Execution time evaluation

Prof. J.-D. Decotignie
CSEM Centre Suisse d'Electronique et de Microtechnique SA
Jaquet-Droz 1, 2007 Neuchâtel
jean-dominique.decotignie@csem.ch

Plan

- Introduction
- Parameters
- Measurement
- Simulation
- Calculation

csem

Introduction

- It is necessary to know the WCET
  - The best case may also be interesting
  - Different approaches
    - The target and the code are available
    - The code is available but not the target

csem

Impacting parameters

- Related to the code
  - compiler
  - Loop limiting
  - Program structure
- Related to hardware
  - Speed and type of processor
  - Cache memories
  - pipeline
  - Speed and type of memory
  - disks
  - ...

csem

Program structure

```
FOR I:=1 TO N DO
        BEGIN
                IF I<5 THEN
                        X[I] := X[I-1] + 3;
                ELSE
                        X[I] := X[I-1] * 5;
                END;
        END;
```

- Difficult to predict N
  - => limit the number of iterations
    no recursion, no GOTO
  - Numerous functionality
- Source code is necessary

Compilation

- The generated code quality has some influence
- It is necessary to have the compiled code (assembly)

```
FOR I:=1 TO N DO
        BEGIN
                IF I<5 THEN
                        X[I] := X[I-1] + 3;
                ELSE
                        X[I] := X[I-1] * 5;
                END;
        END;
```


Hardware

- Memory access time
- Processor speed
- Data dependencies (multiplication, shift)
- pipeline

INSTANT

	N	N+1	N+2	N+3	
fetch	instruction i	instruction 2	instruct. i+2	– – –	
decode	– – –	instruction i	instruct. i+1	– – –	
execute	– – –		instruction i	instruct. i+1	– – –

Pipeline

- MC 68010

```
        MOVE.W      #Length, D0
LOOP:   MOVE.W      (A0)+,(A1)+
        DBEQ   DO, LOOP
```

- 14 levels on Pentium Pro
- problems
  - data => temporary stall
  - control => stop or purge
  - Loop prediction

Cache memories

- Often not welcome
- However performances are so much better
- cache miss = 24 cycles on SPARC 2
- Problem in case of context switch
  => can be partitioned
- How to predict their behavior

Memories

- Dynamic memories must be refreshed periodically
- The access time of the first byte is longer than the next ones for fast memories (SDRAM)
  => access by bursts (blocks)

Memory management

- Logical addressing (or virtual)
  => translation
- Translation tables with cache for the most recent ones
- Virtual memory
  => disk access time should be taken into account

Degrees of influence

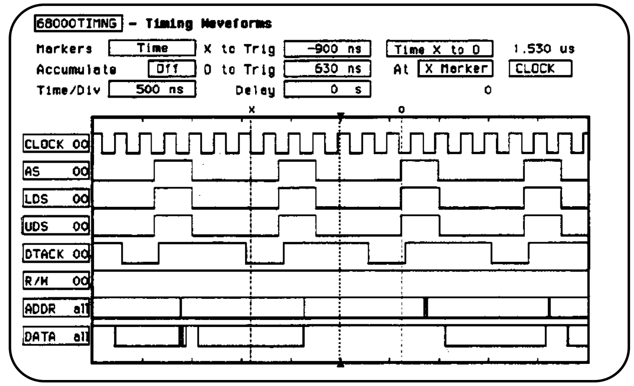
Feature	ratio worst/best	mean /best	ratio worst/mean
Loading data cache	[14.5, 8.0]	[1.5, 1.3]	[9.7, 6.2]
Pipeline effects	12	Not available	Not available
Loading instruction cache	7	1.8	3.9
Wirting data cache	2	1.5	1.3
jumps	1.9	1.1	1.7
ALU instructions	1.8	1.4	1.3
Exceptions	1.2	1.0	1.2
refreshing dynamic RAM	1.1	1.0	1.1
Memory management tables (TLB)	< 88	3.6	<25

Measurement

- Direct measurement
- Double loop technique
- SW stamps
- Indirect measurements

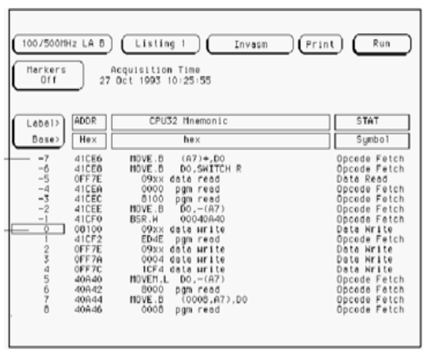
Direct measurement

- Based on a logic analyser



Direct measurement (2)

- Code may be disassembled by the logic analyzer



Direct measurement(3)

- One has to find the corrrespondance between the high level language code and the assembly language
- Beware of the false triggers

```
....
80574   RTS
80576   MOVEM.L      DO-D5/A0-A5,-(SP)
80578   ....
```

- Connection is sometimes difficult
  - Connect the logic analyzer on other ICs
  - Indirect observation (penalty)
- Sometimes, it is impossible to observe (caches)

Software probes

- Close to indirect measurements using a logic analyzer
- Clock reading instructions are inserted in the code
- The clock value is recorded (variables, memory, ...)
- a posteriori analyis

- Introduces perturbations in the measurements
- Precision and resolutions are sometimes limited (not on Pentium)
- However, cheap and easy to deploy

Double loop test

- Used when the internal clock has limited resolution

```
...
GET_TIME(T1);(* start instant *)
FOR I:=1 TO N DO
        (* code for which we want to measure the exec. time *)
END;
GET_TIME(T2);(* end instant *)
GET_TIME(T3);(* start instant *)
FOR I:=1 TO N DO
END;
GET_TIME(T4);(* end instant *)
....
MeanExecutionTime := ((T2-T1)-(T4-T3))/N;
```


Double loop test

- The measured code must be isolated from its context
  => interactions with it should be emulated
- The execution time may vary depending on the presence or absence of the loop (12%)
- Source code may be compiled differently outside its context

Measurements - Analysis

- Simple techniques
- The hardware should be available
- Often intrusive
- One measurement only reflects one execution
  is it the worst (best) case ?

```
FOR I:=1 TO N DO
    BEGIN
        IF I<5 THEN
            X[I] := X[I-1] + 3;
        ELSE
            X[I] := X[I-1] * 5;
        END;
    END;
```

Measurements – Analysis (2)

- Is certainly usable !
- With caches, jumps, pipelines, …

IT BECOMES DIFFICULT TO MEASURE

csem

Simulation

- When the hardware is not available
- Requires detailed informations on the processor operations and its the operations of its peripherals (often confidential)
- The results are highly dependent on the model quality
- Slow
- Problems with the external events (must be emulated)
  => replaced by precedence relationships
- Suffers from the same problems as measurements
  - Are we sure it is the worst (best) case

csem

Calculation

- A whole class of techniques
- Hardware must not be available
- Similar to simulation but we are only interested in the calculation time => may be quicker
- The various techniques do not all take into account all aspects (pipeline, cache, memory management, ...)
  - Old techniques => simple processors
  - Still a very active research domain

csem

Calculation

- Table lookup
- Instruction counting
- Time schemas
- Taking pipelines into account
- Taking caches into account

csem

Table lookup

Instruction	Size	Register	Memory
ASR, ASL	byte, word long	6 + 2n(1/0) 8 + 2n(1/0)	8(1,1)+ -
LSR, LSL	byte, word long	6 + 2n(1/0) 8 + 2n(1/0)	8(1,1)+ -
ROR, ROL	byte, word long	6 + 2n(1/0) 8 + 2n(1/0)	8(1,1)+ -
ROXL, ROXR	byte, word long	6 + 2n(1/0) 8 + 2n(1/0)	8(1,1)+ -

Mode d'adressage		Byte, Word	Long
Dn	Data Register Direct	0(0/0)	0(0/0)
An	Address Register Direct	0(0/0)	0(0/0)
(An)	Address Register Indirect	4(1/0)	8(2/0)
(An)+	Address Register Indirect with Postincrement	4(1/0)	8(2/0)
-(An)	Address Register Indirect with Predecrement	6(1/0)	10(2/0)
d16(An)	Address Register with Displacement	8(2/0)	12(3/0)

csem

Table lookup(2)

- We add the individual execution times
- The assembly code must be available
- The match with the high level language must be done
- Does not take into account caches and pipelines
- Difficult to use at large scale (no technique to compose basic blocks)
- More accurate than measurement in some case
  - Allows to evaluate the longest paths (resp. shortest)

csem

Instruction counting

- We count the number of instructions
- The result is multiplied by a constant to get the worst case and another to get the best case execution time
- Constants are determined by measuring the execution of a representative set of programs
- simple
- Acceptable results

csem

Time schemas

- due to Alan Shaw
- The starting point is the high level language program
- 4 steps
  - decomposition of a high level language statement in its basic blocs (defined in its time schema)
  - Prediction of the assembly language implementation of the basic blocks
  - Calculation of the execution time of the basic block from the generated assembly code
  - Calculation of the execution time of the instruction from the basic block execution times and the time schema

csem

Time schemas - example

- Execution time : $T(S)=[t_{\min}(S),t_{\max}(S)]$
  1) time schema of S1: a:=b+c : T (S1) = T(b) + T(+) + T(c) + T(a) + T(=) => 5 basic blocks.

  Addition on intervals : $[t_1,t_2]+[t_3,t_4]=[t_1+t_3,t_2+t_4]$

  2) prediction of assembly language for each block

b:	MOVE M,R	M and R represent resp.
+:	ADD M,R	memory locations
c:	nothing	and generic registers.
a:	nothing	
:=:	MOVE R,M	

  3 et 4) according to the time schema, S1 execution time will the sum of the durations of the 3 blocks.

csem

Control structures

- S: if (exp) then S1 else S2 end;

$$T(S)=\left[\min\left(t_{1_{low}},t_{2_{low}}\right)\max\left(t_{1_{up}},t_{2_{up}}\right)\right]$$

- with

$$\left[t_{1_{low}},t_{1_{up}}\right]=T(\exp)+T(S1)+T(then)$$

$$\left[t_{2_{low}},t_{2_{up}}\right]=T(\exp)+T(S2)+T(else)$$

csem

Basic block granularity

- Short blocks (terminal symbol of the language)
  - Predicting is simple if code generation follows the parsing structure (often this is the case only for control strcutrures)
  - Problem with compiler optimizations

```
d:= b+c;        MOVE   @B,D0
                ADD    @C,D0
                MOVE   DO,@D
d:= d+a;        ADD    DO,@A
```

  - The variations may be taken into account (parameterized schemas) ex:

$$T(S)=T(\text{var},\text{var}\_type)+T(=,\text{var}\_type,\exp-type)+T(\exp,\exp\_type)$$

csem

Basic block granularity (2)

- Long blocks
- One entry and one exit point
- example: a few consecutive assignment statements
- Code prediction
  - Same algorithm than the compiler
  - From the generated code
- Very good match with measurement

csem

Time schemas - results

Construction	Prediction (short blocks)	Prediction (long blocks)	Measured time	Corrected measurement
Expressions				
a+b	[1.42, 3.46]	[3.46, 3.46]	3.66	3.46
a*b	[5.29, 7.32]	[6.31, 6.31]	6.43	6.31
a>b	[2.44, 4.07]	[3.86, 3.86]	3.97	3.86
Assignments				
a=b	[1.22, 2.03]	[2.03, 2.03]	2.15	2.03
a=b+c	[1.63, 4.07]	[3.66, 3.66]	3.91	3.66
IF simple	[3.46, 4.48]	[3.86, 4.48]	[3.98, 4.71]	[3.86, 4.48]
WHILE simple	[1.83, 6.71]	[2.24, 6.31]	[2.34, 6.62]	[2.24, 6.31]
Empty procedure call	[6.72, 26.44]	[6.72, 6.72]	6.97	6.72

csem

Time schemas – results (2)

- For non trivial programs, the long block approach is the only one practicable

Procedure	Long block calculation	Measurement
Insertion sort	[187.17, 3450.10]	[192.2, 2071.1]
Scheduler	[51.88, 4253.31]	[280.68, 3242.6]
search process list	[25.84, 838.21]	[37.14, 826.6]
allocate idle	[25.02, 129.29]	[52.26, 127.2]
preempt	[105.38, 225.02]	[113.72, 230.7]

csem

Time schemas – analysis

- Close to table lookup
- Control structures may be taken into account
- Gives maximum as well as minimum execution time
- It is also possible to take into account dynamic memory refreshing and periodic interrupts (from timers)
- Approach limited to processors without cache and without pipeline

csem

Extended Time schemas

- Take cache and pipeline into account

	ArrSum	BS	Fib	FFT	Isort	MM	Sqrt
Measurement	242	283	683	5190	2849	9141	302
Calculation pipeline only	592	574	2642	59593	17757	32857	747
same + progr. cache	256	314	710	22297	6709	11653	327
same + data cache	296	346	710	27213	8077	13153	327

- Only for worst case execution time
- Calculation gives a higher result than measurement
- The current version does not eliminate the paths that are not used at execution time

csem

Extended Time schemas

- They exhibit the advantages of the basic time schemas
  - But do not give the (min.) best execution time
- Take into account the effects of pipeline
- Take into account the effects of caches (data and program)
- Not simple
- Give results that are still much higher than measurements
  - Limits of the current models

Conclusion

- Two simple techniques
  - Measurements
  - Instruction counting
- With these techniques, there is no guarantee to find the worst case execution time
  - You should always include overload protection in your scheduling
- For simple processors (microcontrollers), calculating the execution time (min and max) is feasible
  - Better than measurement in terms of risk of exceeding the highest value
- For complex processors, new techniques are coming that are (much) more complicated with results that still need to improve