Worst Case Execution Time Analysis for Synthesized Hardware

Junhee Yoo, Xingguang Feng
and Kiyoung Choi,
(School of EECS, Seoul National University)

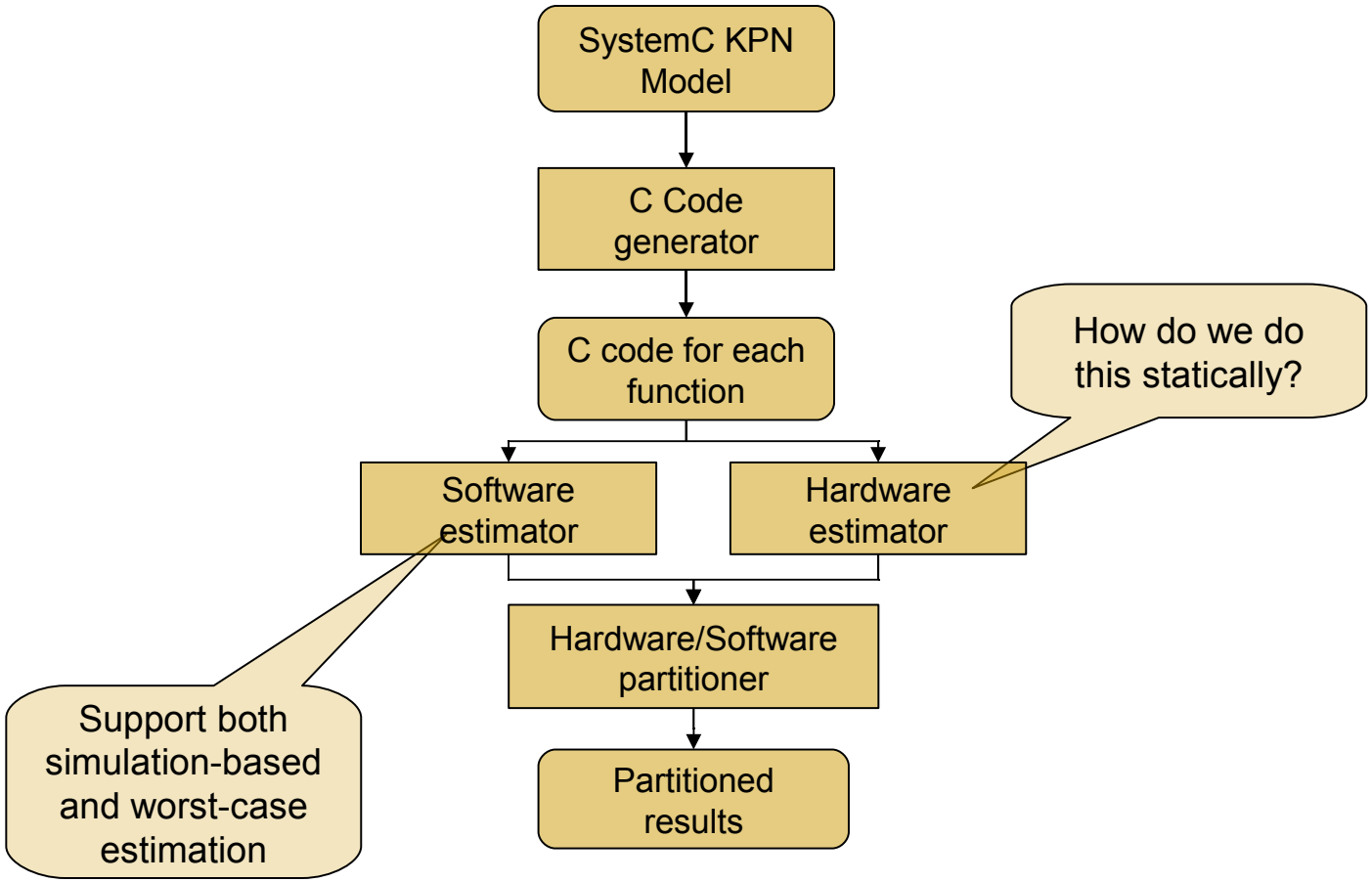
Euiyoung Chung and Kyumyung Choi
(System LSI Division, Samsung Electronics)

Presented on ASPDAC 2006

Contents

- Motivation
  - Why do we need such a tool?
  - Limitations of previous approaches
- Related work
- Details of the analysis flow
  - Flow overview
  - Hardware model
  - ILP formulation
  - Execution constraints
- Experiment results
- Conclusions

Motivation

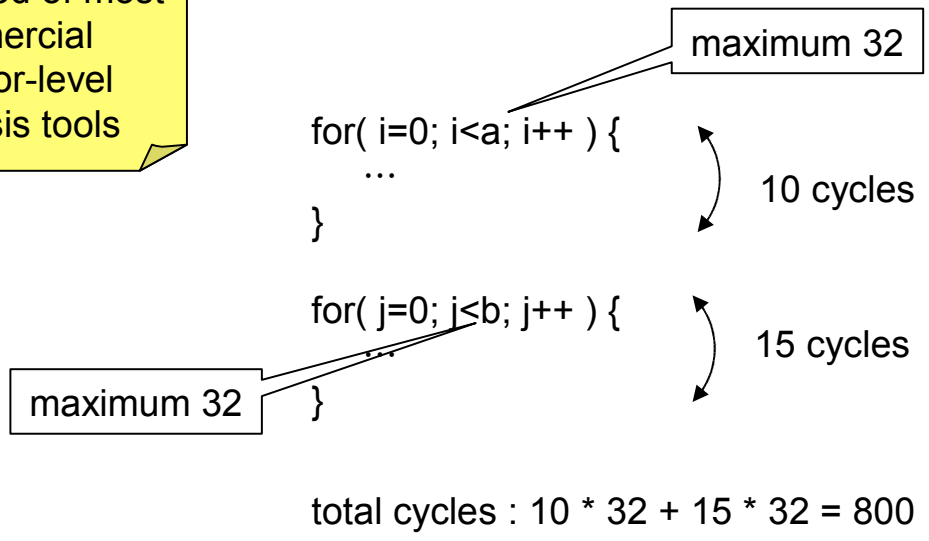


Y. Ahn et al, “An Interactive Environment for SoC Design Starting from KPN in SystemC, *Global Signal Processing Expo., Oct.2004*

Previous implementations

- Simulation-based estimation
  - Needs a lot of simulation effort
  - Cannot guarantee the worst-case execution time
- Naïve loop number calculation

The method of most commercial behavior-level synthesis tools



Motivational example

```
#define NUM_SAMPLES 1024
void karplus_strong(
   unsigned int n, /* … */){
   int i;
   for (i = 0; i < n; i++){
      /* … */
   }
   /* … */
   for (i = n + 1;
        i < NUM_SAMPLES; i++) {
      /* … */
   }
}
```

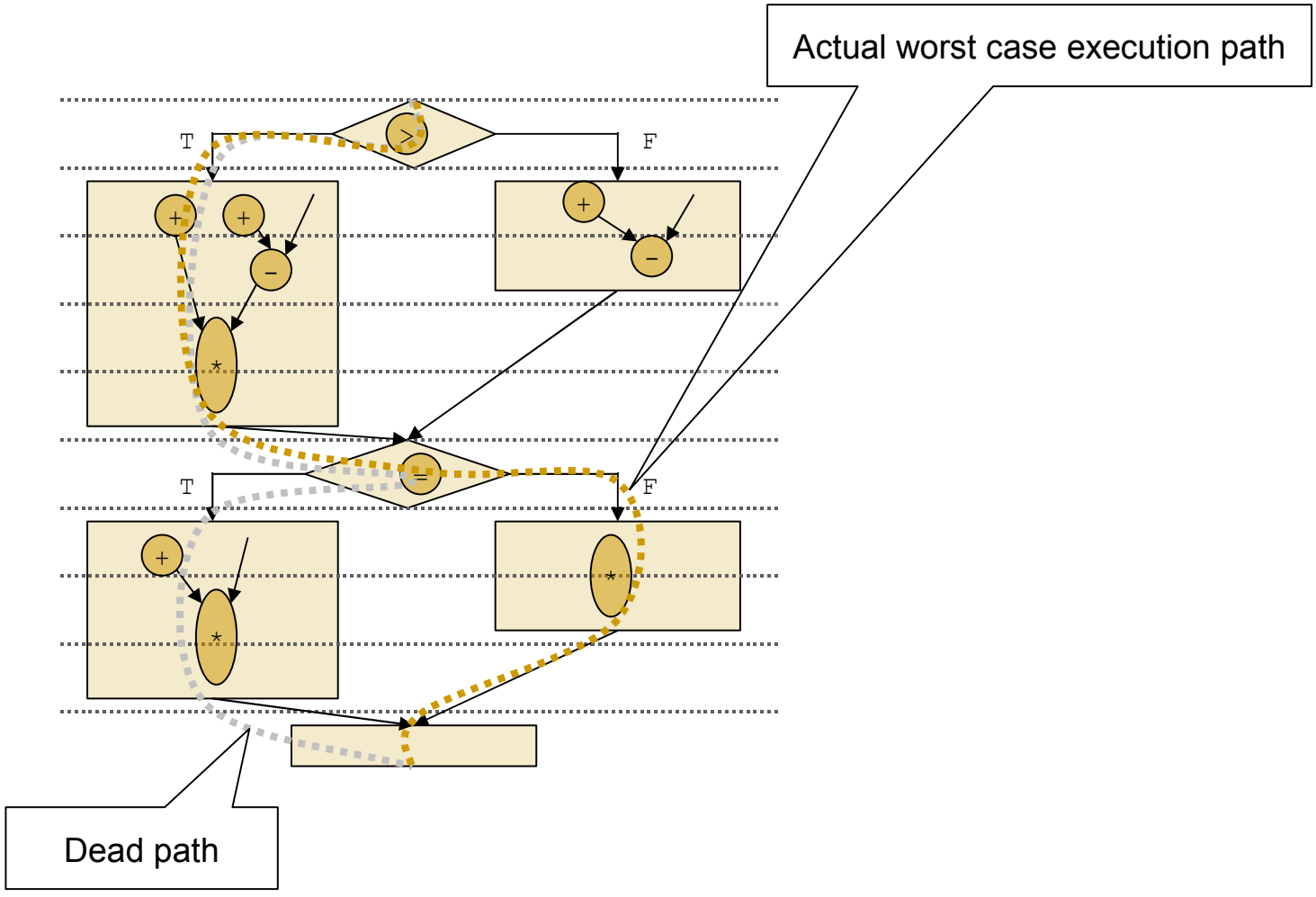
n < NUM_SAMPLES

n iterations, worst case 1023

1023-n iterations, worst case 1023

Using the naïve approach, the number of iterations are overestimated approximately 2x.

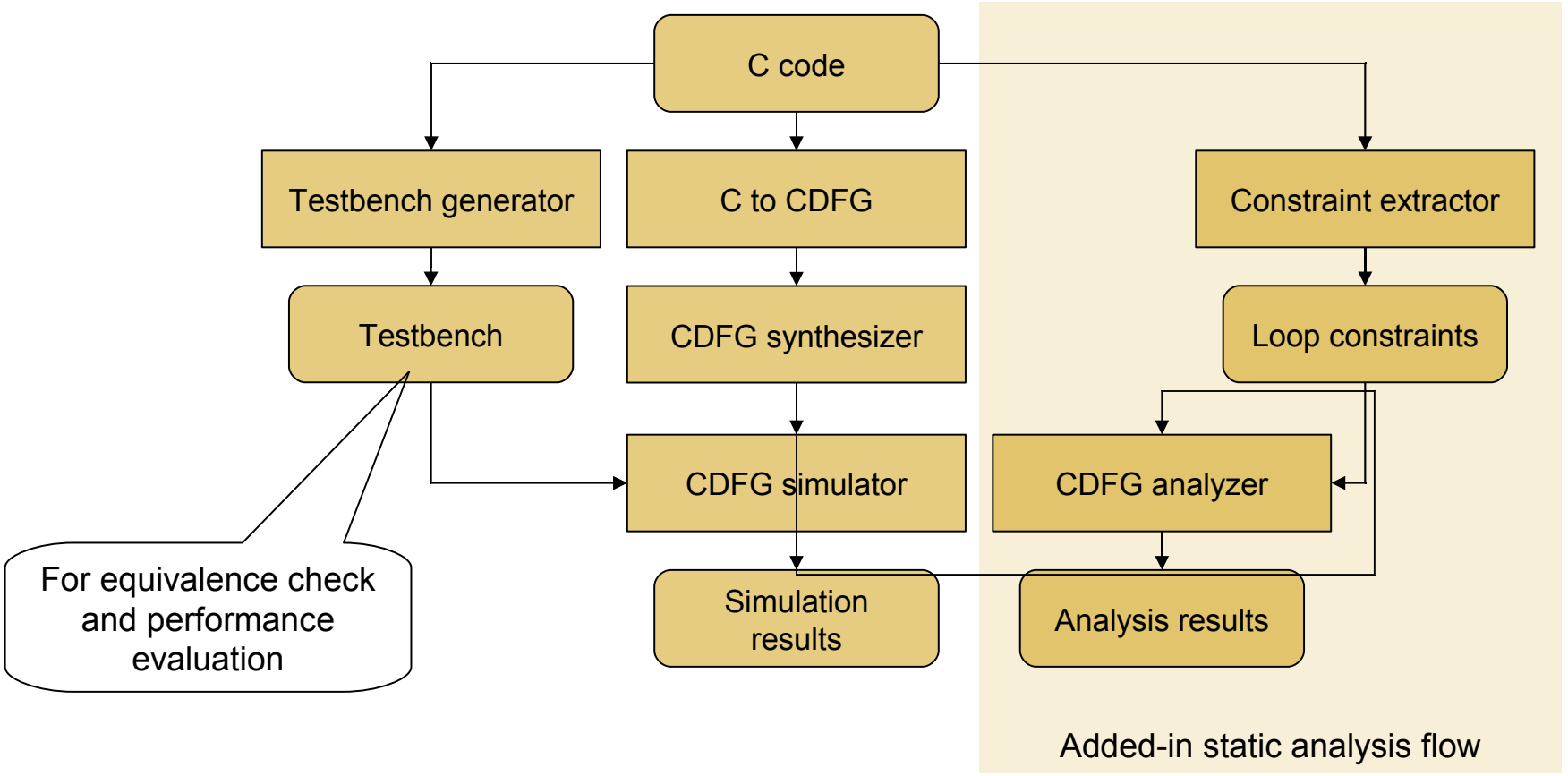
Motivational example



Related work

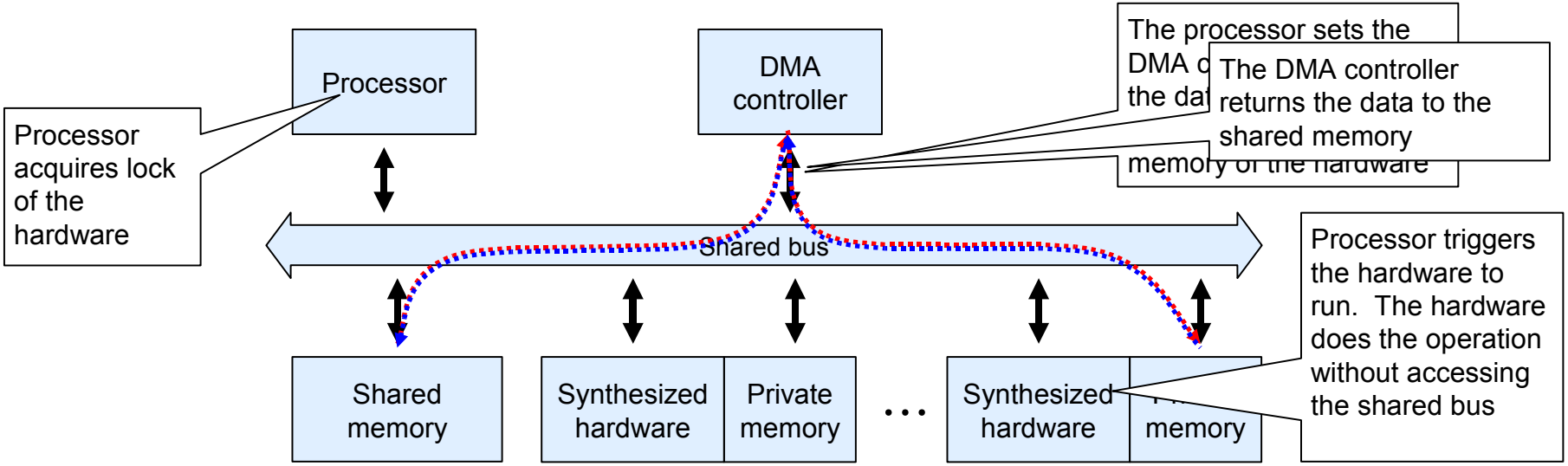
- Y. Li et al, “Efficient microarchitecture modeling and path analysis for real-time software”, IEEE RTSS 1995
  - Presents the basic idea of worst-case execution time (WCET) analysis based on ILP (integer linear programming)
- Software WCET tools
  - Cinderella, http://www.princeton.edu/~yauli/cinderella-2.0/
  - SymTA/s, http://www.symta.org/
- To the best of our knowledge, there was no WCET analysis tool for (behavior-level) synthesized hardware

Hardware analysis flow



Hardware restriction

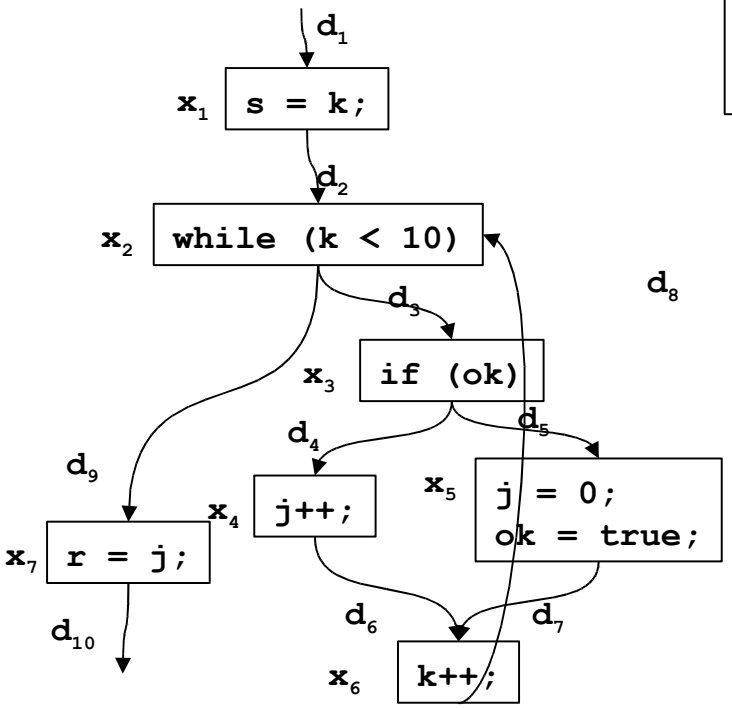
- Restricts global communication while the hardware runs
- Example:



- Partitions the analysis into two sub-problems
  - Scheduling analysis of shared bus (beyond the scope of this paper)
  - Worst case execution time analysis of synthesized hardware
- Using a DMA for on-chip communication is reasonable enough for many applications

ILP formulation

```
/* k >= 0 */
s = k;
while (k < 10)
{
  if (ok)
    j++;
  else {
    j = 0;
    ok = true;
  }
  k++;
}
r = j;
```



'inflow'
= 'outflow'
= number of times the block is executed

$$d_1 = 1$$
$$x_1 = d_1 = d_2$$
$$x_2 = d_2 + d_8 = d_3 + d_9$$
$$x_3 = d_3 = d_4 + d_5$$
$$x_4 = d_4 = d_6$$
$$x_5 = d_5 = d_7$$
$$x_6 = d_6 + d_7 = d_8$$
$$x_7 = d_9 = d_{10}$$

$0 \le x_3 \le 10$ — Loop is executed at most 10 times

$x_5 \le x_1$ — The 'else' path is taken only once per execution

d_i, x_i: Number of times the control path is taken

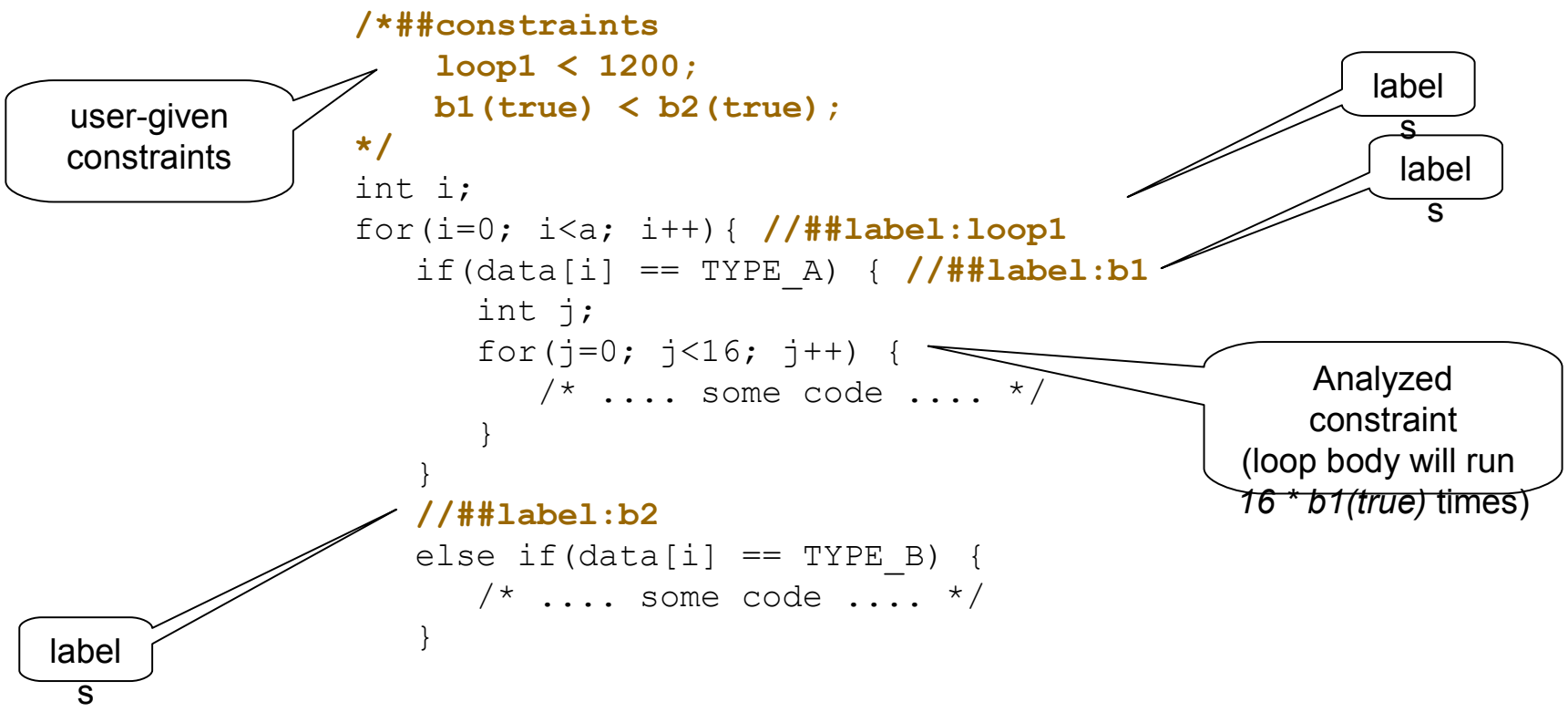
c_i: Number of cycles that it takes to execute the block

Goal : Maximize $\sum c_i x_i$

From Y. Li et al, "Efficient microarchitecture modeling and path analysis for real-time software", IEEE RTSS 1995

Execution constraints

- Constraints can be either user-given or statically analyzed by the analyzer

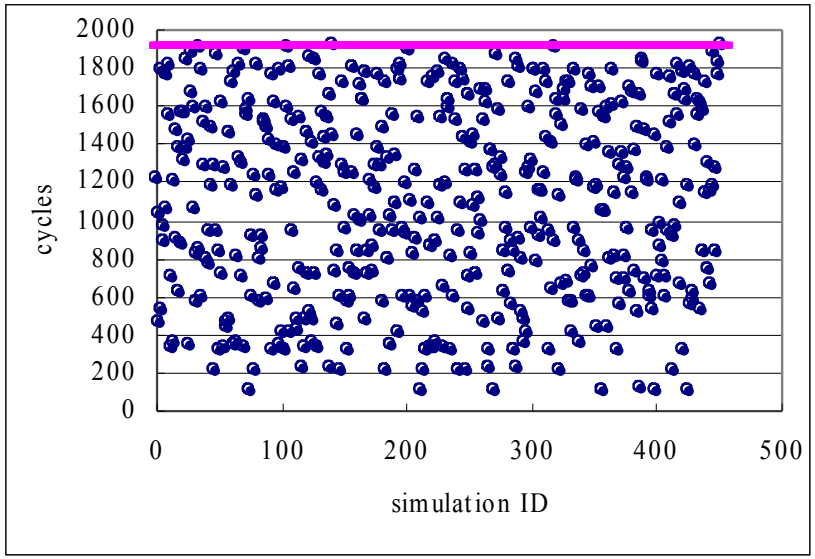


Tool implementation

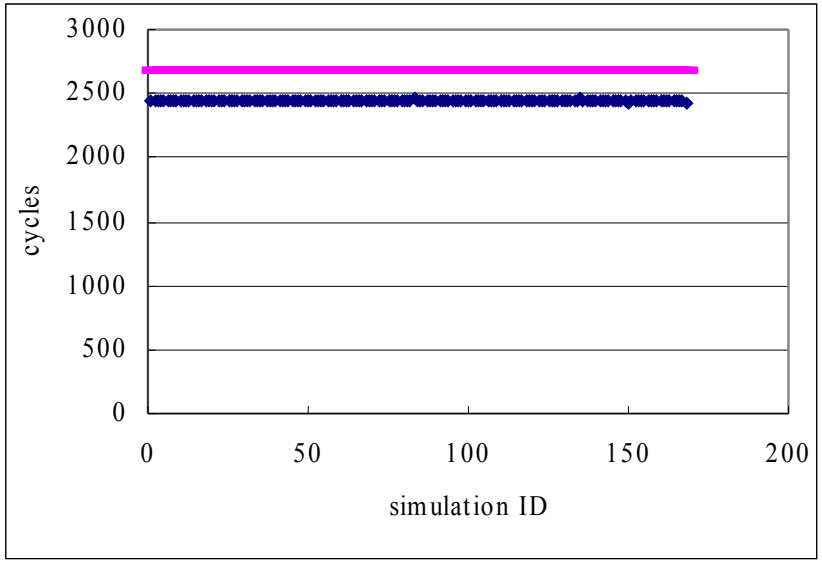
- C language parsing and optimization done using an in-house modified version of SUIF1 (http://suif.stanford.edu/)
- Based on an in-house behavior level synthesis tool from our previous work
  - Tool implemented in standard C++ @ x86 Linux
- GLPK (GNU Linear Programming Kit) for ILP solving (http://www.gnu.org/software/glpk/glpk.html)
- Written both as a subroutine that can be used by other tools, and an independent application

Experiment results

- Two functions from h.263 encoder



SAD_Macroblock



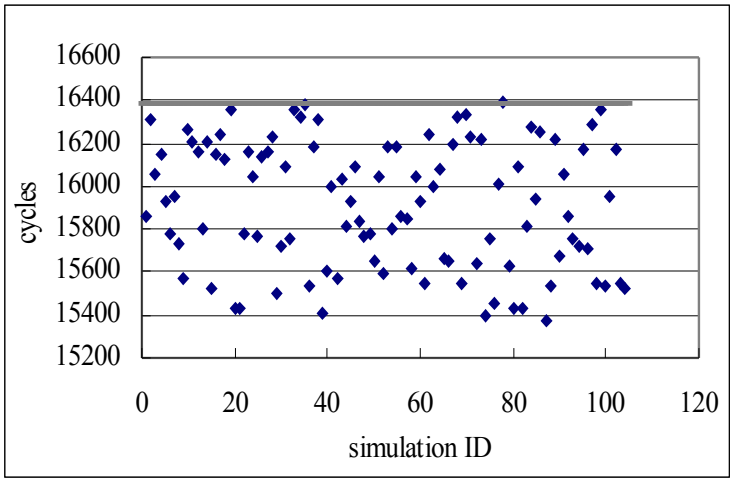
Quantize

Blue dots represent simulation results, while the magenta line represents the analyzed worst-case execution time.

Experiment results (2)

```
#define NUM_SAMPLES 1024
#define COMB_FILTER(cn,cn1,v0,vn,vn1) \
   ((((v0)-MID)*NSF + ((vn)-MID)*(cn) \
   +((vn1)-MID)*(cn1) /256) + MID)

void karplus_strong(int cn, int cn1,
   unsigned int n, short block[NUM_SAMPLES],
   short blockprev[NUM_SAMPLES]){
   int i;
   for (i = 0; i < n; i++){
      block[i] =
         COMB_FILTER(cn, cn1, MID,
         blockprev[NUM_SAMPLES + i - n],
         blockprev[NUM_SAMPLES + i - n - 1] );
   }
   block[n] =
      COMB_FILTER(cn, cn1, MID, block[0],
         blockprev[(NUM_SAMPLES - 1)] );
   for (i = n + 1; i < NUM_SAMPLES; i++) {
      block[i] =
      COMB_FILTER(cn, cn1, MID, block[i - n],
      block[i - n - 1]);
   }
}
```



Naïve calculation : 31,731 cycles

Our approach : 16,385 cycles

Conclusions

- Contribution
  - Presenting a method of doing worst-case execution time of synthesized hardware
- Still more work to be done
  - More research on automatic constraint detection
  - Improving the behavior level synthesis tool
  - Worst case power estimation
  - Integrating bus scheduling and worst case estimation

- For questions, please contact Junhee Yoo, ihavnoid@poppy.snu.ac.kr

Dealing with hierarchical structures

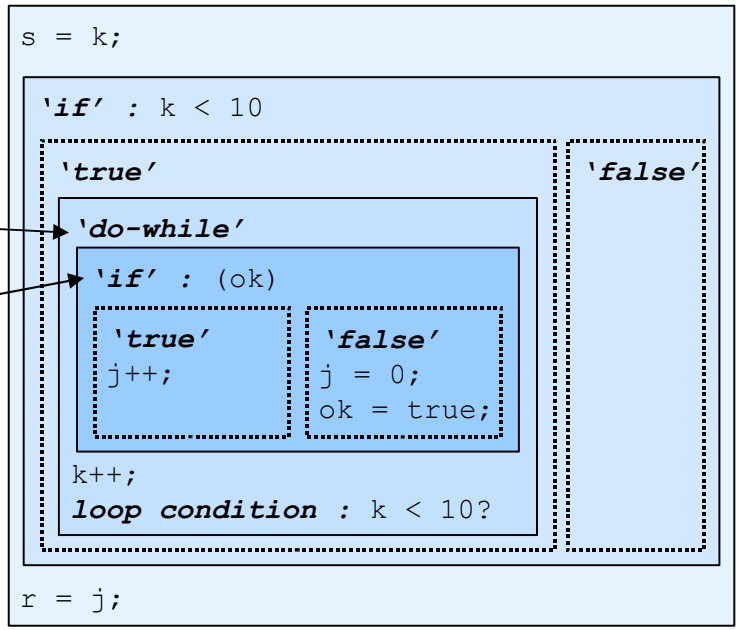
```
/* k >= 0 */
s = k;
if (k < 10){
  do {
    if (ok)
      j++;
    else {
      j = 0;
      ok = true;
    }
    k++;
  }while (k < 10);
}
r = j;
```

body

cond1

loop1

cond2



'inflow'
= 'outflow'
= number of times the block is executed

```
body = 1
cond1 = body
cond1 = cond1.true + cond1.false
cond2 = cond2.true + cond2.false
loop1 = cond2
```

Loop is executed at most 10 times

```
0 body  <= loop1 <= 10 body
```

The 'else' path is taken only once per execution

```
cond2.false <= body
```

FAQ : Isn’t adding constraints too difficult?

- No!
- Most constraints are trivial enough to be automatically analyzed
  - Approximately 70% of loops of h.263 encoder have fixed number of iterations
  - Most of the other loops also have data dependency, but are trivial enough to be easily analyzed
- Although we may have missed some constraints, we still have a result higher than worst case

Constraint optimization