
                    RETURNS ArrDreal1)
LET
p := LET
   coa := double_real(coas);
   sia := double_real(sias);
   delta := double_real(deltas);
   irpp := ir + 2;
   theta := delta;
```

```
sqr2 := SQRT(2.0d0);
pp := FOR INITIAL
        n := 1;
        c1 := sqr2;
        pLoop1 := ARRAY ArrDreal1[1: 1.0d0 / sqr2]
                  ||
                  FOR jm IN 2, jxxmx
                  RETURNS ARRAY of 0.0d0
                  END FOR;

   WHILE n <= irpp REPEAT
        n := old n + 1;
        fn := double_real(old n);
        fn2 := 2.0d0 * fn;
        fn2sq := fn2 * fn2;
        c1 := old c1 * SQRT(1.0d0 - 1.0d0 / fn2sq);
        c3 := c1 / SQRT(fn * (fn + 1.0d0));

        s1, s2 :=
                FOR INITIAL
                kk := 1;
                ang := fn * theta;
                n1 := old n + 1;
                ss1, ss2 := 0.0d0, 0.0d0;
                c4, c5 := 1.0d0, fn;
                a, b := - 1.0d0, 0.0d0;

                WHILE kk <= n1 REPEAT
                kk := old kk + 2;
                k := old kk - 1;
                ss2 := old ss2 + old c5 * SIN(old ang) * old c4;
                c4t := if k = old n then 0.5d0 * old c4
                               else old c4
                   END if;
                ss1 := old ss1 + c4t * COS(old ang);
                a := old a + 2.0d0;
                b := old b + 1.0d0;
                fk := double_real(k);
                ang := theta * (fn - fk - 2.0d0);
                c4 := (a * (fn - b + 1.0d0) / (b * (fn2 - a))) * c4t;
                c5 := old c5 - 2.0d0

                RETURNS         VALUE of ss1
                        VALUE of ss2      % to s1 and s2
                END FOR;

        pLoop1 := IF old n - irpp < 0
                  THEN old pLoop1[old n + 1 : s1 * c1; old n + irmax2 : s2 * c3]
                  ELSEIF old n - irpp = 0
                  THEN old pLoop1[old n + irmax2 : s2 * c3]
                  ELSE old pLoop1
             END IF

        RETURNS VALUE of pLoop1          % to pp
        END FOR;

p2 :=
        IF ir = 2
        THEN pp

        ELSE    FOR INITIAL
                m := 2;
                ppp := pp

                WHILE m <= ir REPEAT
```

```
m := old m + 1;
fm := double_real(old m);
fm1, fm2, fm3 := fm - 1.0d0, fm - 2.0d0, fm - 3.0d0;
mm1 := old m - 1;
m1 := old m + 1;
c6 := SQRT((2.0d0 * fm + 1.0d0) / (2.0d0 * fm));
p5 := old ppp[irmax2 * old m + 1 : c6 * sia *
                                   old ppp[irmax2 * mm1 + 1]];
mpir := old m + ir + 1;
mt := old m;

ppp := FOR INITIAL
   l := m1;
   p4 := p5;

   WHILE l <= mpir REPEAT
   l := old l + 1;
   fn := double_real(old l);
   c7 := (fn * 2.0d0 + 1.0d0) / (fn * 2.0d0 - 1.0d0);
   c8 := (fm1 + fn) / ((fm + fn) * (fm2 + fn));
   c := SQRT((fn * 2.0d0 + 1.0d0) /
            (fn * 2.0d0 - 3.0d0) * c8 * (fm3 + fn));
   d := SQRT(c7 * c8 * (fn - fm1));
   e := SQRT(c7 * (fn - fm) / (fn + fm));
   lm := irmax2 * mt + old l - mt + 1;
   lmm2 := irmax2 * (mt - 2) + old l - mt + 3;
   lm1mm2 := lmm2 - 1;
   lm2mm2  := lm1mm2 - 1;
   lm1m := lm - 1;

   p4 :=  IF old l - mpir < 0
            THEN old p4[lm:c * old p4[lm2mm2]
                   - d * old p4[lm1mm2] * coa
                   + e * old p4[lm1m] * coa]

          ELSEIF old l - mpir > 0
          THEN old p4

          ELSE LET
                 a := SQRT((fn * fn - 0.25d0) /
                          (fn * fn - fm * fm));
                 b := SQRT((2.0d0 * fn + 1.0d0) *
                          (fn - fm - 1.0d0) * (fn + fm1)
                     / ((2.0d0 * fn - 3.0d0) * (fn - fm)
                          * (fn + fm)));
                 lm2m := lm1m - 1;

                 IN
                 old p4[lm : 2.0d0 * a * coa * old p4[lm1m]
                               - b * old p4[lm2m]]
                 END LET

      END IF

   RETURNS VALUE of p4            % to p6
   END FOR;

RETURNS VALUE of ppp                  % to p2
END FOR

END IF

IN p2
END LET;                  % RETURNS p2 to p
```

```
IN p
END LET

END FUNCTION

---------------------------------------
Filename: SasAlfa.sis

DEFINE SasAlfa
TYPE arrDreal1 = ARRAY [double_real];
TYPE arrDreal2 = ARRAY [arrDreal1];
TYPE arrDreal3 = ARRAY [arrDreal2];
TYPE arrreal1 = ARRAY [real];
TYPE arrreal2 = ARRAY [arrreal1];
TYPE arrreal3 = ARRAY [arrreal2]

FUNCTION SasAlfa(ir, irmax2, jxxmx, ilath : integer; alp : ArrDReal2
                    RETURNS ArrReal3)
LET
lpfin :=            IF MOD(ir, 2) = 0 THEN ir + 1
                    ELSE ir + 2
                    END IF;

alfa :=   FOR hemi IN 1, 2 CROSS latlev IN 1,ilath
          RETURNS ARRAY of
                    IF hemi = 1         % North
                    THEN    FOR specindex IN 1, jxxmx
                            RETURNS ARRAY of real(alp[latlev, specindex])
                            END FOR
                    ELSE    FOR mp IN 1, ir + 1          % South
                            RETURNS VALUE of CATENATE
                                    FOR lp IN 1, lpfin
                                    ilm := (mp - 1) * irmax2 + lp;
                                    RETURNS ARRAY of
                                            IF lp = 1 | MOD(lp, 2) ~= 0
                                            THEN real(alp[latlev, ilm])
                                            ELSE real(-alp[latlev, ilm])
                                            END IF
                                    END FOR
                            END FOR
                    END IF
          END FOR
IN alfa
END LET
END FUNCTION

% Main Program

DEFINE MAIN

type ArrInt1 = Array[integer];
type ArrReal1 = Array[real];
type ArrReal2 = Array[ArrReal1];
type ArrReal3 = Array[ArrReal2];
type ArrDreal1 = Array[Double_real];
type ArrDreal2 = Array[ArrDreal1];
type CplexReal = Record[Repart,Impart:real];
type ArrCplexReal = Array[CplexReal];

global SIN(num : real RETURNS real)
global ACOSR(num : real RETURNS real)

global Cadd(cnum1, cnum2 : CplexReal RETURNS CplexReal)
global Crmul(cons : real; cnum : CplexReal RETURNS CplexReal)
global CabsSqr(cnum : CplexReal RETURNS real)
```

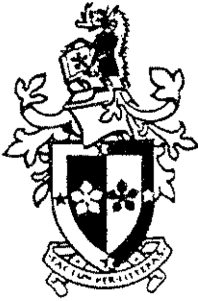
```
global Inital(      ires, ix, iy, mx, jx, jxx : integer; zmean1 : real
                    RETURNS integer, integer, integer, integer,
                            real, real, real, real, real,
                            arrint1, arrint1, arrint1,
                            arrreal1)

global InitFFT(     n : integer
                    RETURNS boolean, boolean, integer, arrint1,
                            ArrReal1, ArrReal1)

global gaussg(nzero : integer
          RETURNS ArrReal1, ArrReal1, ArrReal1, ArrReal1, ArrReal1, ArrDreal1)

global legendre(ir, irmax2, jxxmx : integer;
                    coas, sias, deltas : real
                    RETURNS ArrDreal1)

global SasAlfa(   ir, irmax2, jxxmx, ilath : integer;
                      alp_double: ArrDReal2
                      RETURNS ArrReal3)

function MAIN(
          ires, ix, iy,
          ktotal, idelt, idumpt_i, nrun, imp, istart, izon, ilin:integer;
          zmean_1, hdiff, hdrag, vnu:real;
          p_in, c_in, z_in, zt_mountain:ArrCplexReal
          RETURNS integer,
          integer, integer, integer, integer, integer, integer,
           integer, integer, integer, integer, integer, integer, integer,
           integer, integer, integer, integer, integer, integer, integer,
           integer, integer, integer, real, real, real, real, real, real,
           real, real, ArrInt1, ArrInt1, ArrInt1, ArrInt1, ArrReal1, ArrReal1,
           ArrReal3, ArrCplexReal, ArrCplexReal, ArrCplexReal, ArrCplexReal,
           ArrReal1, ArrCplexReal, ArrCplexReal, ArrCplexReal,
           ArrReal1, ArrReal1)

LET
ixh := ix/2;
iyh := iy/2;
jxx := ires + 2;
jx := ires + 1;
mx := ires + 1;
jxxmx := jxx * mx;
jxmx := jx * mx;
mxmx := mx * mx;
mx2 := mx * 2;
jxmx2 := jxmx * 2;
jxxmx2 := jxxmx * 2;

ifirst := 1;
itflag := 1;
iglobe := 2;
delt := idelt;
idumpt := IF idumpt_i = 0 THEN 1000
                          ELSE idumpt_i
           END IF;

zero := record CplexReal[Repart : 0.0; Impart : 0.0];

ir, ilong, ilath, irmax2, ww, zmean, tw, asq, grav,
kmjx, kmjxx, ksq_1_uncared_for, epsi :=
         Inital(ires, ix, iy, mx, jx, jxx, zmean_1);

ksq := ARRAY[0 : 0] || ksq_1_uncared_for || ARRAY[1 : 0, 0];
```

```
global Inital(       ires, ix, iy, mx, jx, jxx : integer; zmean1 : real
                     RETURNS integer, integer, integer, integer,
                             real, real, real, real, real,
                             arrint1, arrint1, arrint1,
                             arrreal1)

global InitFFT(   n : integer
                     RETURNS boolean, boolean, integer, arrint1,
                             ArrReal1, ArrReal1)

global gaussg(nzero : integer
         RETURNS ArrReal1, ArrReal1, ArrReal1, ArrReal1, ArrReal1, ArrDreal1)

global legendre(ir, irmax2, jxxmx : integer;
                     coas, sias, deltas : real
                     RETURNS ArrDreal1)

global SasAlfa(   ir, irmax2, jxxmx, ilath : integer;
                       alp_double: ArrDReal2
                       RETURNS ArrReal3)

function MAIN(
         ires, ix, iy,
         ktotal, idelt, idumpt_i, nrun, imp, istart, izon, ilin:integer;
         zmean_1, hdiff, hdrag, vnu:real;
         p_in, c_in, z_in, zt_mountain:ArrCplexReal
         RETURNS  integer,
         integer, integer, integer, integer, integer, integer,
          integer, integer, integer, integer, integer, integer, integer,
          integer, integer, integer, integer, integer, integer, integer,
          integer, integer, integer, real, real, real, real, real, real,
          real, real, ArrInt1, ArrInt1, ArrInt1, ArrInt1, ArrReal1, ArrReal1,
          ArrReal3, ArrCplexReal, ArrCplexReal, ArrCplexReal, ArrCplexReal,
          ArrReal1, ArrCplexReal, ArrCplexReal, ArrCplexReal,
          ArrReal1, ArrReal1)

LET
ixh := ix/2;
iyh := iy/2;
jxx := ires + 2;
jx := ires + 1;
mx := ires + 1;
jxxmx := jxx * mx;
jxmx := jx * mx;
mxmx := mx * mx;
mx2 := mx * 2;
jxmx2 := jxmx * 2;
jxxmx2 := jxxmx * 2;

ifirst := 1;
itflag := 1;
iglobe := 2;
delt := idelt;
idumpt := IF idumpt_i = 0 THEN 1000
                             ELSE idumpt_i
          END IF;

zero := record CplexReal[Repart : 0.0; Impart : 0.0];

ir, ilong, ilath, irmax2, ww, zmean, tw, asq, grav,
kmjx, kmjxx, ksq_1_uncared_for, epsi :=
          Inital(ires, ix, iy, mx, jx, jxx, zmean_1);

ksq := ARRAY[0 : 0] || ksq_1_uncared_for || ARRAY[1 : 0, 0];
```

```
AbortFFT, AbortInitFFT, nfax, ifax, trigf, trigb := InitFFT(ix);

coa, w, sia, delta, wocs, WORKiyh := gaussg(ilath);     % size iyh

wix :=    IF ilin = 0                    % Indeed
          THEN    FOR lat_level IN 1, ilath
                  RETURNS ARRAY of w[lat_level] / real(ix)
                  END FOR
          ELSE    w
          END IF;           % size iyh; of the North

winv, coainv := FOR lat_level IN 1, ilath
                  winv := wix[iy / 2 + 1 - lat_level];
                  coainv := -coa[iy / 2 + 1 - lat_level]
                  RETURNS ARRAY of winv
                          ARRAY of coainv
                  END FOR;

wiy, coaiy :=  wix || winv,  coa || coainv;        % size iy; of North & South

deltaiy, siaiy, wocsiy :=     % size iy; of North & South
          FOR lat_level IN 1, iy
                  deltai :=  ACOSR(coaiy[lat_level]);
                  siai :=  SIN(deltai);
                  wocsi :=  wiy[lat_level] / (siai * siai);
          RETURNS ARRAY of deltai
                  ARRAY of siai
                  ARRAY of wocsi
          END FOR;
wocsilath, wilath :=
          IF iglobe = 2       % Indeed, highlight the South
          THEN wocsiy, wiy

          ELSE    FOR lat_level in 1, ilath
                  wocsiyhalf := 2.0 * wocsiy[lat_level]
                  RETURNS ARRAY of wocsiyhalf
                  end FOR
                  || ARRAY_adjust(wocsiy, ilath + 1, iy),

                  FOR lat_level in 1, ilath
                  wiyhalf := 2.0 * wiy[lat_level]
                  RETURNS ARRAY of wiyhalf
                  end FOR
                  || ARRAY_adjust(wiy, ilath + 1, iy)
          END IF;

alp_double :=
   FOR lat_level IN 1, ilath
    alp_LGN := legendre(ir, irmax2, jxxmx, coaiy[lat_level],
                              siaiy[lat_level], deltaiy[lat_level]);
  RETURNS ARRAY of alp_LGN
  END FOR;                          % arraysize [iyh levels, spectral_indices]

alp := SasAlfa(ir, irmax2, jxxmx, ilath, alp_double);

constant := grav / asq;

% Here is the Trouble Spot:

% When these two are put out seperately, iflopt disallows:
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;

% The inexplicable solution:
% 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;

hnew := IF ilin = 0          % Indeed
        THEN h
        ELSE    ARRAY_fill(1, jx, zero) || ARRAY_adjust(h, jx + 1, jxmx)
        END IF;

p, c_taken, z := FOR row IN 1, jxmx
          p, c, z:= IF row > 256 THEN zero, zero, zero
                                       ELSE p_in[row], c_in[row], z_in[row]
                    END IF;
          RETURNS ARRAY of p
                  ARRAY of c
                  ARRAY of z
          END FOR;

c :=    IF istart = 0 THEN ARRAY_FILL(1, jxmx, zero)        % Indeed
        ELSEIF ARRAY_SIZE(c_in) = 0 THEN ARRAY_FILL(1, jxmx, zero)
        ELSE c_taken
        END IF;

znew := IF istart = 0                 % Indeed
          %------------------------------------Linear Balance Equation
        THEN
                For m IN 1, mx
                zjm :=
                  FOR j IN 1, jx
                  jm := kmjx[m] + j;
                  jmx := kmjxx[m] + j;
                  realn := real(m + j - 2);
                  realn1 := realn + 1.0;
                  zj:=IF (j = 1 & m = 1) THEN zero
                        ELSEIF (j = jx & m = mx)
                        THEN Crmul( - tw / realn / realn * epsi[jmx], p[jm - 1])
                        ELSE Crmul( - tw / realn / realn1,
                            Cadd(Crmul(realn1 / realn * epsi[jmx], p[jm - 1]),
                            Crmul(realn / realn1 * epsi[jmx + 1], p[jm + 1])))
                        END IF
                  RETURNS ARRAY of zj
                  END FOR
                RETURNS VALUE of CATENATE zjm
                END FOR
          %----------------------------
        ELSEIF array_size(z_in) = 0 THEN ARRAY_FILL(1, jxmx, zero)
        ELSE z
        END IF;
pm := p;
p1 :=   FOR j IN 1, jxmx
        RETURNS ARRAY of p[j].Repart
        END FOR;
cm := c;
zm := znew;
th_time_step:=1;
```

```
IN
1,
mx, jx, jxx, ilin, mx2, jxmx, jxxmx, nfax, ilath, imp,
istart, idumpt, ir, irmax2, ires, ix, ixh, iy, delt, ilong,
izon, ifirst, th_time_step,
hdiff, hdrag, tw, zmean, vnu, asq, ww, grav,
kmjx, kmjxx, ksq, ifax, epsi, wocsilath, alp,
p, c, znew, hnew, p1, pm, cm, zm,
trigb, trigf
end let
end function
```



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 -l -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

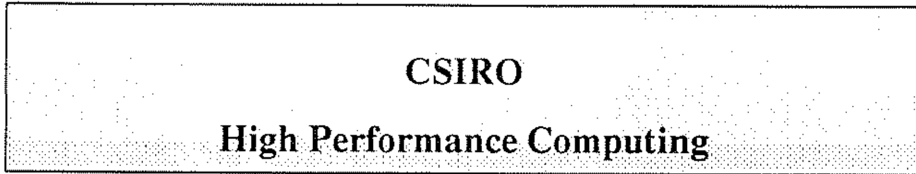
[1] Pau S. Chang and Greg K Egan, "An Parallel Implementation of a Barotopic Spectral Numerical Weather Prediction Model in the Functional Language SISAL", SIGPLAN Notices, Vol. 25, No. 3, March, 1990, pp. 109-117, Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming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



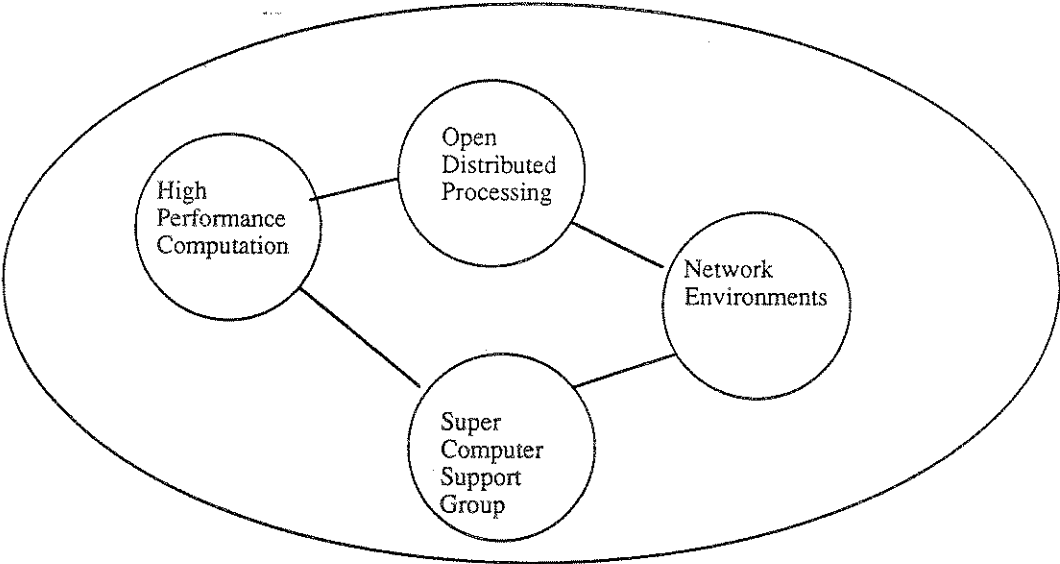
Directions - a short statement

Dr David Abramson

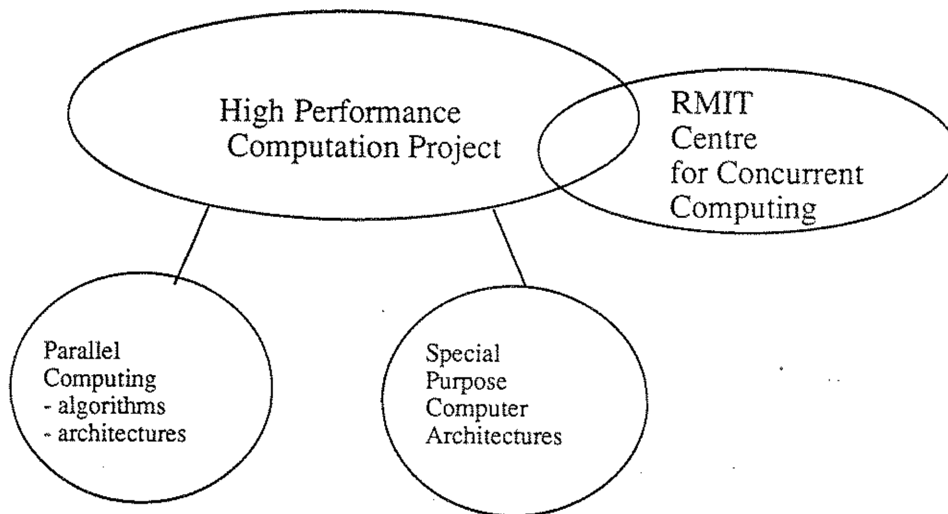
Some Comments on SISAL

David Abramson
Commonwealth Scientific and Industrial Research Organisation
Division of Information Technology

High Performance Computation and Communications



High Performance Computation



Language Requirements

Any language that we might use, should possess some or all of the following attributes:

Expressive Power
High Level features
Efficiency
Simplicity
An incremental paradigm
Portability

Languages

Currently we would consider the following languages as part of a tool kit:

SISAL
C/Linda
C/Pascal & Argonne
Fortran & Sched

Language/Requirements?

	Expressive Power	High Level features	Efficiency	Simplicity	Incr paradigm	Portability
SISAL	3	4	2	3	1	4
C/Linda	3	3	4	4	5	5
C/Argonne	2	3	5	4	5	5
Fort Sched	1	2	?	4	5	5

Problems with SISAL

Efficiency
- Memory Allocation/Deallocation problems

Expressive power
- Strict Arrays
- Parallel Array Operations

Inc Paradigm
- Lack of familiarity

Applications and their requirements

	Demands
Simulation	Multi Lingual(Coded in C), Efficiency
Molecular Structure	Multi Lingual(Coded in Fortran), Large Memory
Image Processing	Very Large structures
CSIRAC II	Needs determinate 'function' language
DSP	Streams? Efficiency

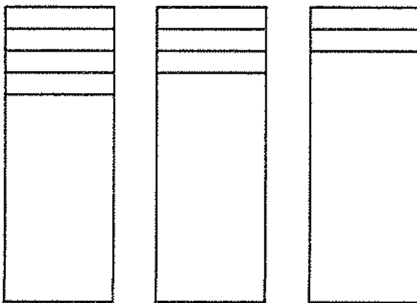
BUT

ADVANTAGES OF SISAL MUST OUTWEIGHT
THE DISADVANTAGES (Incremental Paradigm)

Strict Arrays

Streams don't solve all of the problems.

Consider the primes example:



Merge Operator

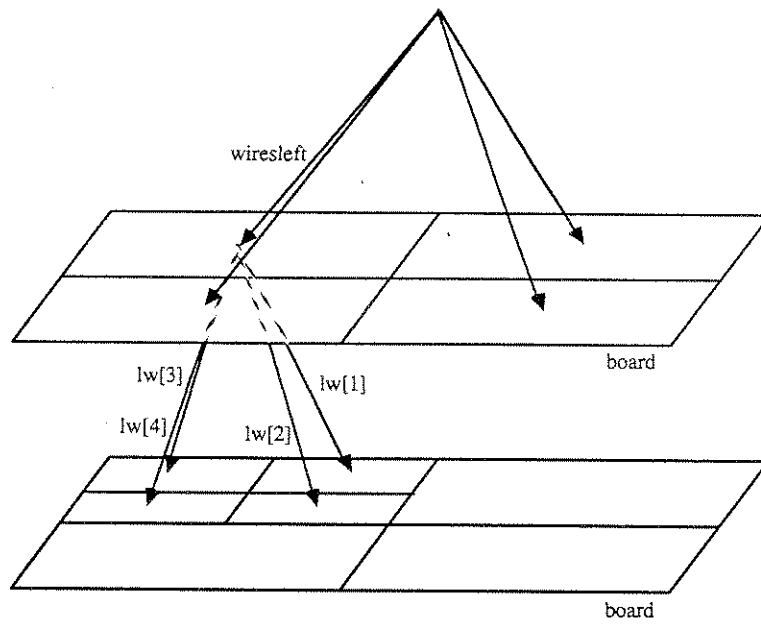
```
do 10 i = 1,updatearraysize

        mainarray(updatearray(i)) = 0

10  continue
```

FORTRAN

```
for initial i := 0;mainarray := inputarray;
    while i <= updatearraysize repeat
        i := old i + 1;
        mainarray := old mainarray[updatearray[i]:0];
returns value of mainarray
end for
```

SISAL

Recursive Subdivision



Recursive Splitting

```
function route (   board: Grid;
                   wiresleft :Wirelist;
                   bounds :corners;
                   returns Grid, Wirelist)

   if keepsplitting(...) then
       let
       newbounds := computecorners(bounds);
       newwires:= splitwires(wiresleft);
       newboard, wiresnotdone :=
       for corner in 1,4
            newboard,wires:= route(board,newwires[ corner ],newbounds[ corner ]);

            returns array of newboard, value of catenate wires
       end for;

       currentwiresleft := newwires[5] || wiresnotdone; %5th element are wires held at this level
       currentboard := mergeboard(newboard);
     in
       processboard(currentboard,currentwiresleft, bounds)
       end let
   else
       processboard(board,wiresleft, bounds)
   end if

end function;
```

Histogramming

```
do 10 i = 1,samplesize
   compute(histaddr)

   histogram(histaddr) =
   histogram (histaddr)+1

10 continue
```

FORTRAN

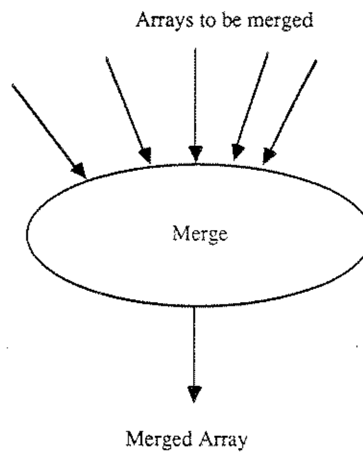
```
for initial i := 0;histogram := array_fill(0,maxhist,0);
      while i <= samplesize repeat
               histaddr := compute(...);
            newval := old histogram [histaddr] + 1;
      histogram := old histogram [histaddr:newval];
               i := old i + 1;
            returns value of histogram
      end for
```

SISAL

Arbitration

```
% conflict_vector is initialised to a large value

let

 resource_list :=
      for w in 1,numworkers
         resource := choose(w);
      returns array of resource

 end for;   %build list of chosen resources, one for each worker

  in

 for initial
      w := 0;
    conflict_vector := array_fill(0,numworkers,big_number);
 while w < numworkers repeat

 % process resources, giving priority to low numbered workers

      w := old w + 1;
      conflict_vector := if old conflict_vector[resource_list[w]] > w then
         old conflict_vector[resource_list[w]:w] else
         old conflict_vector
      end if
      returns value of conflict_vector
 end for
 end let

% Now proceed to check whether successful
```

Merge Operator



Semantics

1) If all of the corresponding elements of the arrays to be merged have the same value, then the element of the reduced array is the same as the corresponding elements in the original arrays.

2) If one element differs from all other corresponding elements in the arrays to be merged then the differing value is placed in the reduced array

3) If more than one element differs from corresponding elements in the merged arrays, then the corresponding cell in the reduced array is set to *error value*.

New Code

```
for i  in updatearraysize

    reducedarray := mainarray[updatearray[i]:0];
    returns value of merge reducedarray

end for
```

New Splitting code

```
function route (    board: Grid;
                    wiresleft :Wirelist;
                    bounds :corners;
                    returns Grid, Wirelist)

    if (keepsplitting) then
        let
        newbounds := computecorners(bounds);
        newwires:= splitwires(wiresleft);
        currentboard, wiresnotdone :=
        for corner in 1,4
            newboard,wires:= route(board,newwires[ corner ],newbounds[ corner ]);

            returns value of merge newboard, value of catenate wires
        end for;

        currentwiresleft := newwires[5] || wiresnotdone;
        in
        processboard(currentboard,currentwiresleft, bounds)
        end let
    else
        processboard(board,wiresleft, bounds)
    end if

end function;
```

New Histogram

```
for i  in samplesize

    histaddr := compute(...);
    newhistogram :=  histogram [histaddr: 1];

    returns value of sum merge newhistogram

end for
```

New Arbitration

```
% process resources, giving priority to low numbered workers

    for w in 1,numworkers

    newconflict_vector := conflict_vector[resource_list[w]:w]
       returns value of least merge newconflict_vector
 end for

% Now proceed to check whether successful
```

University of Adelaide

Directions - a short statement

Dr Andrew Wendelborn

Functional Programming in SISAL

Dr. A. L. Wendelborn

Objective: program development in a parallel environment

- parallel machine
- parallel software tools: applicative parallelism
- parallel thinking

Specific projects

- multiprocessor implementations of SISAL
- input/output in functional languages
- compiler construction in SISAL
- SISAL-level debugger for OSC
- object-oriented implementation of OSC

Multiprocessor Experiments

Translation of SISAL to Leopard-2 multiprocessor
  initial experiments on Encore Multimax

SISAL - IF1 - C translation with OSC

- threads implementation
- interactive i/o
- stream operations
- storage allocation
- port to Leopard 1/2
- program tuning

SISAL in SISAL Compiler

targetted to CSIRAC II
- significant non-numeric application
- 5 modules, 8300 lines

investigate effectiveness of SISAL programming
- for expressing parallelism
- from software engineering viewpoint

parallel compilation
- new algorithms for parallel lexical analysis developed
- other aspects under investigation

prototype compiler developed

performance comparison on CSIRAC II, Manchester and OSC

future

OSC as basis for
- port to Leopard
- tuning experiments etc

i/o in functional languages
- compare SISAL (1&2), Haskell etc

impact of SISAL 2.0 ???
- critique

University of Adelaide

Sisal on the Encore and Leopard Multiprocessors.

Hugh Garsden

SISAL(OSC) on ENCORE and Leopard-1

Hugh Garsden
University of Adelaide

OSC Stream Buffering

Wanted to get two interactive programs running -

- a character stream processor
- a primitive "shell" program, which takes a line of input and returns a prompt

Found there were two problems -

- terminal input is buffered by the terminal driver; this means that input does not reach the program immediately, so output is also delayed
- OSC internal stream buffering is not correct, selection of certain stream configurations can make the program behave incorrectly

Stream I/O Buffering

- Want input typed at the terminal to be made available immediately
- The OSC option -sb1 is supposed to do this, but it doesn't do enough
- Need to get rid of buffering by the terminal driver - use `ioctl` UNIX system call
- Must be called from within OSC program. It is used to set the terminal driver to CBREAK mode.
- Program can now access characters as soon as they are typed
- Add `-cbreak` option to OSC so CBREAK can be enabled at run-time
- OSC output is buffered only internally - for convenience added an option to control this independently of input

Internal Stream Buffering

Consider the 'buff' program

We wanted to have characters passed through the program one at a time

If running with > 1 Worker, this means must set consumer wakeup threshold to 1. Use OSC option -st1.

If running with 1 Worker, must also force producer to block after 1 element is produced. Set stream buffer size to 1. Also Use OSC option -ss1.

First case worked ok. Second case produced deadlock.

Internal stream buffering (cont.)

Examination of the stream code showed that it contained a bug

As implemented, the producer of a stream blocked just before the buffer is filled, instead of just after

This explained the deadlock, producer had the next stream element but did not place it in the buffer. It blocked, and consumer was not woken.

In general this happens when options -ssN -stN are used on OSC

Once the problem was tracked down, it was simple to modify some stream macros to fix it

```
 % This program contains a few quirks to get it to run.
 % It splits a stream of characters into lines, in a way that OSC
 % can handle. Each line is built incrementally from the input stream,
 % when a \n is reached, a complete line is returned.
define main

type schar = stream[character];
type achar = array[character]

function process (line : achar returns character)
    % If line = "yes" then '*' else '>' end if
       .....
end function

function main (input : schar returns schar)
  for initial
    processed_line := '>';    % initial prompt
    line := "";
    s := input;
    got_a_line := true  % hit RETURN produces initial prompt
  while ~stream_empty(s) repeat
    ch := stream_first(old s);
    s := stream_rest(old s);

 % If ch = '\n', then reset line to
 % empty, otherwise add ch to the end of the current line.

    line := if ch = '\n' then "" else array_addh(old line,ch) end if;

  % Peek ahead, should check if s is empty, but doesn't seem to mind.
  % Peek ahead is required to get prompt to come out at the right
  % time

    got_a_line := stream_first(s) = '\n';
    processed_line := if got_a_line then process(line) else ' ' end if
                                                          % ^ never used
  returns
    stream of processed_line when got_a_line
  end for
end function
```

Prompt

```
define main

type schar = stream[character]

function main (s : schar returns schar)
  for el in s
    ord := integer(el);
    new_el := if ord > 96 & ord < 123  % is el in range a-z
              then character(ord-32)   % if yes then make
                                       % upper case
              else character(ord)
              end if
    returns stream of new_el
  end for
end function
```

Buff

Running the programs

```
medusa> buff -cbreak -st1 -ss1 -sbo1
ENCORE SISAL 1.2
aAbBcCdDeEfFgG^Dmedusa>
```

The input was "abcdefg^D".

```
medusa> prompt -st1 -ss1 -sbo1
ENCORE SISAL 1.2

>a line
>another line
>yes
#a line again
>last line
>^Dmedusa>
```

The input was
"\na line\nanother line\nyes\naline again\nlastline\n^D".
The initial \n is required to get the first prompt.

Unimplemented Stream Ops

Many stream operations are not implemented by OSC

Examples -

- value of stream
- stream catenenate
- stream append
- stream prefixsize

Decided to implement some of these, initially stream append and stream catenate

Problem - lack of documentation on OSC back end phases

Strategy - implement append and catenate in terms of some other nodes as a familiarisation exercise, then do them "properly"

Preliminary implementation

Consider stream catenate example only

Replace

```
s1 || s2 || s3
```

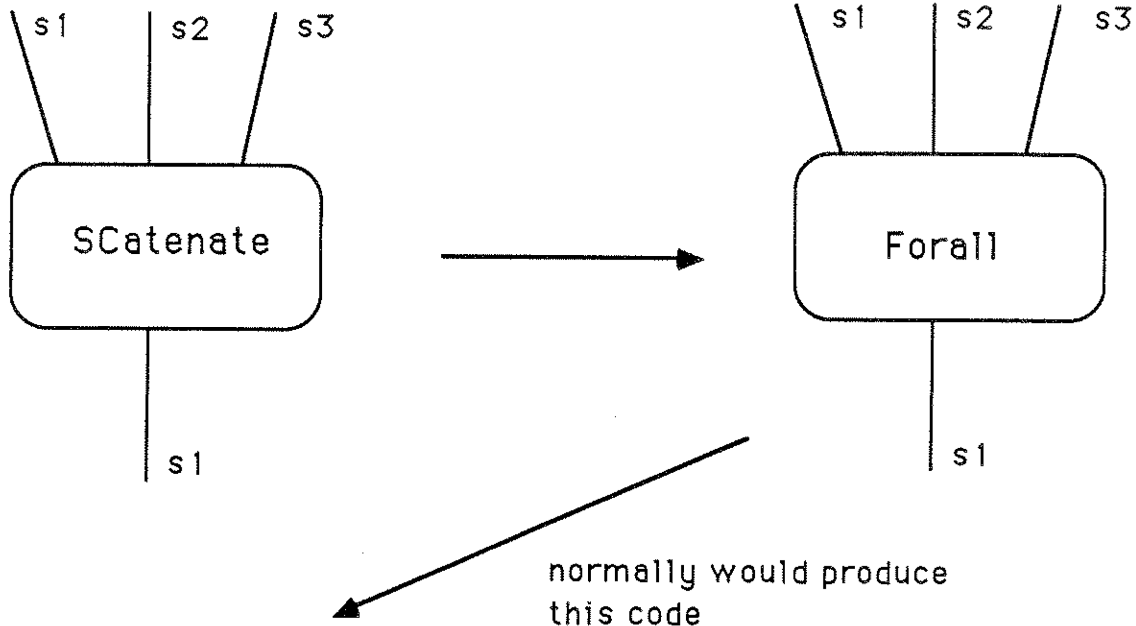
by

```
for e1 in s1 dot e2 in s2 dot e3 in s3
returns stream of e1
end for
```

This loop takes three streams and produces an output consisting of the first. It throws the other two away.

Do the transformation at the IF1 level, during the load phase of OSC

A stream task will produced for the loop. At the C gen phase, tweak the C code to effect a catenate.




```
SGathInit( tmp4,, ((struct Args2*)args)->Out1 );
tmp5 =  ((struct Args2*)args)->In3;
tmp7 =  ((struct Args2*)args)->In2;
tmp9 =  ((struct Args2*)args)->In1;
for ( ;; ) {
   SScat( char, tmp6, tmp5 );
   SScat( char, tmp8, tmp7 );
   SScat( char, tmp10, tmp9 );
   SGathUpd( char, tmp4, tmp10 );
   }
SetEos( tmp4 );
DeallocStream( tmp5, 7 );
DeallocStream( tmp7, 7 );
DeallocStream( tmp9, 7 );
```

but is
modified to this

```
SGathInit( tmp4, ((struct Args2*)args)->Out1 );
tmp5 =  ((struct Args2*)args)->In3;
tmp7 =  ((struct Args2*)args)->In2;
tmp9 =  ((struct Args2*)args)->In1;
for ( ;; ) {
   SScat( char, tmp10, tmp9 );
   SGathUpd( char, tmp4, tmp10 );
}
for ( ;; ) {
   SScat( char, tmp8, tmp7 );
   SGathUpd( char, tmp4, tmp8 );
}
for ( ;; ) {
   SScat( char, tmp6, tmp5 );
   SGathUpd( char, tmp4, tmp6 );
}
SetEos( tmp4 );
DeallocStream( tmp5, 7 );
DeallocStream( tmp7, 7 );
DeallocStream( tmp9, 7 );
```

Proper implementation

On examination of the code, it turns out that stream catenate is in fact implemented in the Backend phases, but there is no C gen for it

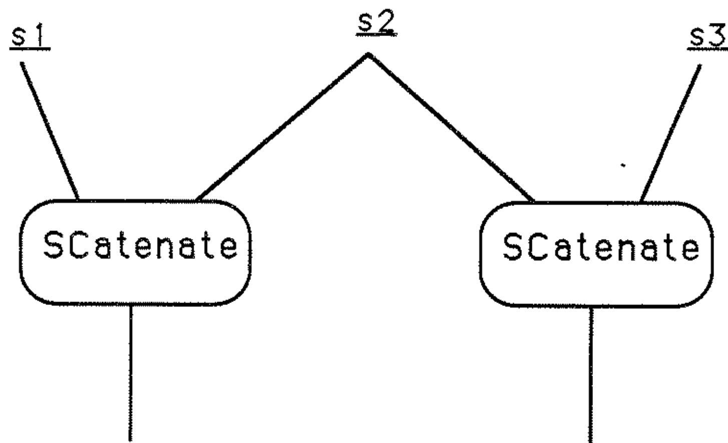
The problem is how to implement it in run-time; implementation of OSC operations is not straight-forward, what to do depends on pragmas and other information contained on edges, you need to know how to interpret this information

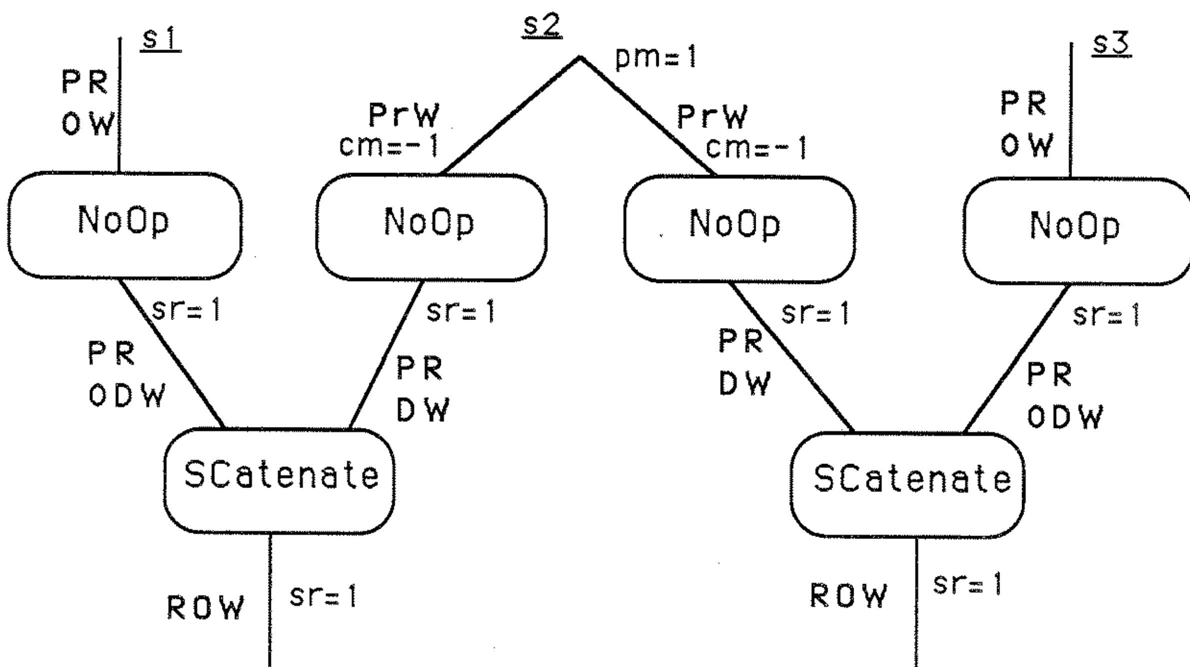
A study of the available documentation explains most of what is happening. It appears as though it is simply a matter of linking stream buffers.

In some cases an input stream cannot be operated on in place - it must be copied. It turns out that the code already exists to do this.

Will add run-time code to link buffers based on ACatenate

```
define main

type schar = stream[character]

function main (s1, s2, s3 : schar returns schar, schar)
  s1 || s2, s2 || s3
end function
```





Experiments with lazy deallocation

Consider a loop in a SISAL program

During execution of each iteration temporary variables may be created and destroyed

This requires allocating memory for them and deallocating it at the end of each loop

Want to investigate the possibility of lazy de-allocation; don't deallocate at the end of each loop but defer it until a Worker becomes idle or boundary tag pool is exhausted

Deallocations can then proceed concurrently with the main work of the program, or may even never happen

```
define main

function gen (returns array[array[integer]])
    % Create 1000 arrays all of size 100
  for i in 1, 1000 cross j in 1, 100 returns array of j
  end for
end function

function main (returns integer)
    % Iterate through the arrays, doing some work which
    % requires a temporary array
  for initial
    i := 1;
    aa := gen();
    sizes := 0;
  while i <= array_size(aa) repeat
    tmp := array_addh(aa[old i],0);   % temp value
    sizes := array_size(tmp) + 1;
    i := old i + 1
  returns
    value of sum sizes
  end for
end function
```

Test program

Preliminary results

Tested the extreme case of lazy deallocation - no deallocation

Found that removing deallocation actually slowed program down, this is because removing deallocation means there are no blocks in the cache - subsequent allocates must go to boundary tag pool

Even in a program that did not use the cache, no deallocation adversely effected the "tidiness" of the boundary tag pool

Results so far are not conclusive, just indicate directions for detailed study

Beyond lazy deallocation, analysis of a program to allow re-use of temporary variable's memory may give best results

Some test results

Time with alloc/dealloc of tmp	0.616
Time with no dealloc of tmp	0.766

	User	System	Total
Time with alloc/dealloc	0.626	0.006	0.632
Time with no dealloc	0.595	0.197	0.792

Time with alloc/dealloc	0.616
Time with re-use of memory	0.375

SISAL on the Leopard-1

Leopard-1 is a prototype of a multiprocessor workstation

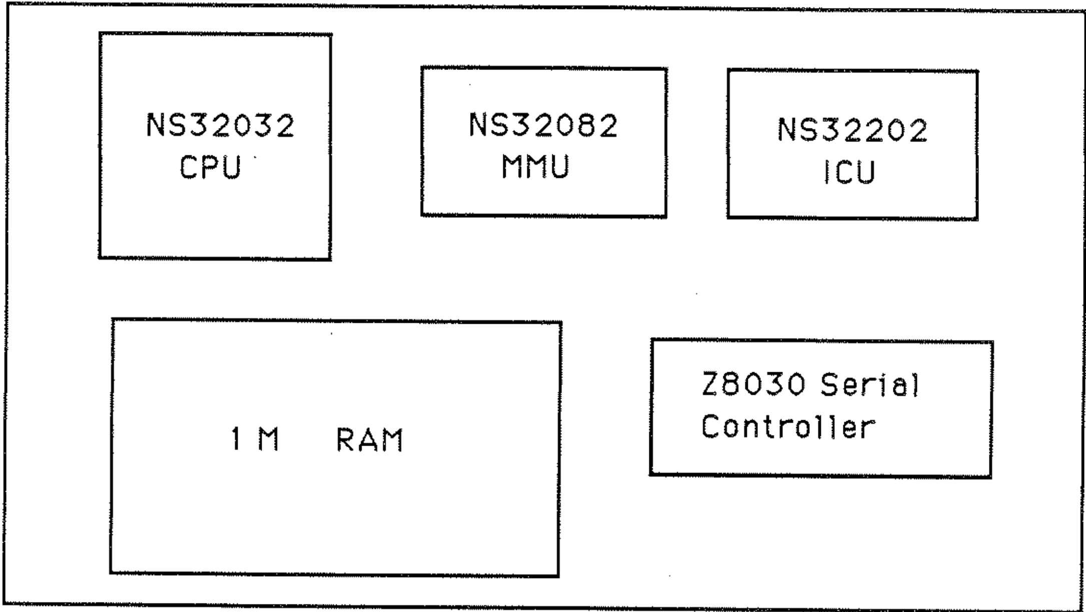
It currently runs the MINIX operating system, but only on one board

MINIX is a rewrite of UNIX version 7, and is used mainly as a teaching tool

There is a project underway to put a multiprocessor version of MINIX on the Leopard-1

It would then be possible to run OSC on it

The advantage with the Leopard-1 is that it is a single user machine.



Processor
Boards

Boards connected to
backplane

Shared memory
board - 2 M RAM

Leopard-1

Porting OSC to MINIX

MINIX implements the basic UNIX system calls and libraries

Those that are not implemented will have to be removed from OSC. Example - `profil`, to do profiling. Most of these will not be important.

One that OSC must have is `share`, this will have to be implemented in MINIX. Requires manipulating page tables.

Context switching code for NS32020 CPU already exist as part of the ENCORE version, so no problem there

MINIX is not a fast OS. Main experiments will be to test relative speedups and performance tuning, in single user environment

Alternatives to Leopard-1/MINIX

It may be possible to run OSC on the bare machine

Replace all system calls in OSC with stubs that execute some primitive operation. Example - can `read` and `write` from/to shared memory.

Will need to download the OSC program to each board seperately and set them running together

The OSC program will have to handle clock interrupts and keep track of time

Leopard-2 is currently under development. This is a 4 processor workstation running Chorus.

Chorus is a multiprocessor version of UNIX System V. Has similarities to MACH. Provides threads.

Could run OSC on Leopard-2/Chorus

An Overview of SISAL 2.0

Andrew L Wendelborn
Department of Computer Science
University of Adelaide

arrays

multi-dimensional arrays may be defined explicitly (not as array of array)

```
type TwoI = array [..,..] of integer    cf
type TwoI = array [array [integer]]
```

very different expression of item and subarray
selection update generation

```
A[3,..]          selects row 3 of a 2D array
A[2..4, 2..3 : v]     update 3*2 section of an array
array [ i in 1..size(A) : A[i] ]    generate copy
```

array cast to change bounds

```
<2,5> array [ j in 0..3 : j*j ]
```

reductions

SISAL 1.2 provided inbuilt reductions over arrays and streams
    `value of sum/product/least/greatest/catenate`

replaced with ability to define reductions
- syntax similar to function definition
- specifies
  - parameters
  - initial values
  - transformation at each reduction step

particularly useful to carry several items of information along the reduction
- e.g. position of greatest element

```
reduction max1( V:double; K:integer returns double,integer )
 initial
   Vacc := -1D0; Kacc := 0;
 in
   if ( V > Vacc ) then
     V, K
   else
     Vacc, Kacc
   end if
end reduction
```

loops

sequential and parallel forms no longer distinguished syntactically

different forms: `for` `while` `until` `do`

revised notation for updating loop variables

```
i := old i + 1
```
now written as
```
new i := i + 1
```

histograms

histogramming comes syntactically for free by combining array construction, reduction, and the AT phrase to specify array element:

```
FOR x IN a
RETURN
   ARRAY[1..m] OF REDUCE sum(1) AT g(x)
END FOR
```

array size, reduction operator, and AT phrase must all be present

`g(x)` **must be in** `1..m`

`sum` **can be replaced by any predefined or user-defined reduction**

1 can be replaced by some weight factor

streams

stream types are as before
changes in stream generation and inquiry, and substream selection

generation
    from a loop
    by a stream generator expression
        replication `repl( e,n )`
        progression `10..1..-3` (triplet)

selection and inquiry
    single component, or substream defined by a triplet

```
let
    S := stream [4,5,6,7,8,9,10];   n := 7;
in
    S[2..], S[1], S[4..n-1..2], S[n-2]
```

yields

`[5,6,7,8,9,10]` (rest) `4` (first) `[7,9]` `8`

... streams

scalar operations are extended pointwise to work with streams

```
max( stream [12, 4, 35],
     stream [ 8, 9,  3],
        15              )  =    stream [15, 15, 35]
```

operator `suffix` expresses partial consumption of stream

```
word, new instream :=
   for char in instream
   while char <> ' ' do
   return
      reduce catenate(char),
      suffix instream
```

modules

module facility similar to that of Modula-2 or Oberon
separate compilation; organization; re-use

compilation units
`definition`
public interface - types and functions
also specify *imports* from other modules
`module`
completes accompanying `definition`
completely defines all public items

```
DEFINITION MatrixRoutines;
  TYPE TwoDim = ARRAY [..,..] OF TYPE;
  FUNCTION MatMult( A, B: TwoDim RETURNS TwoDim );
  FUNCTION Transpose( A: TwoDim RETURNS TwoDim );
END DEFINITION;

MODULE MatrixRoutines;
  FUNCTION Matmult( A, B: TwoDim RETURNS TwoDim );
        ...
  FUNCTION Transpose( A:TwoDim RETURNS TwoDim );
        ...
END MODULE;
```

mixed language programming

SISAL 2.0 defines module interfaces to foreign language routines

language specified in e.g. `definition lib in Modula-2;`

procedures with persistent state are handled thus:

value of `type state`
embodies all persistent state required by routines in a `definition`
associated function `instance` obtains a new state value
routines have extra `state` value to reflect state change

SISAL interface requires
inclusion of state parameter in declaration
specification of parameters as `in` `out` `inout`

the arity of a foreign procedure treated as a SISAL function is the number of `out+inout` parameters in its declaration

requires special foreign language compiler and loader to allocate and reference state storage

higher order functions
polymorphism

curried functions now permitted

no examples available, but can guess from syntax spec.

```
type ft = function ( a:t returns t )
    where t is any type (including another function type)
let
    f( a:t returns t ) := a
in  f(5)
end let
```

similarly, few examples of polymorphism

```
TYPE TwoDim = ARRAY [..,..] OF TYPE;
```
in definition/module

```
TYPE TDR = ARRAY[..,..] OF REAL;
FUNCTION MatMult( A, B: TDR RETURNS TDR )
```
in program

miscellaneous

case construct

```
case word of
   "data","flow" : c-10;
   otherwise     :  0;
end case
```

type conversions automatic for

```
double -> real -> integer
```

operator `**`

union selection with dot notation

```
a . tag
```

D.2 Sieve of Eratosthenes

```
% This program returns the primes between 2 and Limit using the Sieve of
% Eratosthenes.

definition MathF77 in FORTRAN;
  function dsqrt( a:double returns double );
end definition

Program PrimesExample;

  from MathF77: dsqrt;

  type StrmInt = stream of integer;

  function Filter( S:StrmInt; M:integer returns StrmInt )
    for I in S do
    returns stream of I unless mod( I, M ) = 0
    end for
  end function

  function Sieve( Limit:integer returns StrmInt )
    for
      S := stream[ 2..Limit..2 ];
      Maxt := integer( dsqrt( double( Limit ) ) );
    until empty( S ) do
      T := S[1];

      new S := if T <= Maxt then Filter( S[2..], T )
               else S[2..] end if;
    returns stream of T
    end for
  end function

end program
```

D.5 Matrix Multiply

```
% This program illustrates the definition and use of a matrix package
% comprising matrix multiply and matrix transpose operations.

definition MatrixRoutines;
  type TwoDimI = array [..,..] of integer;
  type TwoDimD = array [..,..] of double;
  type TwoDimR = array [..,..] of real;
  type TwoDim  = array [..,..] of type;

  function MatMult( A,B:TwoDim; M,N,L:integer returns TwoDim );
  function Transpose( A:TwoDim; M,N:integer returns TwoDim );
end definition;

module MatrixRoutines

  function Matmult( A,B:TwoDim; M,N,L:integer returns TwoDim );
    for i in [1..M] cross j in [1..L] do
      S := for k in [1..N] do
           returns reduce sum( A[i,k] * B[k,j] )
           end for
    returns array [..,..] of S
    end for
  end function

  function Transpose( A:TwoDim; M,N:integer returns TwoDim );
    for i in [1..M] cross j in [1..N] do
    returns array [1..N,1..M] of A[i,j] at [j,i]
    end for
  end function

end module;

program MatrixMultiplyExample;

  from MatrixRoutines: MatMult, TwoDimD;

  function MatMult( A,B:TwoDimD; M,N,L:integer returns TwoDimD )
    MatrixRoutines.MatMult( A,B, M,N,L )
  end function

end program
```

May 6 17:20 1990

% This is a transcription of the program provided in the "Sample Programs"
% section of the "SISAL Reeference Manual, Version 2.0"
%

```
program GaussjExample;
```

% This program uses Gauss-Jordan elimination to solve A*x=B for x.

```
  type Onei    = array [..] of integer;
  type Twod    = array [..,..] of double;


  reduction max1( V:double; K:integer returns double, integer )
  initial
    Vacc := -1D0; Jacc := 0; Kacc := 0;
  in
    if ( Vacc > V ) then
      V, K
    else
      Vacc, Kacc
    end if
  end reduction

  reduction max2( V:double; J,K:integer returns double, integer, integer
  initial
    Vacc := -1D0; Jacc := 0; Kacc := 0;
  in
    if ( Vacc > V ) then
      V, J, K
    else
      Vacc, Jacc, Kacc
    end if
  end reduction

  function GetPivot( N:integer; A:TwoD; IPIV:Onei
                     returns double, integer, integer )
    for I in [1..N] do
      V,
      J, NA := if ( IPIV[I] ~= 1 ) then
                 for J in [1..n] do
                   V := if ( IPIV[J] = 0 ) then
                          abs(A[I,J])
                        else
                          error[double] % SINGULAR!!!
                        end if
                 returns max1( V, J )
                 end for
               else
                 -1.0D0, 0
               end if
    returns max2( V, I, J )
    end for
  end function
```

```
  function DoWork( n,r,c:integer; Ain:TwoD; Bin:OneD returns TwoD, OneD
    let
      A_1,
      B_1   := if ( r ~= c ) then
                   Ain[ r,1..n:Ain[c,1..n]; c,1..n:Ain[r,1..n] ],
                   Bin[ r:Bin[c]; c:Bin[r] ]
                 else
                   Ain, Bin
                 end if;

      Pivinv := 1.0D0 / if ( A_1[c,c] = 0D0 ) then
                              error[double] % SINGULAR!!!
                            else
                              A_1[c,c]
                            end if;

      B     := B_1[ c:B_1[c] * Pivinv ];
      A_2   := A_1[c,c: 1.0D0 ];
      A   := A_2[ c,1..n: for l in [1..n] do
                             returns array of A_2[c,l] * Pivinv
                             end for ];

      Bc := B[c];
      Ac := A[c,..];
    in
      for ll in [1..n] do
        Ar,
        Bv := if ( ll ~= c ) then
                  let
                    Dum := A[ll,c];
                    Row := for l in [1..n] do
                           returns array of  A[ll,l] - Ac[l] * Dum
                           end for;
                  in
                    Row[c:Row[c] + Dum], B[ll]- Bc * Dum
                  end let
                else
                  Ac, Bc
                end if;
      returns array [..,..] of Ar at [ll],
              array [..]  of Bv
      end for
    end let
  end function

  function GaussJ( n:integer; A:TwoD; B:OneD returns OneD )
    let
      IPIV := array[1..n: fill(0)]
    in
      for I in [1..n] do
        big, r, c  := GetPivot( n, A, IPIV );
        new IPIV   := IPIV[ c:IPIV[c] + 1 ];
        new A, new B := DoWork( n, r, c, A, B )
      returns B
      end for
    end let
  end function

end program
```

```
definition  SGE in FORTRAN;

    type onedr = array[..] of real;
    type onedi = array[..] of integer;
    type twodr = array[..,..] of real;

    function SGEFA( inout A: twodr;
                    in    LDA: integer;
                    in    N: integer;
                    out   PVT: onedr;
                    out   INFO: integer );

    function SGESL( in    A: twodr;
                    in    LDA: integer;
                    in    N: integer;
                    in    PVT: onedi;
                    inout B: onedr;
                    in    JOB: integer );
end definition;
```

Figure 5: Definition for Using LINPACK Routines

```
from SGE: onedi, onedr, twodr, SGEFA, SGESL;

type threedr = array[.., .., ..] of real;

function solveall( A: threedr; B: twodr returns twodr );

    for k in liml(A,1) .. limh(A,1) do
        MYA := A[k, .., ..]; N := size(MYA);
        LU, PVT, INFO := SGEFA( MYA, N, N, , );
        MYB := B[k, ..];
        x := SGESL( LU, N, N, PVT, MYB, 0 )
    returns array[..,..] of x
    end for;

end function;
```

Figure 6: Using LINPACK from SISAL

```
definition F in FORTRAN;
   type state;
   function instance( returns state );
   function SUBA( inout X: state; out Z: real; in Y: real );
   function SUBB( inout X: state; out H: real );
   function FUNCC( in Z: real returns real )
end definition F;
```

Figure 3: Definition F for Three FORTRAN Routines

```
from F: state, instance, SUBA, SUBB, FUNCC;

let
    s := F.instance();
    t, a := F.SUBA( s, , 2.3 );
    u, b := F.SUBB( t, a );
    c := F.FUNCC( a );
in ···
```

Figure 4: Using FORTRAN Routines from SISAL