SARA: A Cray Assembly Language Speedup Tool

Robert G. Babb II

Oregon Graduate Center
Department of Computer Science
and Engineering
19600 N.W. von Neumann Drive
Beaverton, OR 97006-1999 USA

Technical Report No. CS/E 89-017

October, 1989

SARA: A CRAY ASSEMBLY LANGUAGE SPEEDUP TOOL

Robert G. Babb II
Department of Computer Science and Engineering
Oregon Graduate Center
19600 NW Von Neumann Dr
Beaverton, Oregon 97006
USA

1. Introduction

SARA (Single Assignment Register Assembler) is an extended form of CAL (Cray Assembly Language) meant for obtaining near optimal performance from relatively short (100's of instruction) Cray X-MP basic block code sequences. The SARA Optimizing Preprocessor (informally referred to also as "SARA") converts SARA source files into a form that is acceptable as input to standard Cray Research Inc. CAL Assemblers. The SARA Optimizing Preprocessor can greatly speed up the job of CAL coding by automating the difficult, tedious, and error-prone tasks of assigning registers and ordering instruction sequences to take maximum advantage of the Cray X-MP architecture.

2. History

Several years ago, while working as a consultant for one of the first commercial customers for the Cray X-MP, I was asked to attempt to "speed up" their computing throughput by coding lower precision (16 bit) scalar and vector "VFUNCTION" versions of x**y and 1/sqrt(x) in CAL, for specified ranges of values of x and y. I accomplished this task manually (achieving speedups ranging from 1.7x to 2.3x for scalar and vector versions over the standard Fortran library functions), but found the job of scheduling instruction sequences for optimal performance on the X-MP to be very difficult. The complex timing and conflict characteristics for Cray-style architectures makes it difficult to remember them all, and people are not very good at producing optimum schedules for instruction streams of this type. The job of scheduling instructions and assigning registers is handled fairly well

by the latest generation of Fortran compilers for Crays, but when coding in Cray Assembly Language, programmers still can not benefit from this capability. SARA was created to help with this problem.

3. Language Syntax

3.1 SARA ON/OFF Blocks

SARA source files consist of alternating "SARA ON" and "SARA OFF" sections. CAL statements in SARA OFF sections are copied unchanged. However, as discussed later, SARA-assigned register values can be referenced symbolically in subsequent SARA OFF code sections. The currently implemented versions of SARA are basic block schedulers only. Branch instructions must occur only in SARA OFF code sections. Within SARA ON blocks, CAL macros or pseudo-ops for storage definition are also diagnosed as unrecognized.

3.2 Pseudoregisters and Single Assignment

Since SARA is an extended form of CAL, SARA input source lines generally follow standard CAL formats, with the addition of "*:" control lines and a notation for "pseudoregisters". In SARA, data dependencies between machine instructions are specified explicitly via the use of pseudoregister identifiers. For example, the SARA form of the CAL scalar floating point add instruction

```
S1  S2+FS3
```

could be represented in SARA as

```
S.X  S.Y+FS.Z
```

A pseudoregister is like a variable that can be assigned a value only once (hence "single assignment"), but whose value (once assigned) can be referenced as many times as necessary. Each SARA instruction therefore typically puts its result into a new pseudoregister. SARA allows a CAL programmer to pretend that the machine has an infinite supply of scalar (S), vector (V), and address (A) registers available. Obviously, many different pseudoregisters will be used in a typical basic block.

One result of the operation of the SARA Optimizing Preprocessor is a mapping from pseudoregister identifiers (X, Y, and Z in the example above) to actual registers. Pseudoregister identifiers can consist of any combination of upper case letters and numeric digits. Identifiers beginning with a letter can be up to 8

characters long, identifiers beginning with a digit can be up to seven characters long, and are prefixed automatically with a "%" character.*[1]

Register assignments in the SARA output file are specified via CAL "SET" statements. For the example above, if SARA assigned the registers S1, S2, and S3 to pseudoregisters S.X, S.Y and S.Z, the SARA output would look like:

```
    .
    .
    .
*:SARA OFF
X         SET       O'1
Y         SET       O'2
Z         SET       O'3
    .
    .
    .
    S.X  S.Y+FS.Z
```

SARA programmers can either let the SARA preprocessor assign specific registers, or they can "pre-assign" specific register numbers. In the example above, suppose that S2 and S3 had been assigned values corresponding to Y and Z outside this SARA ON block. Then the example could be modified to look like:

```
    S.X  S2.Y+FS3.Z
```

to force the correct register assignments.

SARA assigns registers automatically only for the S, V, and A registers. All other register designators (such as T, B, SB, SM, and ST) must have a number pre-assigned by the programmer (T70.XYZ for example). A tag on a T register corresponds to a T register load "lifetime". For most registers the pseudoregister to actual register binding need not be explicitly expressed. The exceptions to this rule are the special registers A0, S0, VL and VM. These registers must always be specified as 'sreg.preg' where sreg is A0, S0, VL or VM, for example: S0.100, A0.FINAL, VL.00, and VM.LEFT.

---

* There are a few other minor restrictions on pseudoregister identifiers. They can not duplicate a symbol (such as a location label) in the same program. SARA does not check for this. If the identifier corresponds to one of the reserved Cray register names:

A0-7, B00-B77, S0-7, T00-T77, V0-V7, SB0-7, SM00-37, ST0-7, CA, CE, CI, CL, MC, RT, SM, VL, VM, or XA

SARA will prefix the name with a "%" character.

3.3 Input Declarations

All pseudoregister operand identifiers used in a SARA ON block must be defined previously in the file. This can occur in one of three ways:

1) defined as a result in a previous instruction in the same block
2) declared explicitly as a "*:IN" input value
3) declared automatically as a *:IN value passed in from a previous SARA ON block.

In case 1), a previously unencountered pseudoregister is automatically defined by its appearance on the left (result) side of a CAL operation instruction. In case 2), a *:IN declaration is used to specify pseudoregister identifiers for previously computed register values that exist at the point that a SARA ON block is entered. In the example above, to specify that Y and Z have been previously computed, the SARA input would look like:

```
*:SARA ON
*:IN    S2.Y
*:IN    S3.Z
   S.X  S2.Y+FS3.Z
    .
    .
    .
```

Note that this floating point add statement could also have been written:

```
   S.X  S.Y+FS.Z
```

since specification of pre-assigned register numbers, once defined in either an assignment statement or a declaration statement, is optional.

The third possibility will be discussed below as part of the explanation of the "*:PASS" and "*:SAVE" SARA directives. Note that it is an error for a pseudoregister identifier to make its first appearance in a SARA source file on the right (operand) side of a CAL instruction, unless it has been declared via a *:IN statement.

3.4 Unites

Although SARA uses the single-assignment paradigm as a basis for its syntax, the "unite" operation allows specifying that two different pseudoregisters be assigned to the same actual register. The unite operation has the form:

r.preg1~preg2 <right-hand-side>

where:

'r' is S, V, or A

'~' is the unite operator,

'preg1' is a "new" pseudoregister identifier, and

'preg2' is a previously defined pseudoregister.

The unite operator forces the SARA Optimizing Preprocessor to assign the same actual register to different pseudoregisters. A unite is necessary, for example, when multiple execution paths exist. The single assignment rule prohibits one pseudoregister from being used for both result value names. However, subsequent statements need to be able to refer to the result of either computation. Uniting the two pseudoregister identifiers allows either name to be used to refer to the result. This mechanism must also be used whenever the effect of an "update in place" is desired, such as within the body of a loop. The simplest example where a unite is required is the scalar shift operation, which *must* be specified using a unite, because of a restriction of the Cray instruction set, as in:

```
S.19       O'40060
S.FXR~19   S.19<O'60
```

3.5 Output Declarations

In order to schedule instructions within SARA ON blocks, the SARA optimizer needs to know which instructions compute results that are "outputs" of the SARA block. The "*:OUT" declaration is used to accomplish this. The *:OUT statement must be placed immediately before the statement that computes the result that is to be an output of the block. If the output is a memory store operation (which does not have a register as a result), the convention is to specify "MEM" as the output. For example:

```
    .
    .
    .
*:OUT   V.XYZ
        V.XYZ       S.ONE!V.24&VM.23
*:OUT   MEM
        ,A0.35,1    V.39
*:OUT   A1.00
        A1.00       A2.ANM1X+A.KK
```

Every instruction in a SARA ON block must either be used to compute a register value that is an output of the block, or a value that is stored as a MEM output. Otherwise, the SARA Optimizing Preprocessor report that one or more instructions are "unconnected".

3.6 PASS and SAVE Declarations

A programmer who desires to pass a pseudoregister identifier/register number association across a SARA OFF section, can substitute "*:PASS" for "*:OUT". An appropriate *:IN statement will be automatically generated when the next *:SARA

ON statement is encountered. The "*:SAVE" declaration is like the *:OUT declaration except that the former is used when to continue a previously PASSed pseudoregister definition to the next SARA ON block. The SAVE declaration can occur anywhere in the block and can be continued across more than one block.

An example of a (non-useful) sequence of SARA blocks illustrating use of *:OUT, *:PASS, and *:SAVE follows:

```
==============
INPUT TO SARA:
==============
   .
   .
   .
*:SARA ON
*:IN   A1.00
*:IN   A2.01
*:OUT  A3.02
       A3.02       A1.00+A2.01
*:PASS A.03
       A.03        A.02
*:PASS A4.XYZ
       A4.XYZ      A.02*A.00
*:SARA OFF
   .
   .
   .
*:SARA ON
*:IN  A5.ST
      A0.XX        A.03+A.XYZ
*:OUT MEM
      ,A0.XX       A.03
*:SAVE A.XYZ
*:SARA OFF
   .
   .
   .
*:SARA ON
      A0.YY        A.XYZ
*:OUT MEM
      ,A0.YY       A.XYZ
*:SARA OFF

===========
SARA OUTPUT
===========
   .
   .
   .
*:SARA ON
* p00a: SARA source:
*
*   1. *:IN   A1.00
```

```
*   2. *:IN   A2.01
*   3. *:OUT  A3.02
*   4.        A3.02       A1.00+A2.01
*   5. *:PASS A.03
*   6.        A.03        A.02
*   7. *:PASS A4.XYZ
*   8.        A4.XYZ      A.02*A.00
*:SARA OFF
%00       SET        O'1
%01       SET        O'2
%02       SET        O'3
XYZ       SET        O'4
%03       SET        O'1
* p26a: SARA CAL - CRAY-XMP   1.6bf5  FBCON=1
*                                                    SN  FRT  FIT    PLAIN CAL
    A.%02 A.%00+A.%01                                4.  2    6     A3 A1+A2
    A.XYZ A.%02*A.%00                                8.  4    4     A4 A3*A1
    A.%03 A.%02                                      6.  2    2     A1 A3
   .
   .
   .
*:SARA ON
* p00a: SARA source:
*
*   1. *:IN  A1.%03          ** PASSED IN **
*   2. *:IN  A4.XYZ          ** PASSED IN **
*   3. *:IN  A5.ST
*   4.       A0.XX       A.03+A.XYZ
*   5. *:OUT MEM
*   6.       ,A0.XX      A.03
*   7. *:SAVE A.XYZ
*:SARA OFF
%03       SET        O'1
XYZ       SET        O'4
XX        SET        O'0
ST        SET        O'5
* p26a: SARA CAL - CRAY-XMP   1.6bf5  FBCON=1
*                                                    SN  FRT  FIT    PLAIN CAL
    A.XX A.%03+A.XYZ                                 4.  2    2     A0 A1+A4
    ,A.XX A.%03                                      6. -1    0     ,A0 A1
   .
   .
   .
*:SARA ON
* p00a: SARA source:
*
*   1. *:IN  A4.XYZ          ** SAVED IN **
*   2.       A0.YY       A.XYZ
*   3. *:OUT MEM
*   4.       ,A0.YY      A.XYZ
*:SARA OFF
XYZ       SET        O'4
YY        SET        O'0
* p26a: SARA CAL - CRAY-XMP   1.6bf5  FBCON=1
*                                                    SN  FRT  FIT    PLAIN CAL
    A.YY A.XYZ                                       2.  2    2     A0 A4
```

```
,A.YY A.XYZ                                    4. -1      0     ,A0 A4
```

In SARA output, such as that shown above, the SARA ON blocks are listed twice--first with comment asterisks (reflecting the original input order), then, following the "SETS", re-ordered and with diagnostic information appended. Any original comment fields are not shown in the re-ordered output CAL code.

Following the transformed and re-ordered CAL instruction are a series of comment fields:

-- The "SN" column gives the SARA input statement number (before reordering).
-- "FRT" stands for First Result Time. This corresponds to the number of clocks after issue that a computational result is available for scalar instructions, or the time the first in a stream of vector result values should be available.
-- "FIT" stands for Forward Issue Time. An estimate of the latest time that the corresponding instruction could issue (in clocks before the end, or T=0) and still have all computational work completed. (This estimate is not always exact, since SARA views time as running backwards!) While some instructions will issue earlier than SARA predicts, the largest (first) FIT gives a good estimate of the running time of overall sequences from beginning to end since instructions on the critical path will issue at the intervals predicted.
-- The column headed "PLAIN CAL" shows the equivalent simple CAL instruction with assigned register numbers filled in.

4.0 Outline of the SARA Optimization Strategy

Scheduling and register assignment by the SARA Optimizing Preprocessor are done in reverse time order (backward in time), in the following steps:

1. Determine the "latest forward issue times" for all instruction trees based only on "binary conflicts". (It is possible to determine for any two Cray X-MP instructions the closest they could issue based on the resources they require).
2. Working backwards in time, "bubble" instructions (and their predecessors) in order to account for:
   --function unit reservations/conflicts
   --operand register reservations/conflicts

--issue time conflicts

--register assignment conflicts

The last item is treated as co-equal to the other conflicts, and corresponds to exhausting available register "bandwidth".

3. Choose an instruction to schedule based on a combination of global and local criteria. "Local" means that instructions are chosen as candidates for scheduling from among the set of all mutually conflicting instructions whose issue could tie up a resource at a particular issue time slot. The global criteria is that we bubble the instructions which would have the least worst effect on the resulting set of changes to earliest forward issue times. The remaining instruction is "frozen" (scheduled).

SARA does not guarantee that the resulting schedules are optimal (this is an NP-complete problem [1]) but has performed very well against good hand-coded CAL, and does better than most production compilers. For other studies of various aspects of the instruction scheduling problem for Cray architectures, see [2], [3], and [4].

5.0 Examples

5.1 Scalar Code Example

SARA preprocessing for scalar code sequences tends to result in an instruction stream that is "issue-time limited". This means that a new instruction will issue at (almost) every available clock tick. Some Cray instructions take more than one clock to issue. An example, taken from an experiment to see whether SARA could be used to automatically improve the output of a compiler, is shown below:

```
*:SARA ON
* p00a: SARA source:
*
*   1.          S.W0       W,
*   2.          A.J0       J,
*   3.          S.W1~W0    S.W0>A.J0
*   4.          S.M07      <07
*   5.          S.K0       S.W1&S.M07
*   6. *:OUT MEM
*   7.          K,         S.K0
*   8.          A.AK       S.K0
*   9.          S.C0       C,A.AK
*  10.          S.M01      <01
*  11.          S.C1       S.M01+S.C0
*  12.          A.AK2      S.K0
*  13. *:OUT MEM
*  14.          C,A.AK2    S.C1
```

```
*  15.          S.J1       J,
*  16.          S.MM01     <01
*  17.          S.J2       S.MM01+S.J1
*  18. *:OUT MEM  AFTER: S.J1
*  19.          J,         S.J2
*  20.          S.C57      57
*  21. *:OUT    S0.TEST
*  22.          S0.TEST    S.C57-S.J2
*:SARA OFF
W0        SET        O'3
J0        SET        O'3
J1        SET        O'1
W1        SET        O'3
M07       SET        O'5
K0        SET        O'2
AK        SET        O'2
C0        SET        O'4
MM01      SET        O'7
C57       SET        O'6
J2        SET        O'5
M01       SET        O'3
TEST      SET        O'0
C1        SET        O'1
AK2       SET        O'1
* p26a: SARA CAL - CRAY-XMP   1.6bf5  FBCON=1
*                                                    SN  FRT  FIT    PLAIN CAL
*
    S.W0 W,                                          1. 14    39    S3 W,
    A.J0 J,                                          2. 14    37    A3 J,
    S.J1 J,                                         15. 14    24    S1 J,
    S.W1 S.W0>A.J0                                   3.  3    23    S3 S3>A3
    S.M07 <07                                        4.  1    21    S5 <07
    S.K0 S.W1&S.M07                                  5.  1    20    S2 S3&S5
    A.AK S.K0                                        8.  1    19    A2 S2
    S.C0 C,A.AK                                      9. 14    17    S4 C,A2
    S.MM01 <01                                      16.  1    14    S7 <01
    S.C57 57                                        20.  1    12    S6 57
    K, S.K0                                          7. -1    10    K, S2
    S.J2 S.MM01+S.J1                                17.  3     9    S5 S7+S1
    S.M01 <01                                       10.  1     7    S3 <01
    S.TEST S.C57-S.J2                               22.  3     6    S0 S6-S5
    J, S.J2                                         19. -1     4    J, S5
    S.C1 S.M01+S.C0                                 11.  3     3    S1 S3+S4
    A.AK2 S.K0                                      12.  1     2    A1 S2
    C,A.AK2 S.C1                                    14. -1     0    C,A1 S1
```

In this case, the speedup (measured with Cray X-MP performance monitor [5]) over the original compiler code was 2.3x.

5.2 Vector Code Example

Optimum instruction schedules for the Cray X-MP typically depend upon vector length. Unless directed otherwise (through a `*:VL` directive), SARA schedules for `VL 32`. (The directive was put in for this example, even though unnecessary, to

allow direct comparison of statement numbers with the next example after this one).

```
*  SARA version  1.6bf5   CRAY-XMP. FBCON=1
* Mandelbrot Set Test Case
         IDENT     BENOIT
BENOIT   PROGRAM
         A2        32
         VL        A2
*:SARA   ON
* p00a: SARA source:
*
*    1. *:VL     32
*    2. *:IN     A7.00     CURRENT VECTOR START ADDRESS FOR Z
*    3. *:IN     A6.01       "       "      "      "     "  LAMBDA
*    4. *:IN     A5.02       "       "      "      "     "  ITER
*    5. *:IN     T70.04    - BOXSIZE PARAMETER
*
*    6.          A0.11     A7.00           BASE FOR Z
*    7.          A2.12      2              STRIDE FOR COMPLEX
*    8.          V.XP      ,A0.11,A2.12    XP IS REAL(Z)
*    9.          A0.13     A7.00+1
*   10.          V.YP      ,A0.13,A2.12    YP IS AIMAG(Z)
*   11.          A0.15     A6.01
*   12.          V.RL      ,A0.15,A2.12    RL IS REAL(LAMBDA)
*   13.          A0.17     A6.01+1
*   14.          V.RM      ,A0.17,A2.12    RM IS AIMAG(LAMBDA)
*   15.          V.20      V.RL*RV.XP
*   16.          V.21      V.RM*RV.YP
*   17.          V.SUBX1   V.20-FV.21
*   18.          V.22      V.RL*RV.YP
*   19.          V.23      V.RM*RV.XP
*   20.          V.SUBX2   V.22+FV.23
*   21.          S.30      1.
*   22.          V.31      S.30-FV.XP
*   23.          V.32      V.31*RV.SUBX1
*   24.          V.33      V.YP*RV.SUBX2
*   25.          V.XN      V.32+FV.33
*   26.          V.34      V.31*RV.SUBX2
*   27.          V.35      V.YP*RV.SUBX1
*   28.          V.YN      V.34-FV.35
*   29.          A0.40     A5.02           BASE FOR ITER
*   30.          V.ITERI   ,A0.40,1
*   31.          S.41      1
*   32.          V.ITERN   S.41+V.ITERI
*   33.          V.42      -FV.XP
*   34.          VM.43     V.XP,M
*   35.          V.44      V.42!V.XP&VM.43
*   36.          S.45      T70.04
*   37.          V.46      S.45+FV.44
*   38.          VM.47     V.46,M
*   39.          S.48      VM.47
*   40.          V.52      -FV.YP
*   41.          VM.53     V.YP,M
*   42.          V.54      V.52!V.YP&VM.53
*   43.          V.56      S.45+FV.54
```

```
*  44.          VM.57      V.56,M
*  45.          S.58       VM.57
*  46.          S.59       S.48&S.58
*  47.          VM.60      S.59
*  48.          V.XF       V.XN!V.XP&VM.60
*  49.          V.YF       V.YN!V.YP&VM.60
*  50.          V.70       V.ITERN!V.ITERI&VM.60
*  51. *:OUT    MEM
*  52.         ,A0.40,1    V.70
*  53.          A0.71      A7.00
*  54. *:OUT    MEM
*  55.         ,A0.71,A2.12  V.XF
*  56.          A0.72      A7.00+1
*  57. *:OUT   MEM
*  58.         ,A0.72,A2.12  V.YF
*:SARA  OFF
%00       SET        O'7
%11       SET        O'0
%12       SET        O'2
   .
   .
   .
%34       SET        O'3
YN        SET        O'1
YF        SET        O'0
%72       SET        O'0
* p26a: SARA CAL - CRAY-XMP   1.6bf5  FBCON=1
*                                                    SN  FRT  FIT    PLAIN CAL

     A.%11 A.%00                                     6.   2   661    A0 A7
     A.%12 2                                         7.   1   660    A2 2
     V.XP ,A.%11,A.%12                               8.  17   659    V1 ,A0,A2
     VM V.XP,M                                      34.  36   642    VM V1,M
     V.%42 -FV.XP                                   33.  11   607    V6 -FV1
     V.%44 V.%42!V.XP&VM                            35.   7   572    V4 V6!V1&VM
     A.%13 A.%00+1                                   9.   2   555    A0 A7+1
     V.YP ,A.%13,A.%12                              10.  17   553    V2 ,A0,A2
     VM V.YP,M                                      41.  36   536    VM V2,M
     V.%52 -FV.YP                                   40.  11   501    V5 -FV2
     V.%54 V.%52!V.YP&VM                            42.   7   466    V0 V5!V2&VM
     S.%45 T.%04                                    36.   1   442    S6 T70
     V.%46 S.%45+FV.%44                             37.  11   441    V3 S6+FV4
     VM V.%46,M                                     38.  36   430    VM V3,M
     A.%40 A.%02                                    29.   2   408    A0 A5
     V.ITERI ,A.%40,1                               30.  17   406    V7 ,A0,1
     V.%56 S.%45+FV.%54                             43.  11   404    V6 S6+FV0
     S.%48 VM                                       39.   1   394    S3 VM
     VM V.%56,M                                     44.  36   393    VM V6,M
     S.%41 1                                        31.   1   390    S5 1
     V.ITERN S.%41+V.ITERI                          32.   8   389    V5 S5+V7
     S.%58 VM                                       45.   1   357    S4 VM
     S.%59 S.%48&S.%58                              46.   1   356    S2 S3&S4
     VM S.%59                                       47.   1   355    VM S2
     V.%70 V.ITERN!V.ITERI&VM                       50.   7   354    V4 V5!V7&VM
     ,A.%40,1 V.%70                                 52.  -1   347    ,A0,1 V4
     A.%15 A.%01                                    11.   2   346    A0 A6
```

```
V.RL ,A.%15,A.%12                    12. 17   344    V3 ,A0,A2
V.%22 V.RL*RV.YP                     18. 12   327    V0 V3*RV2
A.%17 A.%01+1                        13.  2   310    A0 A6+1
V.RM ,A.%17,A.%12                    14. 17   308    V6 ,A0,A2
S.%30 1.                             21.  2   292    S1 1.
V.%21 V.RM*RV.YP                     16. 12   291    V7 V6*RV2
V.%31 S.%30-FV.XP                    22. 11   290    V5 S1-FV1
V.%23 V.RM*RV.XP                     19. 12   255    V4 V6*RV1
V.%20 V.RL*RV.XP                     15. 12   219    V6 V3*RV1
V.SUBX1 V.%20-FV.%21                 17. 11   184    V3 V6-FV7
V.%32 V.%31*RV.SUBX1                 23. 12   149    V7 V5*RV3
V.SUBX2 V.%22+FV.%23                 20. 11   148    V6 V0+FV4
V.%33 V.YP*RV.SUBX2                  24. 12   113    V4 V2*RV6
V.XN V.%32+FV.%33                    25. 11   101    V0 V7+FV4
V.%35 V.YP*RV.SUBX1                  27. 12    66    V4 V2*RV3
V.XF V.XN!V.XP&VM                    48.  7    53    V7 V0!V1&VM
A.%71 A.%00                          53.  2    40    A0 A7
,A.%71,A.%12 V.XF                    55. -1    38    ,A0,A2 V7
V.%34 V.%31*RV.SUBX2                 26. 12    30    V3 V5*RV6
V.YN V.%34-FV.%35                    28. 11    18    V1 V3-FV4
V.YF V.YN!V.YP&VM                    49.  7     7    V0 V1!V2&VM
A.%72 A.%00+1                        56.  2     2    A0 A7+1
,A.%72,A.%12 V.YF                    58. -1     0    ,A0,A2 V0
```

For comparison, shown below is the result schedule for the same test case but with `VL` set to 2:

```
* p26a: SARA CAL - CRAY-XMP   1.6bf5  FBCON=1
*                                     SN  FRT  FIT    PLAIN CAL

A.%15 A.%01                          11.  2   163    A0 A6
A.%12 2                               7.  1   162    A2 2
V.RL ,A.%15,A.%12                    12. 17   161    V6 ,A0,A2
A.%11 A.%00                           6.  2   160    A0 A7
V.XP ,A.%11,A.%12                     8. 17   158    V5 ,A0,A2
A.%13 A.%00+1                         9.  2   156    A0 A7+1
V.YP ,A.%13,A.%12                    10. 17   154    V2 ,A0,A2
A.%17 A.%01+1                        13.  2   153    A0 A6+1
V.RM ,A.%17,A.%12                    14. 17   151    V4 ,A0,A2
V.%20 V.RL*RV.XP                     15. 12   140    V3 V6*RV5
V.%21 V.RM*RV.YP                     16. 12   134    V1 V4*RV2
V.%23 V.RM*RV.XP                     19. 12   128    V0 V4*RV5
S.%30 1.                             21.  2   123    S6 1.
V.%22 V.RL*RV.YP                     18. 12   122    V7 V6*RV2
V.%31 S.%30-FV.XP                    22. 11   121    V4 S6-FV5
V.SUBX1 V.%20-FV.%21                 17. 11   115    V6 V3-FV1
V.SUBX2 V.%22+FV.%23                 20. 11   109    V1 V7+FV0
V.%32 V.%31*RV.SUBX1                 23. 12   104    V3 V4*RV6
V.%33 V.YP*RV.SUBX2                  24. 12    98    V0 V2*RV1
V.%34 V.%31*RV.SUBX2                 26. 12    92    V7 V4*RV1
V.%42 -FV.XP                         33. 11    87    V1 -FV5
VM V.XP,M                            34.  6    82    VM V5,M
V.XN V.%32+FV.%33                    25. 11    81    V4 V3+FV0
V.%44 V.%42!V.XP&VM                  35.  7    76    V0 V1!V5&VM
V.%52 -FV.YP                         40. 11    75    V3 -FV2
VM V.YP,M                            41.  6    70    VM V2,M
V.%54 V.%52!V.YP&VM                  42.  7    64    V1 V3!V2&VM
```

```
S.%45 T.%04                          36.  1    60    S5 T70
V.%46 S.%45+FV.%44                   37. 11    59    V3 S5+FV0
V.%56 S.%45+FV.%54                   43. 11    53    V0 S5+FV1
A.%40 A.%02                          29.  2    50    A0 A5
V.ITERI ,A.%40,1                     30. 17    48    V1 ,A0,1
VM V.%46,M                           38.  6    42    VM V3,M
S.%48 VM                             39.  1    36    S2 VM
VM V.%56,M                           44.  6    35    VM V0,M
S.%41 1                              31.  1    32    S4 1
V.ITERN S.%41+V.ITERI                32.  8    31    V3 S4+V1
V.%35 V.YP*RV.SUBX1                  27. 12    30    V0 V2*RV6
S.%58 VM                             45.  1    26    S3 VM
S.%59 S.%48&S.%58                    46.  1    25    S1 S2&S3
VM S.%59                             47.  1    24    VM S1
V.%70 V.ITERN!V.ITERI&VM             50.  7    23    V6 V3!V1&VM
V.YN V.%34-FV.%35                    28. 11    18    V1 V7-FV0
,A.%40,1 V.%70                       52. -1    16    ,A0,1 V6
V.XF V.XN!V.XP&VM                    48.  7    15    V3 V4!V5&VM
A.%71 A.%00                          53.  2    10    A0 A7
,A.%71,A.%12 V.XF                    55. -1     8    ,A0,A2 V3
V.YF V.YN!V.YP&VM                    49.  7     7    V0 V1!V2&VM
A.%72 A.%00+1                        56.  2     2    A0 A7+1
,A.%72,A.%12 V.YF                    58. -1     0    ,A0,A2 V0
```

It is interesting to note that the Mandelbrot set example can be speeded up further by *adding* read statements. Many vector code sequences are limited (as far as SARA optimization is concerned) by the number of vector registers available. Since there are two vectors of values that are read once at the beginning of the Mandelbrot example (`V.VY` and `V.VX`) and then used several times, adding re-reads of these values can have the effect of freeing up vector registers for intermediate results. The speedup in this case from adding the re-reads is about 1.08x for vector length `32`. This is an example of the kind of experimental code tuning that is possible with the aid of SARA, but which would be very tedious otherwise.

5.3 The SARA Coloring Game

One way of checking whether the (latest possible) issue times predicted by SARA make sense is to find the conflict (or other timing rule) that explains why each instruction is scheduled at the clock specified. For example, if two vector instructions both use the floating point adder, they must be scheduled at least `VL+4` apart. If the instruction sequence shows them further apart, then there must be another explanation for their separation, such as a vector register reservation conflict. The SARA coloring game consists of using a variety of colored pens to code the various explanations, and diagramming these effects. The resulting patterns can form not only interesting artistic designs, but can also give a global idea of what resources are in short supply at which points in the execution

sequence. Such analysis frequently leads to ideas for speeding up the sequence still further.

6.0 Conclusions

6.1 Limitations

SARA currently does not recognize data dependences through memory. Hence, results may be incorrect if a value is stored and then read from memory within the same SARA block, since SARA may schedule the read *before* the write!

Another limitation of the current version of SARA (September 1989) is the problem of "runaways". Runaways occur when SARA attempts to evaluate too many overlapping expression trees in parallel. When a runaway occurs, a gap arises in the predicted instruction schedule that is larger than the largest possible interval between instructions for a Cray X-MP. SARA recognizes runaways, but currently does not have the ability to fix the problem automatically because SARA uses a "persistent" scheduling strategy. The SARA programmer must manually force the completion time of one or more trees backward in time from the end (T=0) by assigning a minimum separation in clocks as a `MINSEP` parameter on the appropriate `*:OUT` declarations.

6.2 Results

SARA is written in Fortran for portability. It runs on a wide variety of UNIX and non-UNIX machines, including Crays under CTSS and COS, as well as DEC VAX computers under VMS. The speed at which scheduling and register assignment can be accomplished is somewhat problem dependent, but ranges from 20-30 statements a second on a SUN 3/80 to hundreds of statements a second on a Cray X-MP. The performance of the SARA Optimizing Preprocessor could be greatly improved without affecting its functionality.

6.3 Future Work

Although SARA was originally designed to function as a CAL programmer's assistant, we are also investigating the possibility of using the technology to improve the performance of existing production Cray higher level language compilers. In its original mode, in addition to its use for squeezing that last bit of performance out of computationally intensive kernels, it could also serve as a

library maintenance and portability aid for Cray-like architectures, since optimum schedules for mathematical library routines are often invalidated by different instruction timings in newer machines, even if the instruction set is upward-compatible.

We plan to enhance SARA in the near future to include the ability to accept raw CAL as an input language (by generating appropriate internal pseudoregister identifiers), to add the ability to breakup runaways automatically, and to support other Cray X-MP-like architectures. SARA currently supports both the X-MP and the Scientific Computer Systems SCS-40. We are currently adding support for the Cray Y-MP and may also add support for the Cray 2.

7.0 References

[1] S. Arya, "An Optimal Instruction-Scheduling Model for a Class of Vector Processors", IEEE Trans. Computers, Vol. C-34, No. 11, Nov. 1985, pp. 981-995.

[2] D. Bernstein, H. Boral, and R. Y. Pinter, "Optimal Chaining in Expression Trees (Preliminary Version)", SIGPLAN Notices, Vol. 21, No. 7, July 1986, pp. 1-10.

[3] J. Tang and E. S. Davidson, "An Evaluation of Cray-1 and Cray X-MP Performance on Vectorizable Livermore Fortran Kernels," in Proc. International Conference on SuperComputing, St. Malo, France, July 1988, pp. 510-518.

[4] S. Weiss and J. E. Smith, "A Study of Scalar Compilation Techniques for Pipelined Supercomputers," SIGPLAN Notices, Vol. 22, No. 10, Oct. 1987, pp. 105-109.

[5] E. A. Williams, "Measurement of Two Scientific Workloads Using the CRAY X-MP Performance Monitor", Technical Report SRC-TR-88-020, Supercomputing Research Center, Lanham, MD., Nov. 1988.