

Supporting User-Friendly Analysis  
of Timing Constraints

by

Lo Ko and David Whalley  
Florida State University

and

Marion Harmon  
Florida A&M University

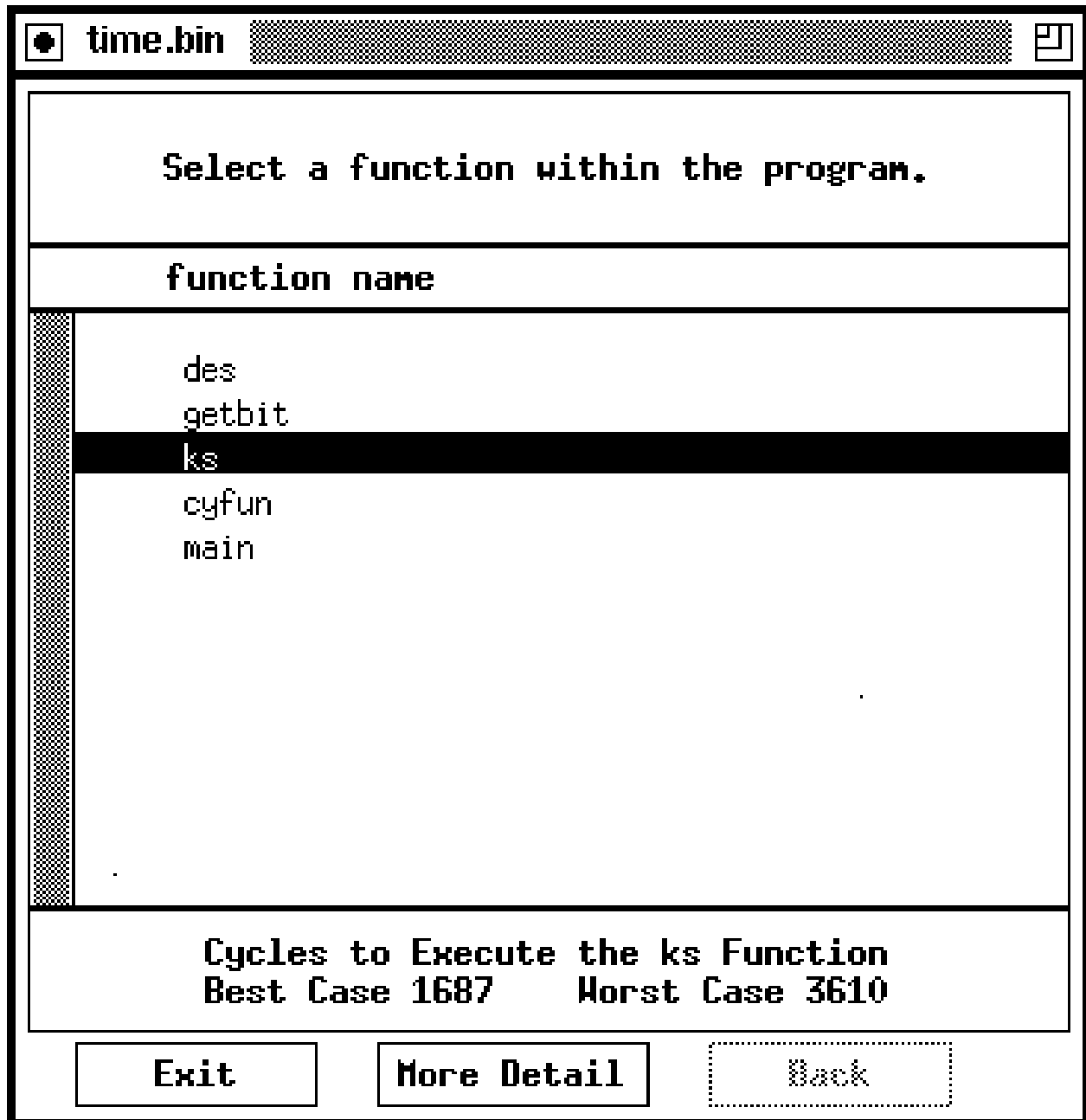
## The Problem

- 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 request and understand timing predictions associated with the more accurate low-level analysis?

## Goals

- A user should be able to quickly select a portion of the program for timing prediction.
- The user should only be allowed to select portions of the program for which timing bounds can be obtained.
- The corresponding portions of the source code and machine code levels of the program selected by the user for timing prediction should be depicted.

## Main Window at Function Level



## 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	}

## Assembly Code Window

Assembly Code of des.s	
blk	assembly code
	sll %o4,1,%o4
	# block 5 (lines 36-36)
	L219:
	ld [%o5],%o0
	cmp %o0,%o0
	be,a L224
	st %o0,[%sp + ,1_itmp]
	# block 6 (lines 37-38)
	st %o0,[%o5]
	mov 1,%i0
	add %sp,,0_STARG,%i4
	sethi %hi(L214),%i16
	add %i16,%lo(L214),%i13
	add %i13,12,%i16
	add %i13,192,%i2
	# block 7 (lines 38-38)
	L227:
	ld [%i1 + 4],%o1
	st %o1,[%sp + (,0_STARG + 4)]
	ld [%i1],%o0
	st %o0,[%sp + ,0_STARG]
	mov %i14,%o0
	mov %i10,%o1
	call _ks,3
	mov %i16,%o2
	# block 8 (lines 38-38)
	add %i16,12,%i16
	cmp %i16,%i2
	ble L227
	add %i10,1,%i10
	# block 9 (lines 40-40)
	st %o0,[%sp + ,1_itmp]
	# block 10 (lines 40-41)
	L224:
	mov 32,%i1
	cmp %i1,1
	bl L230
	st %o0,[%sp + (,1_itmp + 4)]
	# block 11 (lines 41-41)
	add %sp,,1_STARG,%i16

## Methods for Selecting Code Portions

- Two ways to select code portions for timing predictions.
  - 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

The screenshot shows a window titled "time.bin" with a standard Mac OS-style title bar. The main content area contains the text "Select a loop within the function des." followed by a table with three columns: "loop name", "source lines", and "nest level". The table lists five entries: "entire function", "LOOP 1", "LOOP 2", "LOOP 3", and "LOOP 5". The "LOOP 4" row is highlighted with a dark background. Below the table, there is a text box stating "Cycles to Execute Loop 4 within des" with "Best Case 13081" and "Horst Case 25866". At the bottom of the window are three buttons: "Exit", "More Detail", and "Back".

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
<b>LOOP 4</b>	<b>45..50</b>	<b>1</b>
LOOP 5	56..58	1

Cycles to Execute Loop 4 within des  
Best Case 13081      Horst Case 25866

Exit      More Detail      Back



## Main Window at Path Level

time.bin
☐

Select a path within loop 1  
of the function main.

path	blocks	source lines	
entire loop 1		7..17	
-----			
path 1			
	2	8..8	
	3	8..8	
	6	10..10	
	7	10..10	
	9	12..12	
	11	15..16	
	12..13	16..17	loop 3
	14	7..7	
-----			
path 2			

**Cycles to Execute Path 1 within Loop 1**  
**Best Case 1419    Horst Case 1527**

Exit

More Detail

Back

## Main Window at Subpath Level

time.bin
☐

Select a subpath within path 1  
within loop 1 of the function main.

blocks	source lines
2	8..8
3	8..8
6	10..10
7	10..10
9	12..12
11	15..16
12..13	16..17      loop 3
14	7..7

**Cycles to Execute Subpath from Block 7 To Block 11 Best Case 8      Worst Case 35**

Exit

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;
53	

Select Path    Accept    Cancel    Clear All

## Best Case Path

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

## Worst Case Path

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;

## Selecting an Infeasible Path

C Source Code of des.c	
line #	source code
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	}
61	
62	unsigned long getbit(source,bitno,nbits)
63	immense source;
64	int bitno,nbits;
65	{
66	if (bitno <= nbits)
67	return bit[bitno] & source,r ? 1L : 0L;
68	else
69	return bit[bitno-nbits] & source,l ? 1L : 0L;
70	}
71	
72	void ks(key,n,kn)
73	immense key;
74	great *kn;
75	int n;
76	{
77	static immense icd;
78	static char ipc1[57]={0,57,49,41,33,25,17,9,1,58,50,
79	42,34,26,18,10,2,59,51,43,35,27,19,11,3,60,
80	52,44,36,63,55,47,39,31,23,15,7,62,54,46,38,
81	30,22,14,6,61,53,45,37,29,21,13,5,28,20,12,4};
82	static char ipc2[49]={0,14,17,11,24,1,5,3,28,15,6,21,
83	10,23,19,12,4,26,8,16,7,27,20,13,2,41,52,31,
84	37,47,55,30,40,51,45,33,48,44,49,39,56,34,
85	53,46,42,50,36,29,32};
86	
87	
88	
89	
90	
91	
92	
93	
94	
95	
96	
97	
98	
99	
100	
101	
102	
103	
104	
105	
106	
107	
108	
109	
110	
111	
112	
113	
114	
115	
116	
117	
118	
119	
120	
121	
122	
123	
124	
125	
126	
127	
128	
129	
130	
131	
132	
133	
134	
135	
136	
137	
138	
139	
140	
141	
142	
143	
144	
145	
146	
147	
148	
149	
150	
151	
152	
153	
154	
155	
156	
157	
158	
159	
160	
161	
162	
163	
164	
165	
166	
167	
168	
169	
170	
171	
172	
173	
174	
175	
176	
177	
178	
179	
180	
181	
182	
183	
184	
185	
186	
187	
188	
189	
190	
191	
192	
193	
194	
195	
196	
197	
198	
199	
200	
201	
202	
203	
204	
205	
206	
207	
208	
209	
210	
211	
212	
213	
214	
215	
216	
217	
218	
219	
220	
221	
222	
223	
224	
225	
226	
227	
228	
229	
230	
231	
232	
233	
234	
235	
236	
237	
238	
239	
240	
241	
242	
243	
244	
245	
246	
247	
248	
249	
250	
251	
252	
253	
254	
255	
256	
257	
258	
259	
260	
261	
262	
263	
264	
265	
266	
267	
268	
269	
270	
271	
272	
273	
274	
275	
276	
277	
278	
279	
280	
281	
282	
283	
284	
285	
286	
287	
288	
289	
290	
291	
292	
293	
294	
295	
296	
297	
298	
299	
300	
301	
302	
303	
304	
305	
306	
307	
308	
309	
310	
311	
312	
313	
314	
315	
316	
317	
318	
319	
320	
321	
322	
323	
324	
325	
326	
327	
328	
329	
330	
331	
332	
333	
334	
335	
336	
337	
338	
339	
340	
341	
342	
343	
344	
345	
346	
347	
348	
349	
350	
351	
352	
353	
354	
355	
356	
357	
358	
359	
360	
361	
362	
363	
364	
365	
366	
367	
368	
369	
370	
371	
372	
373	
374	
375	
376	
377	
378	
379	
380	
381	
382	
383	
384	
385	
386	
387	
388	
389	
390	
391	
392	
393	
394	
395	
396	
397	
398	
399	
400	

Select Path    Accept    Cancel    Clear All

## Selecting a Single Path

C Source Code of des.c	
line #	source code
153	0,9,0,4,12,0,7,10,0,0,5,9,11,10,9,11,15,14,
154	0,10,3,10,2,3,13,5,3,0,0,5,5,7,4,0,2,5,
155	0,0,5,2,4,14,5,6,12,0,3,11,15,14,8,3,8,9,
156	0,5,2,14,8,0,11,9,5,0,6,14,2,2,5,8,3,6,
157	0,7,10,8,15,9,11,1,7,0,8,5,1,9,6,8,6,2,
158	0,0,15,7,4,14,6,2,8,0,13,9,12,14,3,13,12,11};
159	static char ibin[16]={0,8,4,12,2,10,6,14,1,9,5,13,3,11,7,15};
160	great ie;
161	unsigned long itmp,ietmp1,ietmp2;
162	char iec[9];
163	int jj,irow,icol,iss,j,l,m;
164	
165	ie,r=ie,c=ie,l=0;
166	for (j=16,l=32,m=48;j>=1;j--,l--,m--) {
167	ie,r = (ie,r <<=1)   (bit[iet[j]] & ir ? 1 : 0);
168	ie,c = (ie,c <<=1)   (bit[iet[l]] & ir ? 1 : 0);
169	ie,l = (ie,l <<=1)   (bit[iet[m]] & ir ? 1 : 0);
170	}
171	ie,r ^= k,r;
172	ie,c ^= k,c;
173	ie,l ^= k,l;
174	ietmp1=((unsigned long) ie,c << 16)+(unsigned long) ie,r;
175	ietmp2=((unsigned long) ie,l << 8)+((unsigned long) ie,c >> 8);
176	for (j=1,m=5;j<=4;j++,m++) {
177	iec[j]=ietmp1 & 0x3fL;
178	iec[m]=ietmp2 & 0x3fL;
179	ietmp1 >>= 6;
180	ietmp2 >>= 6;
181	}
182	itmp=0L;
183	for (jj=8;jj>=1;jj--) {
184	j =iec[jj];
185	irow=((j & 0x1) << 1)+((j & 0x20) >> 5);
186	icol=((j & 0x2) << 2)+(j & 0x4)
187	+((j & 0x8) >> 2)+((j & 0x10) >> 4);
188	iss=is[icol][irow][jj];
189	itmp = (itmp <<= 4)   ibin[iss];
190	}

Select Path    Accept    Cancel    Clear All

## Expanded Selected Path

C Source Code of des.c	
line #	source code
153	0,9,0,4,12,0,7,10,0,0,5,9,11,10,9,11,15,14,
154	0,10,3,10,2,3,13,5,3,0,0,5,5,7,4,0,2,5,
155	0,0,5,2,4,14,5,6,12,0,3,11,15,14,8,3,8,9,
156	0,5,2,14,8,0,11,9,5,0,6,14,2,2,5,8,3,6,
157	0,7,10,8,15,9,11,1,7,0,8,5,1,9,6,8,6,2,
158	0,0,15,7,4,14,6,2,8,0,13,9,12,14,3,13,12,11};
159	static char ibin[16]={0,8,4,12,2,10,6,14,1,9,5,13,3,11,7,15};
160	great ie;
161	unsigned long itmp,ietmp1,ietmp2;
162	char iec[9];
163	int jj,irow,icol,iss,j,l,m;
164	
165	ie,r=ie,c=ie,l=0;
166	for (j=16,l=32,m=48;j>=1;j--,l--,m--) {
167	ie,r = (ie,r <<=1)   (bit[iet[j]] & ir ? 1 : 0);
168	ie,c = (ie,c <<=1)   (bit[iet[l]] & ir ? 1 : 0);
169	ie,l = (ie,l <<=1)   (bit[iet[m]] & ir ? 1 : 0);
170	}
171	ie,r ^= k,r;
172	ie,c ^= k,c;
173	ie,l ^= k,l;
174	ietmp1=((unsigned long) ie,c << 16)+(unsigned long) ie,r;
175	ietmp2=((unsigned long) ie,l << 8)+((unsigned long) ie,c >> 8);
176	for (j=1,m=5;j<=4;j++,m++) {
177	iec[j]=ietmp1 & 0x3fL;
178	iec[m]=ietmp2 & 0x3fL;
179	ietmp1 >>= 6;
180	ietmp2 >>= 6;
181	}
182	itmp=0L;
183	for (jj=8;jj>=1;jj--) {
184	j =iec[jj];
185	irow=((j & 0x1) << 1)+((j & 0x20) >> 5);
186	icol=((j & 0x2) << 2)+(j & 0x4)
187	+((j & 0x8) >> 2)+((j & 0x10) >> 4);
188	iss=is[icol][irow][jj];
189	itmp = (itmp <<= 4)   ibin[iss];
190	}

Select Path    Accept    Cancel    Clear All



## 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 partial highlighting of source lines.
- Displaying pipeline diagrams for subpaths.
- Permitting finer grain level of requests for timing predictions.
- Allowing assertions in the source code.
  - loop iterations
  - timing constraints

## Displaying Pipeline Diagrams

Best Case Pipeline Diagram							
cycle #	IF	ID	EX	FPEX	CA	WB	FWB
1:	9						
2:	10	9					
3:	11	10					
4:	12	11	10				
5:	13	12	11		10		
6:	14	13	12		11	10	
7:	15	14	13		12	11	
8:		15			13	12	
9:			15			13	
10:					15		
11:						15	

Dismiss

blk		assembly code
		.seg "data"
		.common _g,4,"data"
		.seg "text"
		.global _main
		_main:
	# block 1 (lines 7-8)	
	0	save %sp,(-96),%sp
	1	sethi %hi(L16),%o0
	2	sethi %hi(_g),%o1
	3	add %o0,%lo(L16),%o0
	4	call _scanf,2
	5	add %o1,%lo(_g),%o1
	# block 2 (lines 8-8)	
	6	mov %g0,%l0
	7	sethi %hi(_g),%l1
	8	ld [%l1 + %lo(_g)],%o1
	# block 3 (lines 9-9)	
	L19:	
	9	call .mul,2
	10	mov %o1,%o0
	11	st %o0,[%l1 + %lo(_g)]
	# block 4 (lines 8-8)	
	12	add %l0,1,%l0
	13	cmp %l0,10
	14	bl,a L19
	15	ld [%l1 + %lo(_g)],%o1
	# block 5 (lines 10-10)	
	16	sethi %hi(L22),%o0
	17	add %o0,%lo(L22),%o0
	18	call _printf,2
	19	ld [%l1 + %lo(_g)],%o1
	# block 6 (lines 10-10)	
	20	ret
	21	restore
		.seg "data"
	L22:	
		.ascii "%d\n\0"
	L23:	

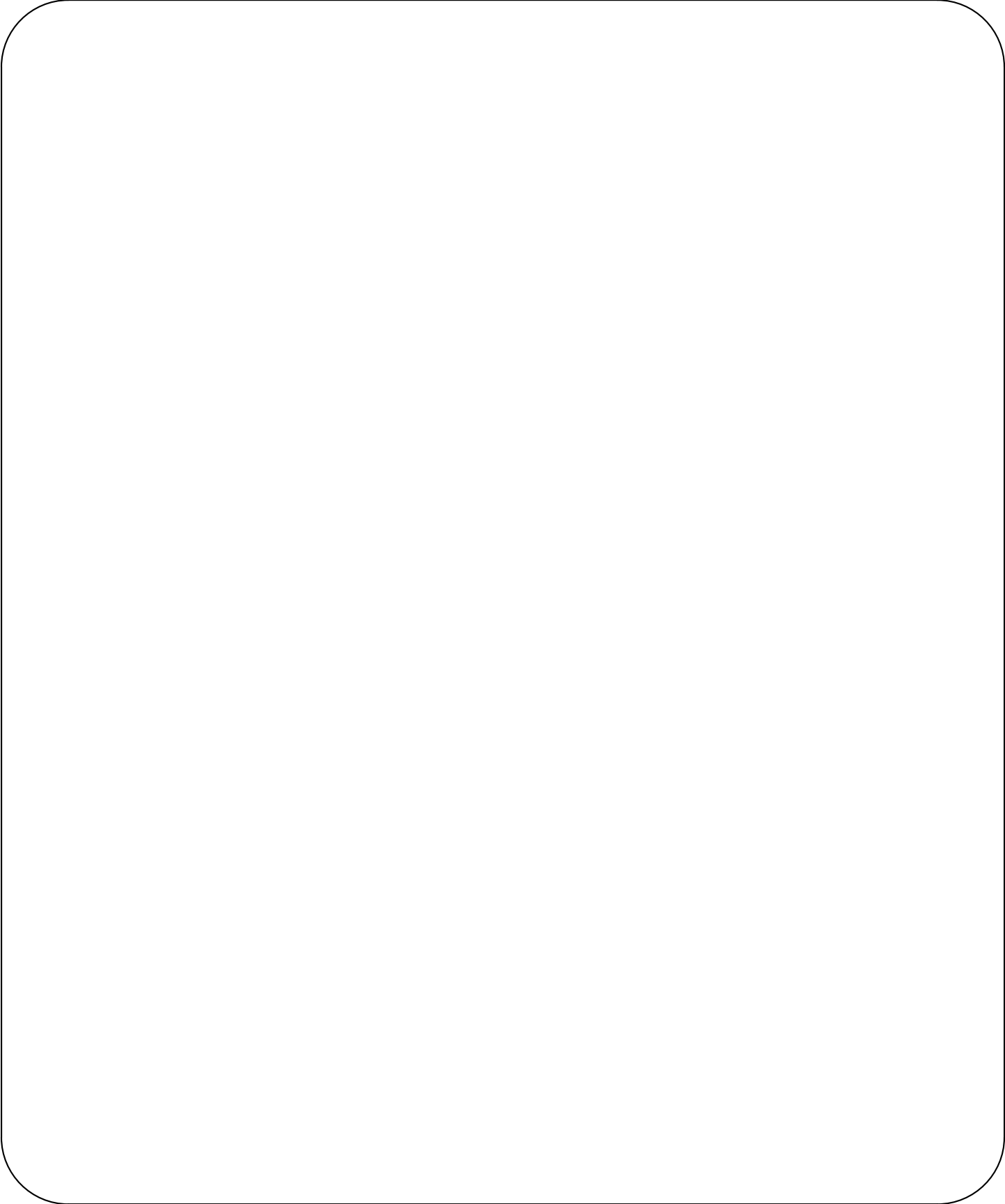
Dismiss

Worst Case Pipeline Diagram							
cycle #	IF	ID	EX	FPEX	CA	WB	FWB
1:	9						
2:	10	9					
3:	11	10					
4:	12	11	10				
5:	12		11		10		
6:	12				11	10	
7:	12					11	
8:	12						
9:	12						
10:	12						
11:	12						
12:	12						
13:	12						
14:	13	12					
15:	14	13	12				
16:	15	14	13		12		
17:		15			13	12	
18:			15			13	
19:					15		
20:						15	

Dismiss

Best Pipeline Dia.

Worst Pipeline Dia.



## Conclusions

- Friendly interface for assisting programmers in the analysis of timing constraints.
  - Two 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.