Supporting the Specification and Analysis of Timing Constraints

by

Lo Ko, Christopher Healy, Emily Ratliff,
Robert Arnold, David Whalley
Florida State University

and

Marion Harmon
Florida A&M University

The Problem

- High-level timing analysis allows a user to relate timing constraints to source code portions.
- Low-level timing analysis on machine code is much more accurate.
- How can a user specify timing constraints at the source code level and obtain timing predictions associated with the more accurate low-level analysis?

Goals

- A user should be able to quickly specify constraints and obtain timing predictions for the specified portions of a program.
- The user should only be allowed to select portions of the program for which timing bounds can be obtained.
- The ability to specify constraints and obtain timing predictions should not inhibit compiler optimizations from being performed.
- The correspondence between source code and machine code of the program selected by the user for timing prediction should be graphically depicted.

Main Window at Function Level

time.bin

Select a function within the program.

function name

des
getbit
ks
cyfun
main

Cycles to Execute the ks Function
Best Case 3588 Worst Case 5633

Exit | Constraints | More Detail | Back

Source Code Window

C Source Code of des.c

line # source code

```
23          49,17,57,25};
24       static great kns[17];
25       static int initflag=1;
26       int ii,i,j,k;
27       unsigned long ic,shifter,getbit();
28       immense itmp;
29       void cyfun(), ks();
30
31       if (initflag) {
32          initflag=0;
33          bit[1]=shifter=1L;
34          for(j=2;j<=32;j++) bit[j] = (shifter <<= 1);
35          }
36       if (*newkey) {
37          *newkey=0;
38          for(i=1;i<=16;i++) ks(key, i, &kns[i]);
39          }
40       itmp.r=itmp.l=0L;
41       for (j=32,k=64;j>=1;j--,k--) {
42          itmp.r = (itmp.r <<= 1) | getbit(inp,ip[j],32);
43          itmp.l = (itmp.l <<= 1) | getbit(inp,ip[k],32);
44          }
45       for (i=1;i<=16;i++) {
46          ii = (isw == 1 ? 17-i : i);
47          cyfun(itmp.l, kns[ii], &ic);
48          ic ^= itmp.r;
49          itmp.r=itmp.l;
50          itmp.l=ic;
51          }
52       ic=itmp.r;
53       itmp.r=itmp.l;
54       itmp.l=ic;
55       (*out).r=(*out).l=0L;
56       for (j=32,k=64; j >= 1; j--, k--) {
57          (*out).r = ((*out).r <<= 1) | getbit(itmp,ipm[j],32);
58          (*out).l = ((*out).l <<= 1) | getbit(itmp,ipm[k],32);
59          }
60    }
```

Select Path | Accept | Cancel | Clear All

Assembly Code Window

Methods for Selecting Code Portions

- Three ways to select code portions for timing predictions.
  - — Constraints Window Selection: The user can quickly access the portions of the program specified in the source code timing constraints.
  - — Main Window Selection: The user can make very fine-grain level requests.
  - — Source Code Window Selection: The user can make requests very quickly.

Main Window at Loop Level

time.bin

Select a loop within the function des.

loop name	source lines	nest level
entire function	31..58	0
LOOP 1	34..34	1
LOOP 2	38..38	1
LOOP 3	41..43	1
LOOP 4	45..50	1
LOOP 5	56..58	1

Cycles to Execute the des Function
Best Case 22084 Worst Case 257867

Exit | Constraints | More Detail | Back

Main Window at Path Level

time.bin

Select a path within the function des.

path	blocks	source lines	
entire function		31..58	
path 1			
	1	31..31	
	5	36..36	
	10	40..41	
	17	45..45	
	18..23	45..50	loop 4
	24	52..56	
	31	56..56	
path 2			
	1	31..31	

Cycles to Execute Path 1 within Function des
Best Case 22084 Worst Case 58873

Exit | Constraints | More Detail | Back

Main Window at Subpath Level

time.bin

Select a subpath within path 1
within the function des.

blocks	source lines	
1	31..31	
5	36..36	
10	40..41	
17	45..45	
18..23	45..50	loop 4
24	52..56	
31	56..56	

Cycles to Execute Subpath from Block 5 To
Block 17 Best Case 56 Worst Case 82

Exit | Constraints | More Detail | Back

Main Window at Assembly Level

time.bin

WARNING: Highlighted source lines may not match selected instructions
Click and drag to select instructions.

block	instructions

```
# block 5 (lines 36-36)
            L219:
                ld    [%o5],%o0
                cmp   %o0,%g0
                be,a  L224
                st    %g0,[%sp + .1_itmp]
-----------------------------------------------
# block 17 (lines 45-45)
            L230:
                mov   1,%l0
                sethi   %hi(L214),%i0
                add   %sp,.3_STARG,%i1
                add   %sp,.1_ic,%l4
```

Cycles to Execute from Inst 69 To Inst 151 Best Case 41 Worst Case 58

Exit | Constraints | More Detail | Back

Selecting a Path via the Source Code Window

C Source Code of des.c

line # source code

```
15          32,24,16,8,57,49,41,33,25,17,9,1,59,51,43,35,
16          27,19,11,3,61,53,45,37,29,21,13,5,63,55,47,39,
17          31,23,15,7};
18       static char ipm[65]=
19          {0,40,8,48,16,56,24,64,32,39,7,47,15,
20          55,23,63,31,38,6,46,14,54,22,62,30,37,5,45,13,
21          53,21,61,29,36,4,44,12,52,20,60,28,35,3,43,11,
22          51,19,59,27,34,2,42,10,50,18,58,26,33,1,41,9,
23          49,17,57,25};
24       static great kns[17];
25       static int initflag=1;
26       int ii,i,j,k;
27       unsigned long ic,shifter,getbit();
28       immense itmp;
29       void cyfun(), ks();
30
31       if (initflag) {
32          initflag=0;
33          bit[1]=shifter=1L;
34          for(j=2;j<=32;j++) bit[j] = (shifter <<= 1);
35          }
36       if (*newkey) {
37          *newkey=0;
38          for(i=1;i<=16;i++) ks(key, i, &kns[i]);
39          }
40       itmp.r=itmp.l=0L;
41       for (j=32,k=64;j>=1;j--,k--) {
42          itmp.r = (itmp.r <<= 1) | getbit(inp,ip[j],32);
43          itmp.l = (itmp.l <<= 1) | getbit(inp,ip[k],32);
44          }
45       for (i=1;i<=16;i++) {
46          ii = (isw == 1 ? 17-i : i);
47          cyfun(itmp.l, kns[ii], &ic);
48          ic ^= itmp.r;
49          itmp.r=itmp.l;
50          itmp.l=ic;
51          }
52       ic=itmp.r;
```

Select Path | Accept | Cancel | Clear All

Best Case Path

C Source Code of des.c

line # source code

```
18      static char ipm[65]=
19         {0,40,8,48,16,56,24,64,32,39,7,47,15,
20         55,23,63,31,38,6,46,14,54,22,62,30,37,5,45,13,
21         53,21,61,29,36,4,44,12,52,20,60,28,35,3,43,11,
22         51,19,59,27,34,2,42,10,50,18,58,26,33,1,41,9,
23         49,17,57,25};
24      static great kns[17];
25      static int initflag=1;
26      int ii,i,j,k;
27      unsigned long ic,shifter,getbit();
28      immense itmp;
29      void cyfun(), ks();
30
31      if (initflag) {
32         initflag=0;
33         bit[1]=shifter=1L;
34         for(j=2;j<=32;j++) bit[j] = (shifter <<= 1);
35         }
36      if (*newkey) {
37         *newkey=0;
38         for(i=1;i<=16;i++) ks(key, i, &kns[i]);
39         }
40      itmp.r=itmp.l=0L;
41      for (j=32,k=64;j>=1;j--,k--) {
42         itmp.r = (itmp.r <<= 1) | getbit(inp,ip[j],32);
43         itmp.l = (itmp.l <<= 1) | getbit(inp,ip[k],32);
44         }
45      for (i=1;i<=16;i++) {
46         ii = (isw == 1 ? 17-i : i);
47         cyfun(itmp.l, kns[ii], &ic);
48         ic ^= itmp.r;
49         itmp.r=itmp.l;
50         itmp.l=ic;
51         }
52      ic=itmp.r;
53      itmp.r=itmp.l;
54      itmp.l=ic;
55      (*out).r=(*out).l=0L;
```

Select Path | Accept | Cancel | Clear All

Worst Case Path

C Source Code of des.c

line # source code

```
18      static char ipm[65]=
19         {0,40,8,48,16,56,24,64,32,39,7,47,15,
20         55,23,63,31,38,6,46,14,54,22,62,30,37,5,45,13,
21         53,21,61,29,36,4,44,12,52,20,60,28,35,3,43,11,
22         51,19,59,27,34,2,42,10,50,18,58,26,33,1,41,9,
23         49,17,57,25};
24      static great kns[17];
25      static int initflag=1;
26      int ii,i,j,k;
27      unsigned long ic,shifter,getbit();
28      immense itmp;
29      void cyfun(), ks();
30
31      if (initflag) {
32         initflag=0;
33         bit[1]=shifter=1L;
34         for(j=2;j<=32;j++) bit[j] = (shifter <<= 1);
35         }
36      if (*newkey) {
37         *newkey=0;
38         for(i=1;i<=16;i++) ks(key, i, &kns[i]);
39         }
40      itmp.r=itmp.l=0L;
41      for (j=32,k=64;j>=1;j--,k--) {
42         itmp.r = (itmp.r <<= 1) | getbit(inp,ip[j],32);
43         itmp.l = (itmp.l <<= 1) | getbit(inp,ip[k],32);
44         }
45      for (i=1;i<=16;i++) {
46         ii = (isw == 1 ? 17-i : i);
47         cyfun(itmp.l, kns[ii], &ic);
48         ic ^= itmp.r;
49         itmp.r=itmp.l;
50         itmp.l=ic;
51         }
52      ic=itmp.r;
53      itmp.r=itmp.l;
54      itmp.l=ic;
55      (*out).r=(*out).l=0L;
```

Select Path | Accept | Cancel | Clear All

Displaying Pipeline Diagrams

Best Case Pipeline Diagram

cycle #	IF	ID	EX	FPEX	CA	WB	FWB
1:	144						
2:	145	144					
3:	146	145	144				
4:	147	146	145		144		
5:	148	147	146		145	144	
6:	148	147	146		145		
7:	148	147			146		
8:	149	148	147			146	
9:	150	149	148		147		
10:	150	149			148	147	
11:	151	150	149			148	
12:	153	151			149		
13:	154	153	151			149	
14:	155	154	153		151		
15:	156	155	154		153	151	
16:	157	156	155		154	153	
17:	157	156	155		154		
18:	158	157	156		155		
19:		158			156	155	
20:			158			156	
21:					158		
22:						158	

Dismiss

Worst Case Pipeline Diagram

cycle #	IF	ID	EX	FPEX	CA	WB	FWB
1:	144						
2:	145	144					
3:	146	145	144				
4:	146		145		144		
5:	146				145	144	
6:	146				145		
7:	146						
8:	146						
9:	146						
10:	146						
11:	146						
12:	146						
13:	147	146					
14:	148	147	146				
15:	148	147			146		
16:	149	148	147			146	
17:	150	149	148		147		
18:	150	149			148	147	
19:	150		149			148	
20:	150				149		
21:	150					149	
22:	150						
23:	150						
24:	150						
25:	150						
26:	150						

Dismiss

Assembly Code of des.s

```
blk   assembly code
  136    st    %g0,[%i2]
  137    sethi   %hi(L283),%o4
  138    sethi   %hi(_bit),%i0
  139    add   %o4,%lo(L283),%i1
  140    add   %i0,%lo(_bit),%g4
  141    add   %i1,32,%g5
  142    add   %i1,1,%g6
  143    ld   [%i2],%o1
# block 24 (lines 193-193)
  L308:
  144    sll   %o1,1,%o1
  145    st   %o1,[%i2]
  146    ldsb   [%g5],%o0
  147    sll   %o0,2,%o0
  148    ld   [%o0 + %g4],%o0
  149    andcc   %o0,%g1,%g0
  150    be,a   L309
  151    mov   %g0,%o0
# block 25 (lines 193-193)
  152    mov   1,%o0
# block 26 (lines 193-193)
  L309:
# block 27 (lines 193-193)
  153    or   %o1,%o0,%o1
  154    st   %o1,[%i2]
# block 28 (lines 192-192)
  155    sub   %g5,1,%g5
  156    cmp   %g5,%g6
  157    bge,a   L308
  158    ld   [%i2],%o1
# block 29 (lines 192-192)
  159    ret
  160    restore
    .seg   "data"
    .seg   "text"
    .global   _main
  _main:
```

Best Pipeline Dia. | Worst Pipeline Dia.

Implementation

- The X Toolkit and Xlib libraries were used.
- Timing Tree
  - — Best and worst case predictions for multiple instances.
  - — Predictions for functions and loops versus paths and subpaths.

Future Work

- Supporting highlighting and selections of portions of source lines.
- Splitting a loop into sections when there are too many paths.
- Allowing assertions in the source code.
  - — loop iterations
  - — data dependencies

Conclusions

- Friendly interface for assisting programmers in the analysis of timing constraints.
  - Three methods for selecting program portions for predictions are supported.
  - Correspondence between source and machine code levels is shown.
  - Users can only select portions for which timing bounds can be obtained.
- Advantages of both high level and low level timing analysis are achieved.