An Estimation Methodology for Designing Instruction Cache Memory of Embedded Systems

Nikolaos Kroupis, Stylianos Mamagkakis, and Dimitrios Soudris

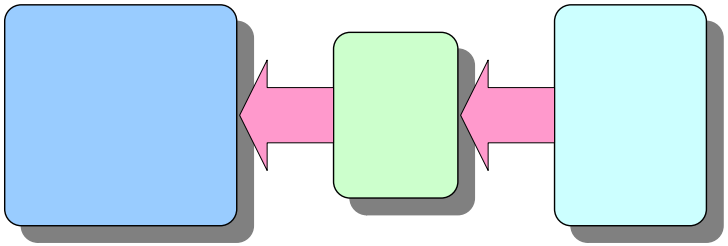
Department of Electrical and Computer Engineering,
Democritus University of Thrace, Xanthi, Greece

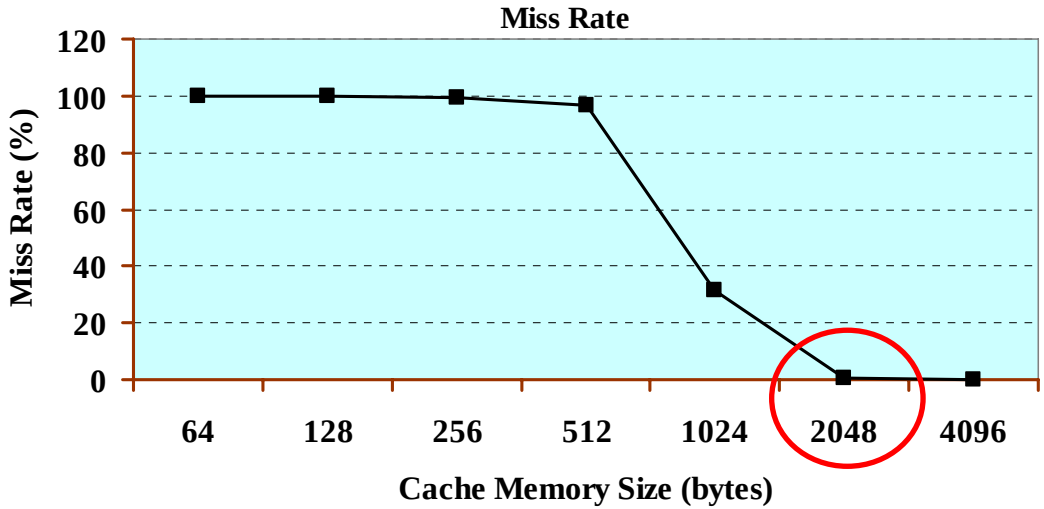
The project is co-funded by the European Social Fund & National Resources - EPEAEK II - PYTHAGORAS II

Presentation Outline

- Instruction caches & processor performance
- Instruction cache assumptions
- The proposed estimation methodology of the number of executed instructions and the number of cache misses
- The implemented estimation software tool
- Experimental results and comparison study
- Conclusions and future work

Cache and Memory Performance





Average Memory Access Time = Hit time + Miss rate x Miss penalty

Improving memory hierarchy performance:

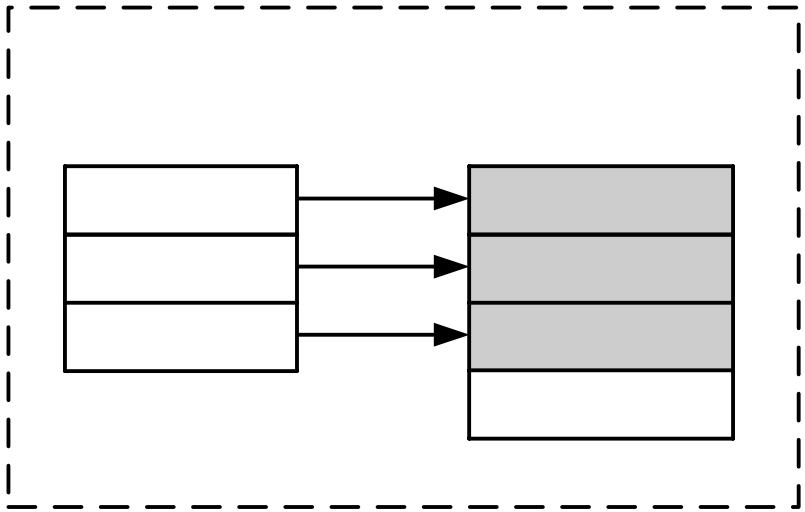
- **Decrease Hit time (IC Technology)**
- **Decrease miss rate (Application Software)**
- **Decrease miss penalty (IC Technology)**

CPU **Instr. Cache** **Instruct. Memory**

Cache Miss Rate Measurements

- **Embedded Processor Core Instruction Set Simulator is needed**
- **Most Instruction Set simulators do not include Cache simulator**
- **Cache simulation is a time consuming procedure**
- **Cache miss rate exploration to find the best cache size and parameters needs days of simulation**

Loop Size & Cache Miss Rate (Type 1)



Loop Type 1

$$Num\_misses = \frac{L\_s}{B\_s}$$

***L_s* = 3 instr.**

Num_misses : Number of instruction misses
L_s : Loop Size in number of instructions
B_s : Instruction cache block size

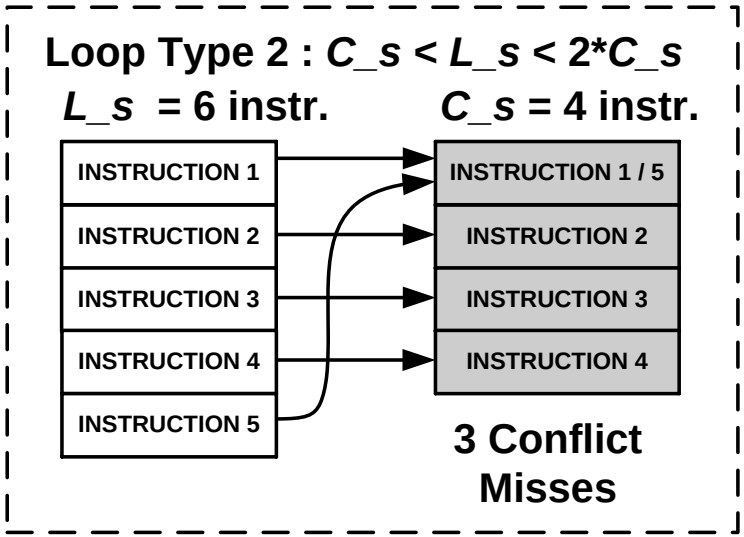
INSTRUCTION 1

INSTRUCTION 2

INSTRUCTION 3

No Conflict

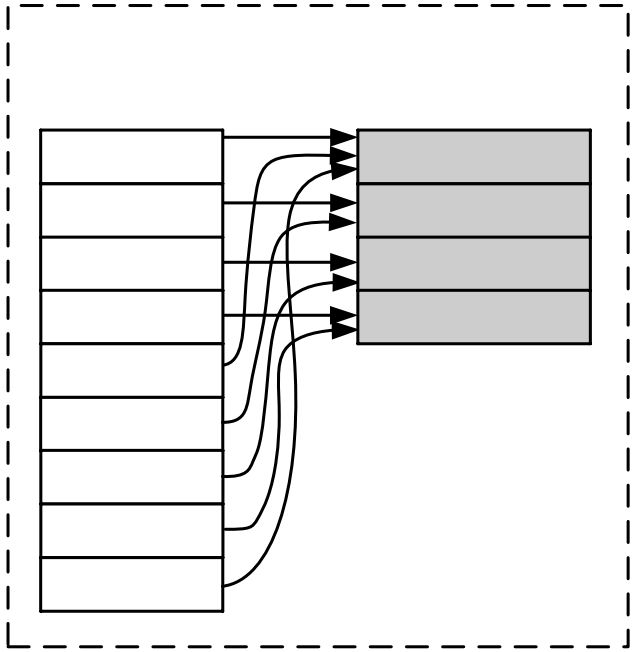
Loop Size & Cache Miss Rate (Type 2)



$$Num\_misses = \frac{L\_s}{B\_s} + (N-1) \times 2 \times \frac{L\_s \bmod C\_s}{B\_s}$$

Num_misses : Number of instruction misses
L_s : Loop Size in number of instructions
B_s : Instruction cache block size
N : Number of loop iterations
C_s : Cache size

Loop Size & Cache Miss Rate (Type 3)



$$Num\_misses = N \times \frac{L\_s}{B\_s}$$

Num_misses : Number of instruction misses
L_s : Loop Size in number of instructions
B_s : Instruction cache block size
N : Number of loop iterations

Loop Type 3

***L_s* = 17 instr.**

INSTRUCTION 1

INSTRUCTION 2

INSTRUCTION 3

INSTRUCTION 4

INSTRUCTION 5

INSTRUCTION 6

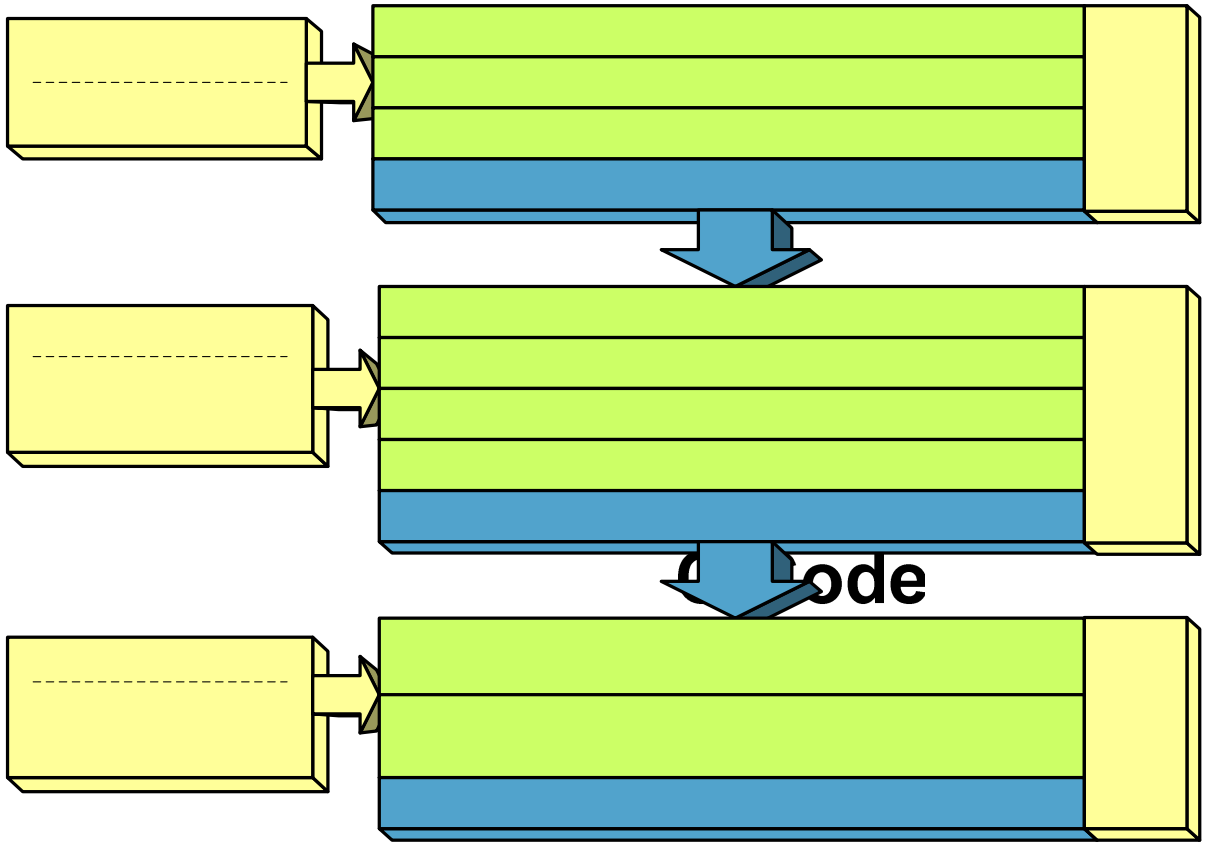
Instruction Cache Miss Rate

$$Num\_references = \frac{L\_s}{B\_s} \times N$$

$$Miss\_rate = \frac{Num\_misses}{Num\_references}$$

Num_misses : Number of instruction misses
Num_references : Number of Memory references
L_s : Loop Size in number of instructions
B_s : Instruction cache block size
N : Number of loop iterations

The Proposed Methodology



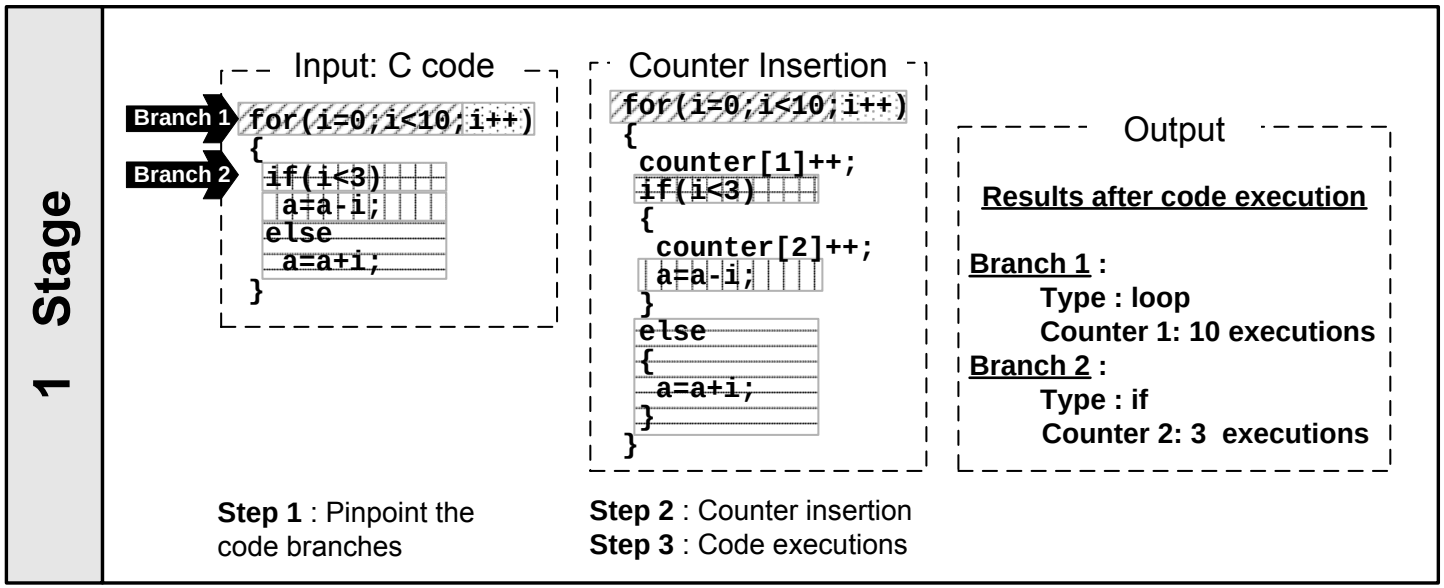
Pinpoint th
Counter in
Code exec
executio

Pinpoint t
Create the

Platform dependent

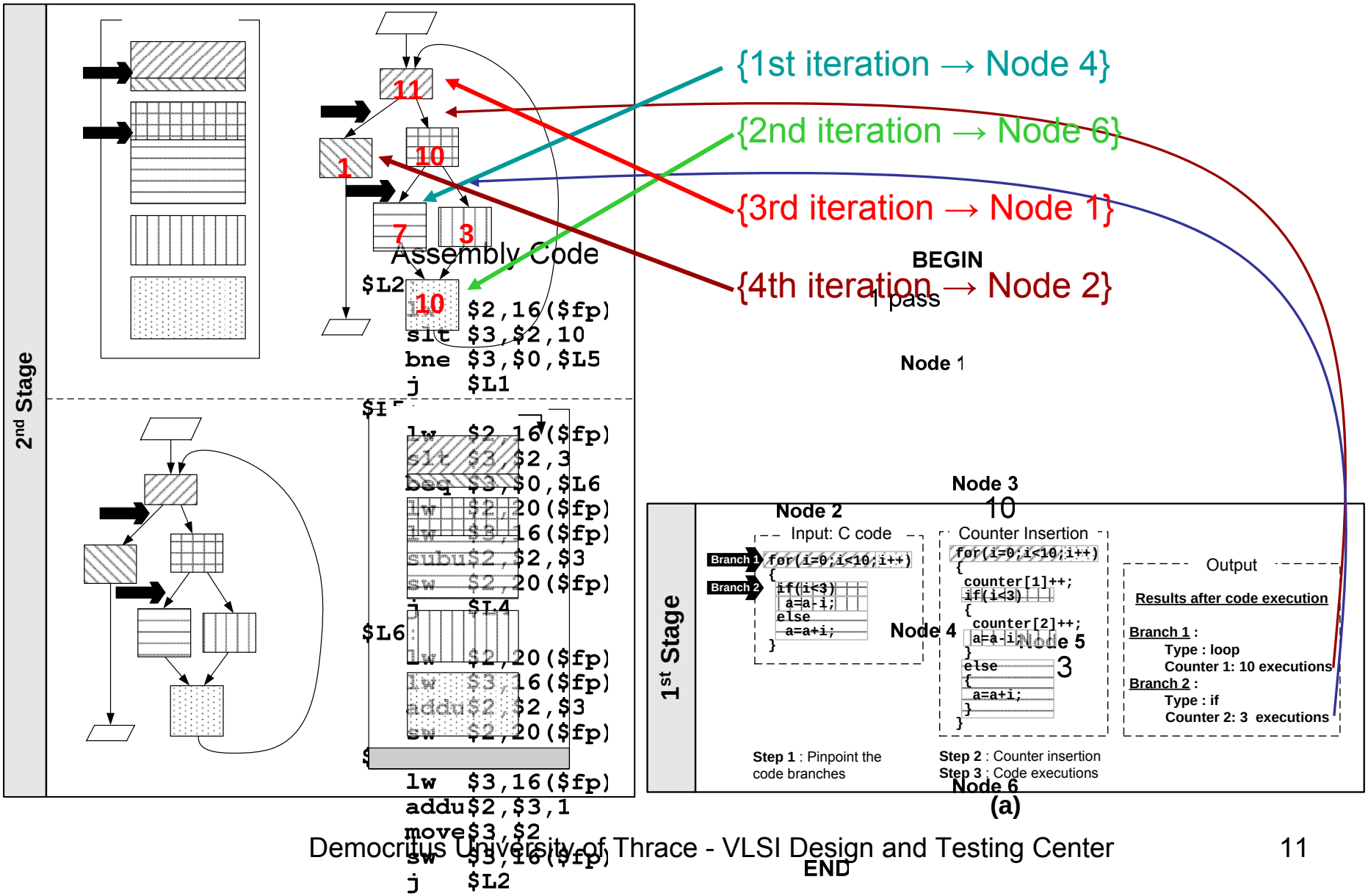
Assembly

The Proposed Methodology: 1st Stage

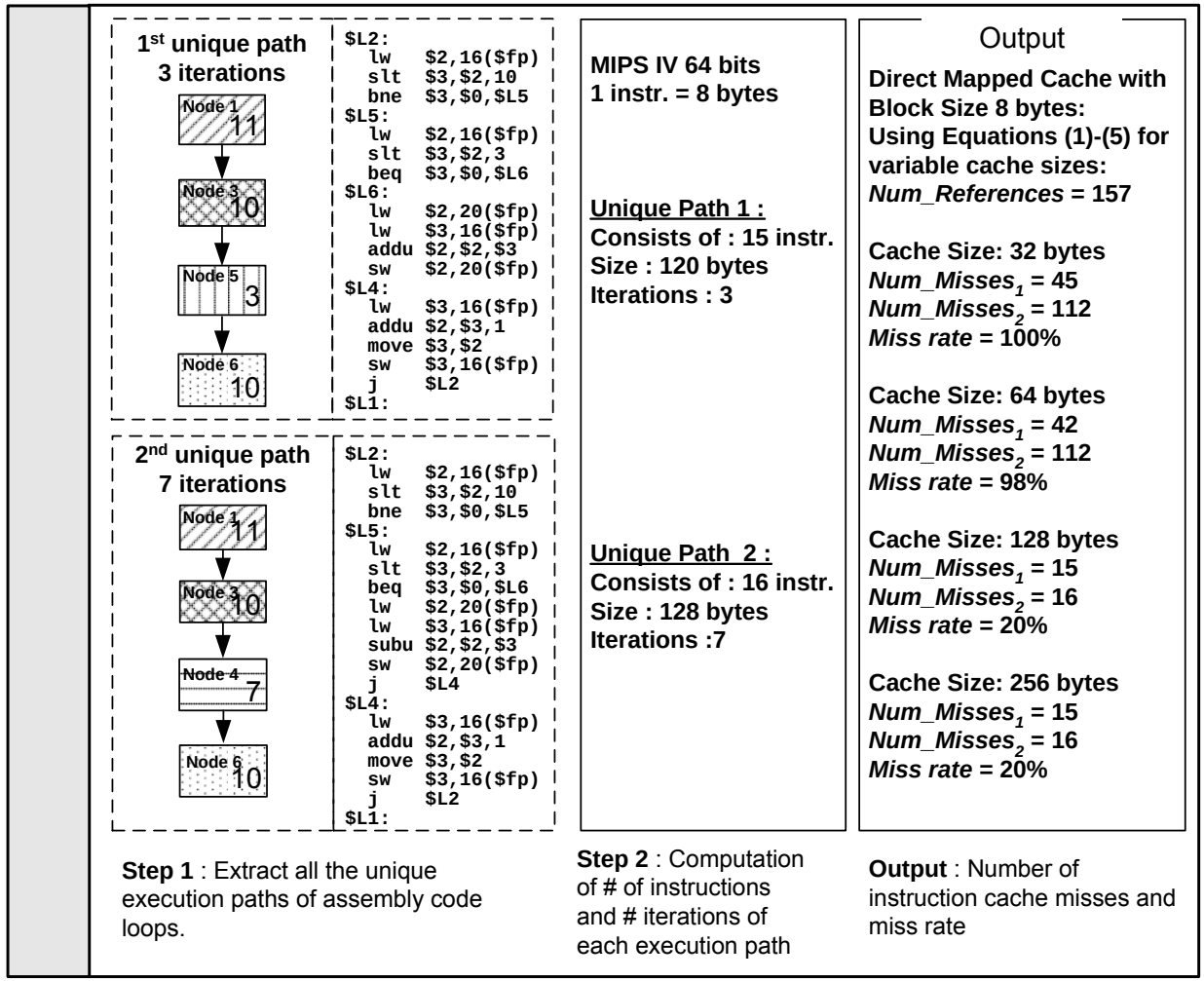


(a)

The Proposed Methodology: 2nd Stage



The Proposed Methodology: 3rd Stage



(c)

FICA Software Tool

Fast Instruction Cache Analyzer (FICA)

- Web based Tool
- PHP Language
- MySQL Database
- MIPS IV Architecture with compiler gcc 2.7.2

- Estimate the number of execution instructions, the number of instruction cache misses and the miss rate
- Extract the critical part of the application which give high miss rate

Number of Executed Instructions

Benchmark	C Code Size (bytes)	Simplescalar (#instructions)	FICA (#instructions)	% Error
FS	1,625	1,138,860,504	1,135,823,110	0.26 %
HS	7,009	37,792,750	36,736,600	2.79 %
3SLOG	3,381	46,717,535	45,865,486	1.82 %
PHODS	3,372	70,865,874	69,375,838	2.10 %
SS	2,587	590,420,249	589,448,105	0.16 %
Wavelet	22,380	39,507,821	39,993,202	1.23 %
Cavity Detector	2,634	18,957,657,996	18,351,605,412	3.19 %
CQ	11,215	6,133,172,759	6,299,874,815	0.26 %
FFT	2,681	687,624	620,154	9.81 %

Number of Instruction Cache Misses

Miss Rate Block size 8 bytes		Cache Size (bytes) 64	128	256	512	1024	2048	4096	Average error
FS	Sim.	100.0	100.0	99.8	99.2	76.8	0.1	0.0	
	FICA	100.0	100.0	99.9	99.1	38.1	0.1	0.0	5.6 %
HS	Sim.	99.9	97.3	92.5	66.4	2.8	1.6	1.5	
	FICA	100.0	95.5	84.8	35.8	3.2	2.0	0.5	6.0 %
3SLOG	Sim.	100.0	99.7	93.1	15.9	1.9	1.8	0.0	
	FICA	100.0	99.3	96.3	12.8	0.9	0.3	0.0	1.3 %
PHODS	Sim.	100.0	99.9	99.6	96.7	31.7	0.8	0.2	
	FICA	100.0	100.0	97.9	94.8	15.5	0.9	0.9	2.9 %
SS	Sim.	99.9	99.9	98.9	79.9	0.5	0.1	0.0	
	FICA	100.0	98.9	97.9	41.2	0.0	0.0	0.0	5.9 %
Wavelet	Sim.	98.7	89.9	50.6	3.4	1.0	0.0	0.0	
	FICA	100.0	91.4	39.3	1.5	0.1	0.0	0.0	2.4 %
Cavity Detector	Sim.	100.0	100.0	94.3	61.4	16.9	0.3	0.1	
	FICA	100.0	100.0	92.4	28.3	0.4	0.0	0.0	7.4 %
CQ	Sim.	100.0	99.4	89.1	46.5	9.6	0.4	0.0	
	FICA	100.0	100.0	73.7	5.8	0.0	0.0	0.0	9.6 %
FFT	Sim.	99.8	98.7	95.7	87.7	7.1	0.5	0.2	
	FICA	100.0	98.0	92.7	43.1	6.1	4.7	4.7	8.3 %

Estimation vs. Simulation Time

(seconds)	Simpescalar	FICA	Speed Up
FS	2.278	0.9	2,531
3SLOG	93	1.3	75
PHODS	142	1.7	86
HS	76	4.5	17
SS	1.182	1.3	909
Wavelet	107	1.5	70
Cavity Detector	37.915	2.9	13,186
CQ	32.268	13.3	2,426
FFT	11	0.9	12

Conclusions

- Number of executed instructions and instruction cache misses can be estimated without simulation process
- Instruction cache memory decisions can be taken from early design steps
- Find out the critical points of the application which give number of cache miss
- Instruction code transformations could be applied

Future Work

- **Methodology extension for second level instruction cache (L2)**
- **Tool extension to accept more embedded processor cores (*ARM, TI,…etc*)**
- **Application of the methodology to more complex system architectures (*e.g. software control caches*)**
- **High level estimation of processor's power consumption using instruction level power parameters**