YET ANOTHER LOOK AT CODE GENERATION FOR PASCAL ON CDC 6000 & CYBER MACHINES

by

LAWRENCE A. LIDDIARD

DECEMBER, 1976

UNIVERSITY COMPUTER CENTER
TECHNICAL REPORT UCC 76002

UNIVERSITY COMPUTER CENTER
UNIVERSITY OF MINNESOTA
MINNEAPOLIS, MINNESOTA 55455

YET ANOTHER LOOK AT CODE GENERATION FOR PASCAL ON CDC 6000 AND CYBER MACHINES

BY

LAWRENCE A. LIDDIARD
ASSOCIATE DIRECTOR/SYSTEMS AND OPERATIONS
UNIVERSITY COMPUTER CENTER

DECEMBER, 1976

In the 1974-1975 school year the Computer Science Department at the University of Minnesota decided that more extensive use of PASCAL would be made in their programming language courses. Since much of this use would be interactive, one concern of the University Computer Center was that PASCAL required the most memory for compilation (approx. 55000_8) of the interactive languages available (for comparison note that BASIC and APL*CYBER require approximately 25000_8 and MNF FORTRAN 44000_8) on our Instructional Time Sharing CDC 6400. Several years experience in running a large volume (202 maximum simultaneous users and 350,000 runs in April 1975 see reference [5]) time sharing service had shown that a successful time sharing service on CDC 6000 machines required that each user be limited to at most 54000_8 words of memory. (In addition to a restriction on memory for each user there are requirements for enough mass storage channels and peripheral processors . . . but that is another story.) Since the PASCAL level 9 compiler required 42121_8 to load and in addition allocated buffers, "stack" and "heap" space, I decided that a reduction in PASCAL load size could be accomplished by rethinking certain PASCAL code generations in the area of procedure calls, constant loads, case statement jumps and/or by combining common procedures (+, , divide, mod, in-line functions). In addition when Urs Ammann of ETH, Zurich was informed of the project, he suggested several core reduction ideas that were implemented by John Strait of our staff in late 1975. In a compiler-compiler the generation of faster and shorter code helps not only the user, but also since PASCAL 6000 compiles itself this will be reflected in a smaller and faster compiler.

In trying to analyze where core reductions can be made in a compiler such as PASCAL, it is worthwhile obtaining several tables that help the

core reducer to concentrate on the essentials of the language (or program) that are amenable to reduction techniques. The first such table is the count of the static number of procedure calls ordered in descending use.

In PASCAL 6000 with a load length of approximately 18,000-60 bit words (43130_8) there are approximately 2300 static (meaning physically present) procedure calls distributed among the approximately 140 procedures that constitute the compiler. The top 10% of the called procedures are listed in the following table with an additional break down into three main areas: code generation, symbol input and error message.

T A B L E 1

PROCEDURE NAME	STATIC COUNT
ERROR	387
GEN15	338
INSYMBOL	214
GEN30	154
COMPTYPES	136
DECREFX	124
NEEDX	116
LOAD	77
SKIP	53
NOOP	45
CLEARREGS	40
EXPRESSION	37
NEXTCH	36
OPERATION	25
TOTAL	1782 = 77% of 2300 calls

	CODE GENERATION			SYMBOL INPUT		ERROR MESSAGE	
NAME	COUNT	WORDS /CALL	CORE REQUIRED	NAME	COUNT	NAME	COUNT
GEN15	338	* 4 =	1352	INSYMBOL	214	ERROR	387
GEN30	154	* 4 =	616	SKIP	53		
COMPTYPES	136	* 4.5 =	612	NEXTCH	36		
DECREFX	124	* 3 =	372				
NEEDX	116	* 3.5 =	406				
LOAD	77	* 3 =	231				
NOOP	45	* 1 =	45				
CLEARREGS	40	* 1 =	40				
OPERATION	25	* 3.5 =	87	* 1 word/call =		*1½ word/call=	
TOTALS	1055		3761		303		580
% of 2300 calls:	46%			13%		17%	
% of load core:			21%		2%		3%

Except for EXPRESSION these procedures fall into the three groups mentioned above and account for 76% of the static procedure calls and 25.5 per cent of the total loaded length of PASCAL 6000. Thus, a large savings can be obtained for each word saved in a static procedure call (2300 * n words saved).

Another place to look for central memory reductions is in the prologue and exit of each procedure. Since there are 140 procedures in the PASCAL 6000, each word saved represents 140 * n (214_8) additional cells saved. Candidates for this reduction are the procedure name word, the word containing the lengths of the executable and total procedure code, the three words for procedure initialization, and the 1 1/2 words for procedure exit. The first two are needed for the current POST MORTEM dump, but could be eliminated if the POST MORTEM dump would obtain these words from the LGO file or would build a specific POST MORTEM file at load time for possible use when errors occurred. The procedure initialization and exit can be reduced to one word and three quarters respectively by use of common entry and exit routines at a probable 8% slow down in compilation speed due to the slowness of NEXTCH (actually 25% of compilation time on a CYBER 74 is currently spent in NEXTCH, and by improving this routine the 8% compilation slow down can be more than compensated for by the NEXTCH speed up).

One final way to look for central memory reduction in the compiler is to look at a table of the largest PASCAL 6000 procedures. This uses the theory that it always pays to look at the fattest routines. When looking at this table the core reducer should look for similar functionality that can be combined into one routine or broken out into a simple subroutine. In addition the longest routines are examined for exactly what makes them long, with a look to improving code production that can reduce the length (in PASCAL it may be the nesting depth of procedure calls from that routine, a poorly done CASE statement, or a rethinking of code generation). Finally, the reverse of top down step wise refinement (integration) can sometimes achieve good savings as was obtained by combining the procedures ROUNDF, ABSF, SQRF, TRUNFC, ODDP, ORDF, CHRF, PREDSUCCF, CARDF, EXPOF, into the routine STDINLINEFUNCS.

TABLE 2

PASCAL 6000 PROCEDURES ORDERED BY LENGTH IN PASCAL2

COMPILER PROCEDURE NAME	LENGTH		
	PASCAL 2	PASCAL 2 MODIFIED	
CALLNONSTANDARD	1266_8	1034_8	
STORE	1244_8	1153_8	
LOAD	1200_8	1146_8	
TERM	1027_8	516_8	⎫ benefits from
EXPRESSION	1025_8	634_8	⎪ depth of nesting
FACTOR	1006_8	533_8	⎪ change by not
BODY	754_8	674_8	⎪ loading static
TYP	712_8	605_8	⎭ link
INSYMBOL	655_8	566_8	
WRITE	603_8	436_8	
UNPACK	535_8	411_8	
FORSTATEMENT	523_8	415_8	
FIELDLIST	515_8	414_8	
INDEXCODE	504_8	414_8	
CASESTATEMENT	503_8	426_8	
PACK	473_8	352_8	
PARAMETERLIST	472_8	375_8	
SIMPLEEXPRESSION	461_8	220_8	("+" and "-" code combined)
PROCEDUREDECLARATION	427_8	404_8	
STDINLINEFUNCS [as separate routines]	[1350_8]	352_8	

Not all of the code reduction features were viable or even desirable; but since the application of each reduction was applied to the previous one, the following table briefly describes the change, the octal length at the PASCAL compiler after the change and the octal (decimal) savings

CHANGE		PASCAL OCTAL LOAD LENGTH	SAVINGS
0	stock level 9 compiler	43121	
1	avoid extra SBi Xj commands when loading a base address	42703	216_8(142)
2	avoid extra Bxi Xj commands at procedure calls	42666	15_8(13)
3	use 2 jump commands/word in CASE statements	42504	162_8(114)
4	correct inefficient case statement in the procedure "STATEMENT"	42462	22_8(18)
5	use common ENTRY/EXIT routines	41577	663_8(435)
6	make the "RJTOEXT" procedure reasonable	41435	142_8(98)
7	use a single procedure for the in line functions (ODD..CARD) called STDINLINEFUNCS	40600	635_8(413)
8	eliminate extra stack manipulation	40441	137_8(95)
9	use MXi mask and LXi shift for constants	40410	31_8(25)
10	eliminate jumps after procedures that end a CASE or THEN	40017	371_8(249)
11	use common code for "+" and "-"	37707	110_8(72)
12	use X5 = MX5 number-1 of static indirects to load if static link not available	35541	2146_8(1126)
13	use RJ procedure rather than SX7 return, JP procedure	34162	1357_8(751)
	TOTAL		6737_8(3551)

Examples of the code generated in the application of these principles are given in the following pages for changes 3, 5, 9, 10, 12 and 13. Note also that the application of change 13 disallows 10.

Change 3

Example of the CASE statement

From procedure OPTIONS

```
    PASCAL                          CODE GENERATION
16  CASE  CHI  OF       .16    SA1   CHI                SB3   x1
20  'B'                 .17    JP    B3+.113
27  'E'             'B'.20     (* code for case 'B' *)
    'L'             'E'.27     (* code for case 'E', etc. *)
    'P'                .113
    'T'                .114    JP    .144       (*'A'*)
    'U'                .115    JP    .20        (*'B'*)
    'X'                .116    JP    .144       (*'C'*)
```

```
     PASCAL                          CODE GENERATION  (continued)
     END                  .117   JP    .144       (*'D'*)
144  IF                   .120   JP    .27        (*'E'*)
                          .
                          .
                          .
                          .142   JP    .144       (*'W'*)
                          .143   JP    .106       (*'X'*)
                          .144
```

If 2 jumps per word are used

```
13   CASE                   .13    SA1   CH1  SX1 X1-2
15   'B'                    .14    LX1   59   SB3 X1     JP B3+.112
24   'E'              'B'.15       (*etc.*)
      .                   .111    JP    .126       (*'A'*)
      .                   .112    PL X1,.15        (*'B'*)  JP .126 (*'C'*)
      .                   .113    PL X1,.126       (*'D'*)  JP .24  (*'E'*)
     END                  .114    PL X1,.126       (*'F'*)  JP .126 (*'G'*)
126  IF                    .
                           .
                           .
                           .124    PL X1,.126       (*'V'*)  JP .126 (*'W'*)
                           .125    PL X1,.103       (*'X'*)  NO       NO
```

Saves: 11 cells

Costs: loss of rememberance of CH1 in X1 and slower execution speed.

single jump/word 48 cycles for SA1 CH1; SB3 X1; NO; JP B3 + .113; JP .20

vs 61 cycles upper SA1 CH1; SX1 X1-2; LX1 59; SB3 X1; JP B3 + .112; PL X1,.15

vs 66 cycles lower jump taken.

(Note the SX1 X1-2 is not needed if only positive case labels are allowed).

Change 5

The current entry code of

```
                SA6 B6                  (save static link) in stack
                SX6 B5                  save old STACK BASE
                LX6 18
                BX7 X7+X6               + return address in STACK + 1
                SB5 B6                  set new STACK BASE
 3 1/2 words    SA7 B5+B1
                SA6 B6+needed           set B6 to NEXT
                SB7 B6+100
                SA0 5                   set return address
                GE  B7,B4,HEAPCOLLISION
```

```
  is changed to 1 word
                SB7 needed
                RJ  P.ENTRB             (see code for 12 and 13)

the current exit code of
                SA1 B5+B1
                SB6 B5                  reset NEXT
  1 3/4 words   SB7 X1                  set return address
                LX1 42
                SB5 X1                  restore old STACK BASE
                JP  B7
  is changed to 3/4 word
                SA1 B5+B1
                JP  P.EXIT
  where P.EXIT  SB7 X1
                LX1 42
                SB6 B5
                SB5 X1
                JP  B7
```

For a CYBER 74 the change in P.EXIT ratio in cycle speed is 33/26 = 1.27; for the 6400 the ratio is 63/51 = 1.24.

Change 9

Examples of the MXi and LXi commands for constants.

In PASCAL the very useful powerset often uses constants that have contiguous bit sequences. For the CDC 6000 machines it is usually best to have code productions that are two 15-bit commands rather than a single 30-bit command, since the 30-bit command often will not fit into the current word causing a non-useful NO command to be produced. The MASK and SHIFT commands of the CDC 6000 allow any constant of the form $(2^n-1)*2^m$ to be generated by MXi n; LXi m+n. If the constant value is greater than 2^{17} this is always the best method since it takes (on a 6400) the same or less time and saves the 60-bit word holding that constant. If the constant is less than 2^{17} and there is only 15-bits left in the current word, this method will avoid the useless NO command.

E X A M P L E S

a) PROCEDURE WITHSTATEMENT (COMP 6147, 6148)

```
IF CBDFSPL <> 0 THEN                    CURRENT PASCAL
   BEGIN NEEDX(0,7,1)
                                        ZR  X7,.143  BX0 X6  NO
                                        SX1 7  SX2 B5+20
                                        SX6 B7  SX7.120   NO
                                        EQ  NEEDX
```

```
                                        MODIFIED PASCAL
                                        ZR  X7,.127  BX0 X6  MX1 3
                                        LX1  3  SX2 B5+20  MX5 1
                                        RJ  NEEDX
```

b) PROCEDURE OPTIONS (COMP 407)

```
   'B' IF CH IN ['1'..'9']          CURRENT  SA1 B2+474  SA2 162
                                    MODIFIED SA1 B2+474  MX2 9  LX2 37
```

c) PROCEDURE GEN30 (COMP 2070)

```
   CBUF = CBUF*1000B*1000B+777777B+FK   CURRENT  BX5 X6  LX5 9  LX5 9  NO
                                                 SA1 217
   CBUF = CBUF*1000000B+777777B+FK      MODIFIED BX5 X6  LX5 18  MX1 18  LX1 18
```

Change 10

Example from Procedure LOAD (COMP 2792-2796)

```
     PASCAL                                CODE GENERATION
    IF SVAL = 0 then GEN15(13B,1,1,1)    .120 IX4 X2-X3  NZ X4,.125  NO
    ELSE                                 .121 SX0 13B  BX2 X1  BX3 X1
125 IF SVAL = 1 then GEN15(76B,1,1,0)    .122 SX6 B7   SX7 .124
    ELSE                                 .123 JP GEN15
131 IF SVAL = 2 then GEN15(76B,1,1,1)    .124 JP .141
                                         .125 IX4  X2-X0  NZ X4,.131  NO
                                         .126 SX0  76B  SX2 B1  SX6 B7
                                         .127 SX7 .130  JP GEN15
                                         .130 EQ .141
                                         .131 SX4 B1+B1  IX5 X2-X4  NZ X5, .136
                                         .132 SX0 76B  SX2 B1  BX3 X2
                                         .133 SX6 B7  SX7 .135
                                         .134 JP  GEN15
                                         .135 JP  .141
                                         .136
```

Rule 1

If the termination of a THEN clause is a procedure call and there is no other clause ending at the same point before an ELSE; more efficient code can be generated by eliminating the standard JP to terminal IF point by making the return address of that procedure call go to the terminal IF point. The above example code would produce the following, saving three words and the corresponding jump times:

```
.120 IX4 X2-X3  NZ X4,.124  NO
.121 SX0 13B  BX2 X1  BX3 X1
.122 SX6 B7  SX7 .141
.123 JP GEN15
.124 IX4 X2-X0  NZ X4,.127  NO
.125 SX0 76B  SX2 B1  SX6 B7
.126 SX7 .141  JP GEN15
.127 SX4 B1+B1  IX5 X2-X4  NZ X5,.133
.130 SX0 76B  SX2 B1  BX3 X2
.131 SX6 B7  SX7 .141
.132 JP GEN15
.133
```

Rule 2

If the termination of an individual CASE is a procedure call and there is no other clause ending at the same point (i.e. more than one address that refers to such a point) then the JP to the terminal CASE point can be eliminated by making the return address and that procedure call go the terminal case point. The coding produced is similar to that in the THEN-ELSE.

Changes 12 and 13

Example from the procedure FACTOR.

Note that this modification depended on Change 5, the use of a common ENTRY point routine.

Previous CODE GENERATED FOR DECREFX(I) and FACTOR(FSYS):

	DECREFX(I)	COMP 5200		FACTOR(FSYS)	COMP 5115
	SA1 B5+12 BX0 X1	] I			SA1 B5+3 BX0 X1
3 ½ words	SA2 B5 SA2 X2 SA2 X2 SA2 X2 BX6 X2	] follow static link		2 ½ words	SA2 B5 BX6 X5 SX7 *+2 EQ FACTOR
	SX7 *+2 EQ DECREFX	] set return address] transfer to routine			

Using changes 12 and 13:

	SA1 B5+12 BX0 X1	] I			SA1 B5+3 BX0 X1
1 ½ words	MX5 4 RJ DECREFX	] if base not in B register] transfer to routine		1 ½ words	SA5 B5 RJ FACTOR	] base in B register

P.ENTRB

is an example of the common ENTRY routine where the static link is stored on the stack and a RJ ROUTINE is used rather than the SX7 RETURN JP ROUTINE.

X5 = STATIC LINK (IF X5 > 0) else MX5 LEVEL-PFLEV-1

where LEVEL and PFLEV are the LEVELS of the caller and called procedure respectively.

```
P.ENTRB  DATA  0
         PL  X5, P.ENT2
         BX6 X5
         SA5 B5
  +      SA5 X5                 ]
         LX6 1                  ]  static link to X5
         NG  X6,*               ]

P.ENT2   BX6 X5                 ]  store static
         SA6 B6                 ]  link on stack

         SA5 P.ENTRB            ]
         AX5 30                 ]  obtain
         SA5 X5-2               ]  RETURN
         AX5 30                 ]  ADDRESS
         SX7 X5                 ]

         BX6 B5                 ]  store B5 and
         LX6 18                 ]  RETURN ADDRESS
         BX7 X7+X6              ]  on stack

         SB5 B6                 ]  set B5
         SA7 B6+B1              ]  to BASE
         SB6 B6+B7              ]  set B6 to NEXT
         SB7 B6+100             ]  exit if no STACK
         LT  B7,B4,P.ENTRB      ]  HEAPCOLLISION
```

This change was the most controversial since although it gave dramatic reduction in central memory, it changed the manner of procedure calls from that documented in reference [3] and definitely slowed down compilation by 8% on the CYBER 74. One way to use the core reduction and slower procedure linkage would be to have the PASCAL compiler internally use the RETURN JUMP method of procedure calls while generating the forms SXi return and JP procedure for user execution binary. This would mean that three assemblies of PASCAL would be required to update to a new PASCAL rather than the two assemblies required presently.

Comparisons

Some comparisons with PASCAL 1 described in [1] are interesting.

(1) by instruction length

1971 PASCAL 1			PASCAL 2 Modified to Save Core		
15,925	48.7%	long instructions (30-bit)	12,959	32.7%	
9,385	28.7%	short instructions (15-bit)	19,676	49.6%	
7,456	22.8%	padding instruction (NOOP)	7,010	17.7%	executable 2564 (6.5%)
32,766	100%	= $12,173_{10}$ words	39,645	100%	= $13,151_{10}$ words

(2) by instruction type

27.6%	fetch and store	8,536	21.5%	
15.0%	load literal	4,085	10.3%	
3.5%	arithmetic	1,810	4.6%	
14.4%	logical and shift and mask	12,523	31.6%	
6.2%	base address register	776	2.0%	
10.5%	jump and subroutine calls	4,905	12.4%	
22.8%	(NOOPS)	7,010	17.7%	
32,766	100.0%	39,645	100.1%	

Reference [3] shows that register remembrances, BXi X6,7 rather than NOOP's and more efficient code generation were one aim in the design of PASCAL2. This manifests in PASCAL2 in increased subroutine calls (from 10.5% to 12.4%) to produce the more efficient code and in the reduction of fetch and store commands (from 27.6% to 21.5%). In addition to [3]'s replacement of NOOP's with logical commands the generation of MX5 level; RJ PROCEDURE compared with PASCAL's SX7 RETURN EQ PROCEDURE caused increased mask commands and a corresponding decrease in LOAD LITERALS (SXi value).

Note that in PASCAL1 15.3% of the loaded space (NOOP commands) is not used compared with 13.3% in the modified PASCAL. The designer of the CDC 6000 machines has recognized this loss in his latest machine the CRAY 1. In that machine JUMPS are to any 16-bit portion of the 64-bit word rather than to the top most portion of the word as is done in a CDC 6000 instruction. Thus the 4,446 non executable NOOPS (of 7010 total) could be eliminated if the CDC 6000 machines had such a feature allowing 8.5% of the loaded compile space to be saved.

Our last way to shorten the compiler is to rewrite the code generation part which currently comprises about one-fourth of the total length. The current code generation scheme seems to have two main deficiencies. The target computer code that PASCAL generates is spread throughout rather than gathered in functional groupings in the compiler. Second, for

"simple ability" machine instructions of micro and CDC 6000 computers, a procedure call for each command of the numerous code generation routines insures that the compiler will be fairly lengthy. In order to avoid this excess length a macro skeleton (i.e. simple) language is designed that a very short macro interpreter can expand to required computer commands, register allocations, decisions and code generation operations.

As an example consider the PASCAL code

```
357(8)  BEGIN NEEDX(0,7,K);  GEN15(10B,K,I,0)          COMP 5324
        DECREFX(I);  GEN15(21B,K,0,LREC.EXP)           COMP 5325
        NEEDX(0,7,I);  GEN15(13B,I,I,I);               COMP 5326
        GEN15(36B,K,K,I);  DECREFX(I)                  COMP 5327
414(8)  END                                            COMP 5328
        (414(8) - 357(8) = 35(8) = 29 cells)
```

rewritten in a macro skeleton similar to that used by E. J. Mundstock when we wrote the MNF compiler. In those skeletons the basic 15-bit command of the CDC 6000 was broken down into its four component parts of 6-bit operation code and 3-bit I, J and K register fields. To each of these fields an additional bit was added to signify a relative register or pseudo command if set to 1, else an absolute register or command:

MACRO Language Example for the previous PASCAL code

```
P:=LREC.EXP
MACRO(MOD1)
```

			pseudo command	command	pseudo register	register I	pseudo register	register J	pseudo register	register K
			1	7	1	3	1	3	1	3 = 20 bits
where MOD1	NEEDX(0,7,K)		1	1	0	0	0	7	1	2
	GEN15(10B,K,I,0)	1st word	0	10	1	2	1	0	0	0
	DECREFX(I)		1	0	1	0	0	0	0	0
	GEN15(21B,K,0,P)		0	21	1	2	0	0	1	7
	NEEDX(0,7,I)	2nd word	1	1	0	0	0	7	1	0
	GEN15(13B,I,I,I)		0	13	1	0	1	0	1	0
	GEN15(36B,K,K,I)		0	36	1	2	1	2	1	0
	DECREFX(I)	3rd word	1	0	1	0	0	0	0	0
	ENDMACRO		1	2	0	0	0	0	0	0

pseudo commands: DECREFX=0; NEEDX=1; ENDMACRO=2; IFDEBUG=3; ENDIF=4; LOAD=5
pseudo register of ATTR: I=0; J=1; K=2; L=3; M=4; N=5; O=6; P=7

Thus approximately 29 cells are replaced by 6 cells which compares favorably with that reduction obtained in the MNF compiler where 1900 pseudo commands occupied 709 words and required a MACRO interpreter of 350 words. If the pseudo commands of MNF were done as normal procedure calls it would have taken approximately 1900 * 2 3/4 = 5225 words in the MNF compiler. Thus the total saving is 4000 words or an 80% reduction in that code generation portion of the MNF compiler. Applying this to the PASCAL compiler with approximately 4000 words for 1055 code generation procedures calls and assuming a MACRO interpreter of 500 words and 100 calls to the MACRO interpreter we can estimate a $2500\text{-}3000_{10}$ word reduction in the loaded length of the PASCAL compiler.

Note that the implementation of this assumes a VALUE declaration initialization of packed records which is currently not available in PASCAL2. The actual implementation of the MACRO skeleton is not as simple as my example but reference [4] or the MNF compiler listing give additional pseudo commands and implementation techniques.

Conclusions

PASCAL 2 is amenable to several different methods of compiler length reduction. As a fellow compiler writer (although since MNF is written in machine language it may be compared with the last of the dinosaurs speaking to Homo sapiens), I would rather see the full language specification and one standard compiler, than to see small subsets such as PASCAL-S. For this reason I think it essential to improve PASCAL2 and with the reductions discussed in this article it should be possible to obtain load lengths of approximately 30_8k for the full language rather than the current 44_8k on a CDC 6000 (i.e. a reduction by one-third).

References

1. N. Wirth, "The Design of a PASCAL Compiler," Software Practice and Experience, Oct-Dec 1971.
2. N. Wirth, "On 'PASCAL' Code Generation and the CDC 6000 Computer," Stanford, February 1972.
3. Urs Ammann, "On Code Generation in a PASCAL Compiler," ETH, April 1976.
4. David Gries, Computer Construction for Digital Computers, "17.5 Computing Code Generation," John Wiley & Sons, Inc., 1971, pages 363-366.
5. M. M. Skow, "MERITSS 1971-1975," University Computer Center, University of Minnesota, Technical Report 75003, October 1975.