Cache Miss Heuristics and Preloading Techniques for General-Purpose Programs

Toshihiro Ozawa
Shin'ichiro Nishizaki
Yasunori Kimura

Processor Laboratory
Fujitsu Laboratories Ltd.

Outline

1. Introduction

2. Cache Miss Heuristics
   What types of load instructions should be preloaded?

3. Global Instruction Scheduling
   How to schedule the selected load instructions

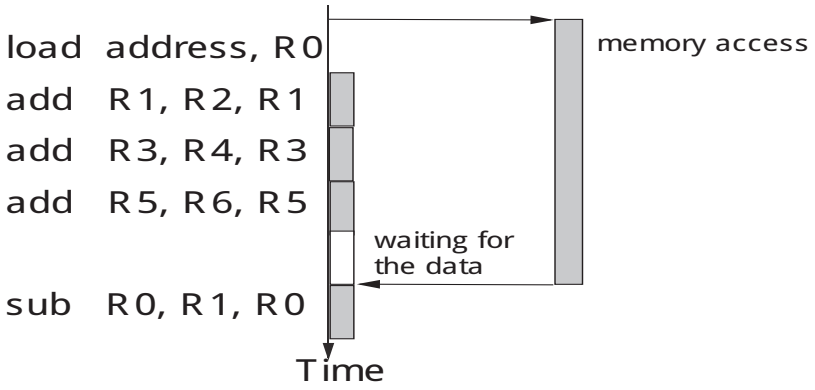
4. Experiments and Results
   Evaluation of preloading on various processors

5. Conclusion

Reducing Memory Overhead by Preloading

- Non-blocking cache is assumed.
- The compiler schedules load instructions early enough from instructions that use the loaded data.

Ex. Exection of a lockup-free cache



Reducing Memory Overhead by Preloading

1. What types of load instructions should be preloaded ?
   If we try to schedule all load instructions early,
   a lot of spill code may be produced.
   → Cache Miss Heuristics

2. How does a compiler schedule them beyond basic blocks ?
   → Global Instruction Scheduling
   □ □ Speculative / Non-speculative

Cache Miss Heuristics

(1) List Access: a load instruction whose base address comes from another load instruction's output.

(2) Stride Access: a load instruction which is in a loop with an address increment.

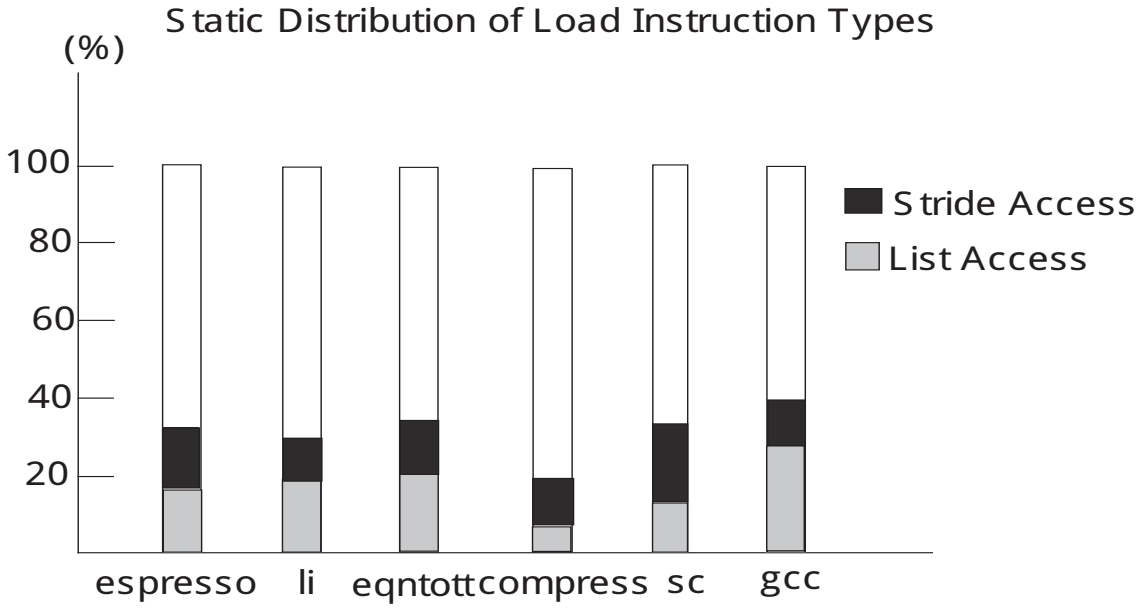
Example of List Access and Stride Access

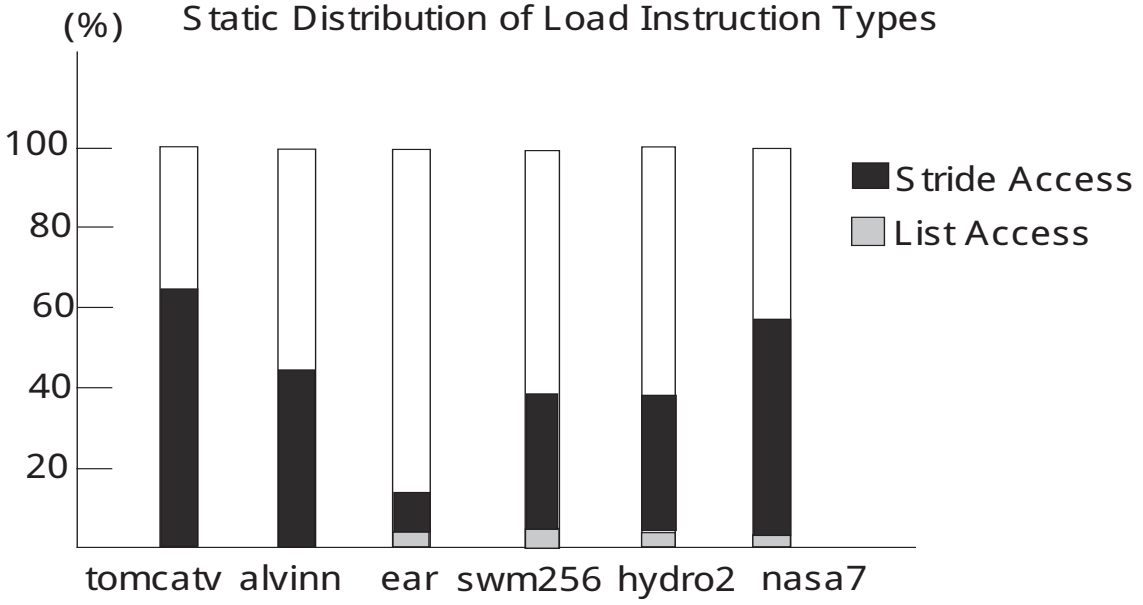
(1) List Access

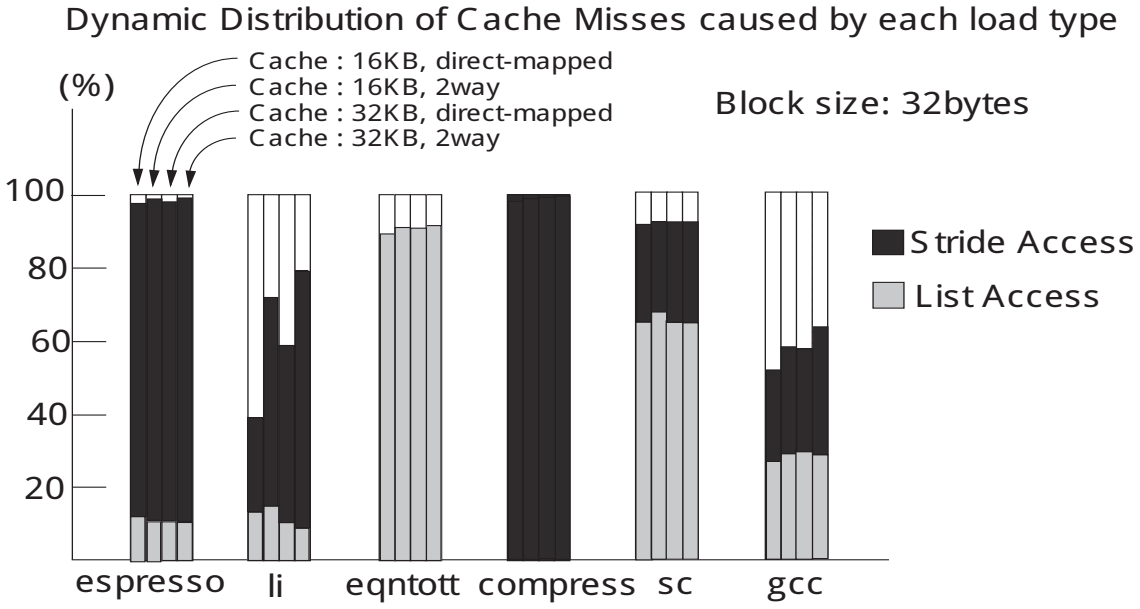
```
load address, R0
    :
load [R0+offset], R1
```

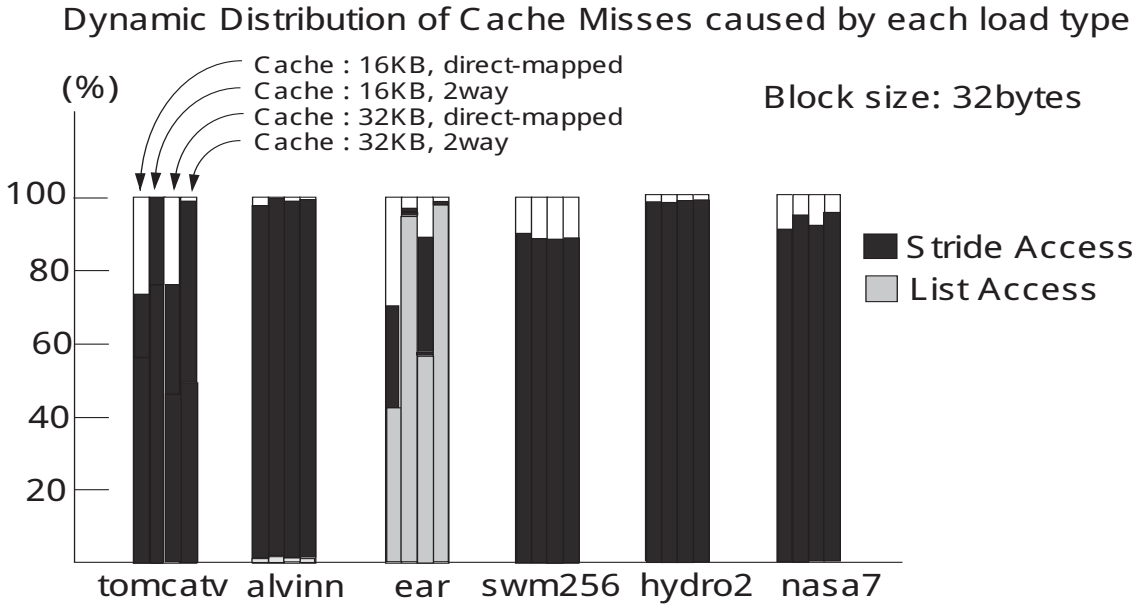
(2) Stride Access

```
L1: add R1, 64, R1
    load [R1+offset], R2
        :
    branch  L1
```


Static Distribution of Load Instruction Types
(%)
100
80
60
40
20
Stride Access
List Access
espresso
li
eqntott
compress
sc
gcc


Static Distribution of Load Instruction Types
(%)
100
80
60
40
20
Stride Access
List Access
tomcatv
alvinn
ear
swm256
hydro2
nasa7


Dynamic Distribution of Cache Misses caused by each load type
(%)
Cache : 16KB, direct-mapped
Cache : 16KB, 2way
Cache : 32KB, direct-mapped
Cache : 32KB, 2way
Block size: 32bytes
100
80
60
40
20
Stride Access
List Access
espresso
li
eqntott
compress
sc
gcc


Dynamic Distribution of Cache Misses caused by each load type
(%)
Cache : 16KB, direct-mapped
Cache : 16KB, 2way
Cache : 32KB, direct-mapped
Cache : 32KB, 2way
Block size: 32bytes
100
80
60
40
20
Stride Access
List Access
tomcatv
alvinn
ear
swm256
hydro2
nasa7

Algorithm of Preloading

Select list access and stride access load instructions

do {

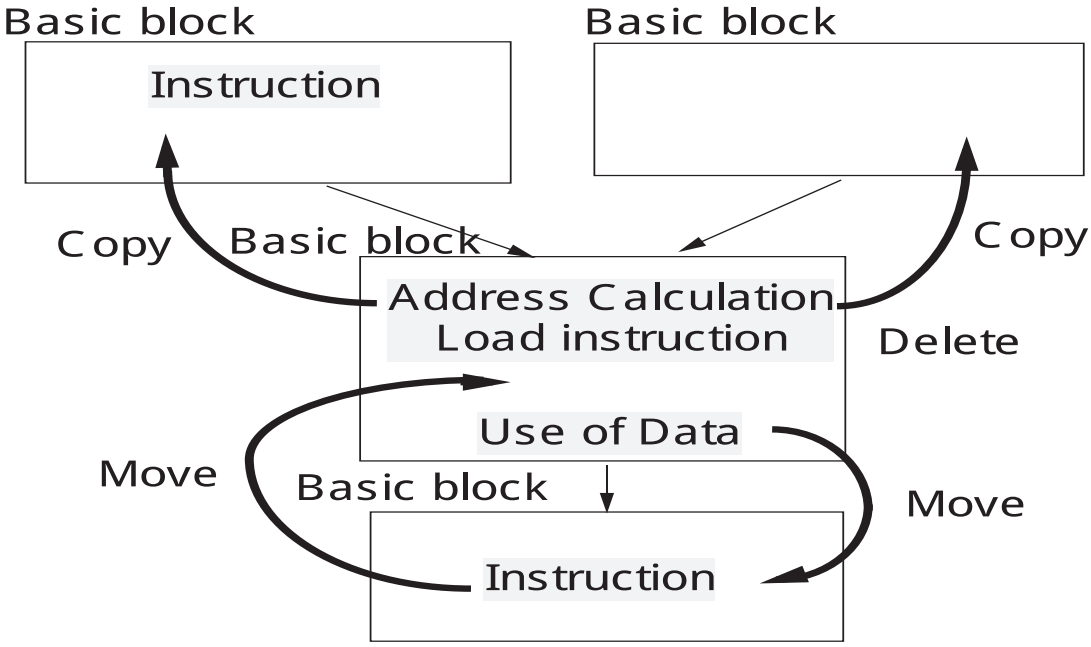
Hosit the selected load instructions

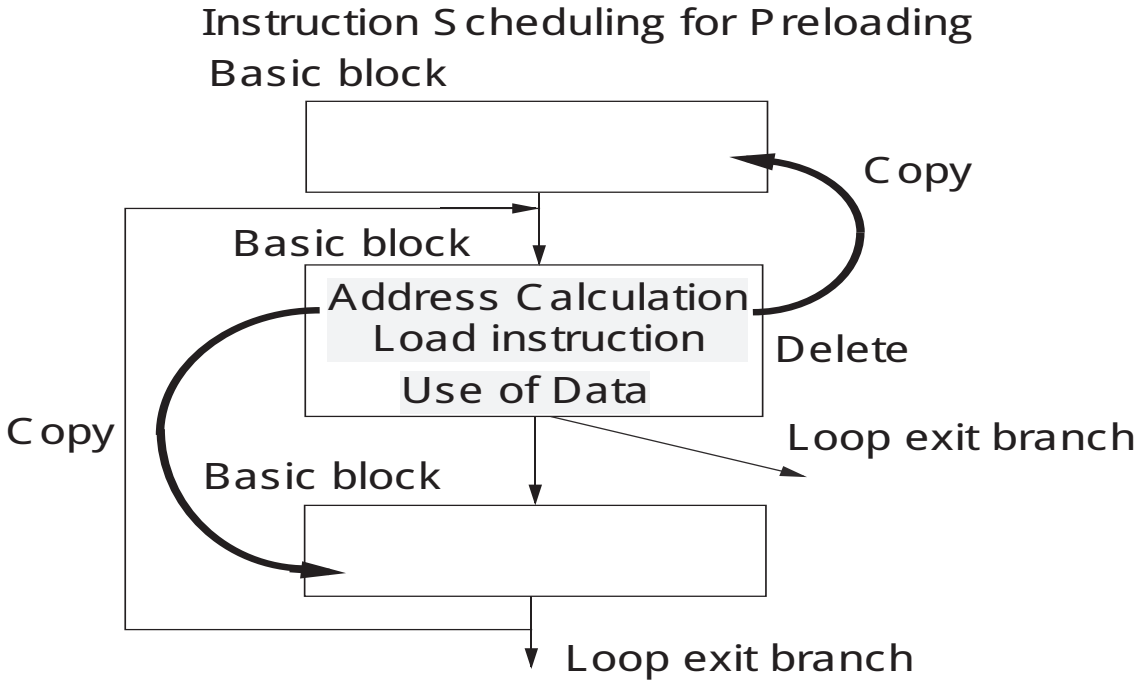
Move down instructions which use the loaded data

Move instructions between the selected load instruction and the instructions that use the loaded data

} while (some instructions are moved beyond basic blocks)

Instruction Scheduling for Preloading





Experiment Environments

Preloading algorithm was implemented on top of Gcc

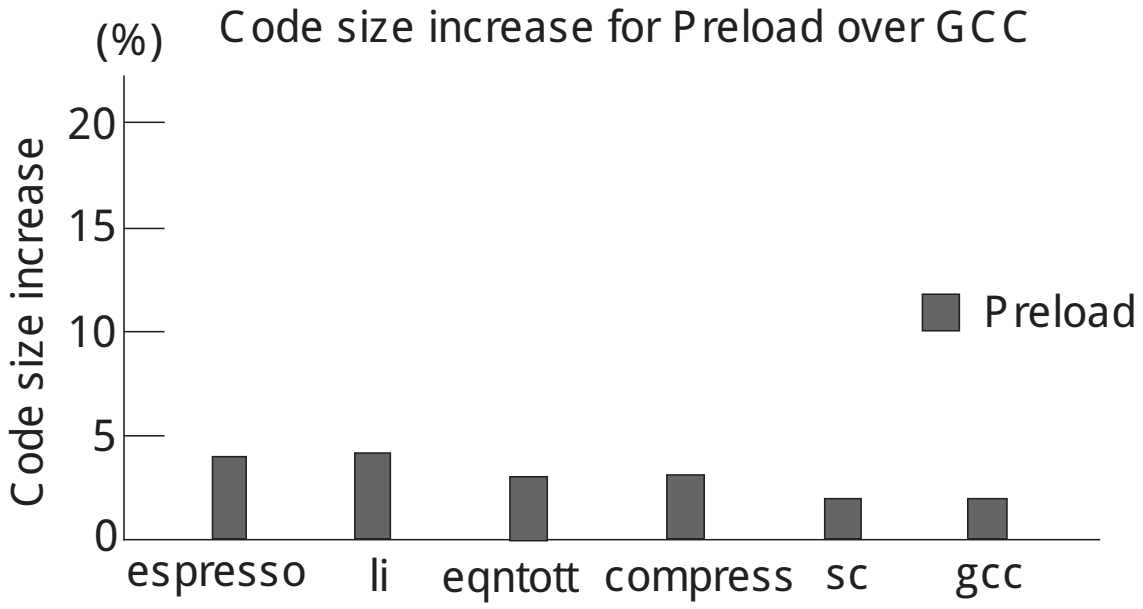
Evaluated on a trace-driven simulator

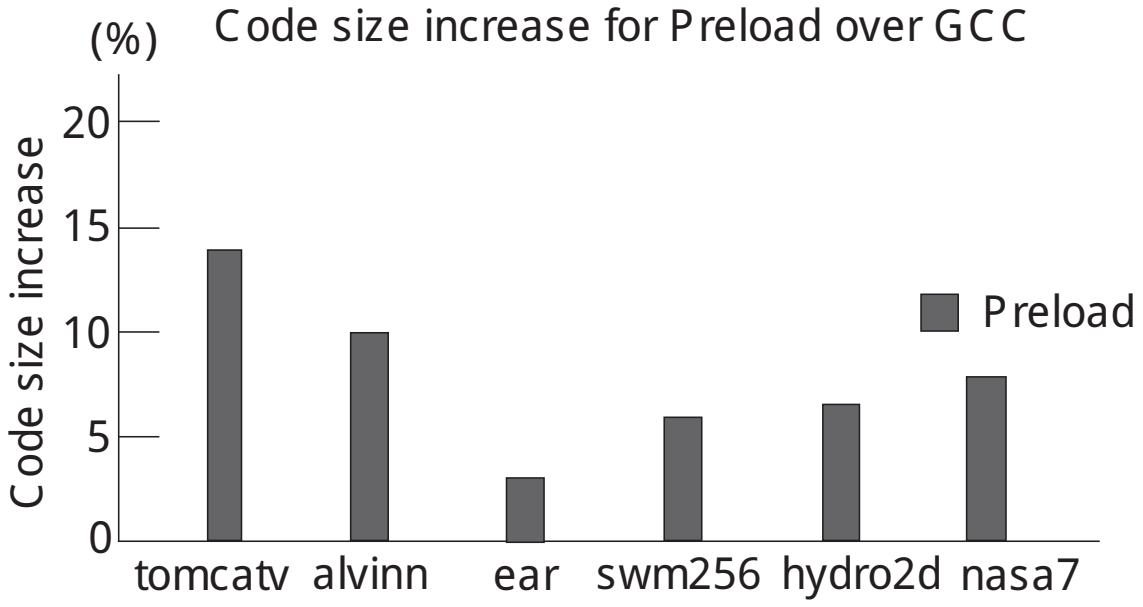
Simulation Model

Processor: SPARC
1 fetch, 1 issue, 4-stage pipeline, in-order execution

Instruction cache: ideal

Data cache: lockup-free, accepts a request every cycle.
The number of incompleted requests is unlimited.
size: 32 Kbytes
latency: 10 ~ 40 cycles


Code size increase for Preload over GCC
(%)
Code size increase
20
15
10
5
0
Preload
espresso
li
eqntott
compress
sc
gcc

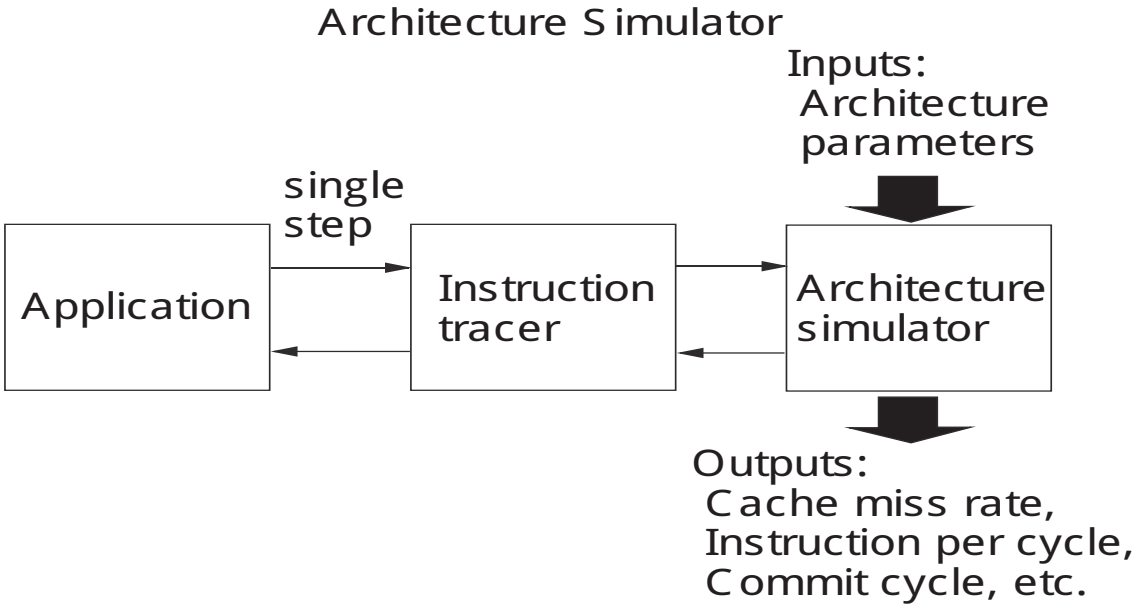

Code size increase for Preload over GCC
(%)
Code size increase
20
15
10
5
0
Preload
tomcatv
alvinn
ear
swm256
hydro2d
nasa7

Number of Spilled Virtual Registers

Program	GCC	Preload
espresso	38	48
li	15	16
eqntott	0	0
compress	0	0
sc	42	41
gcc	108	139

Number of Spilled Virtual Registers

Program	GCC	Preload
tomcatv	0	0
alvinn	0	0
ear	35	46
swm256	8	9
hydro2d	6	9
nasa7	59	79

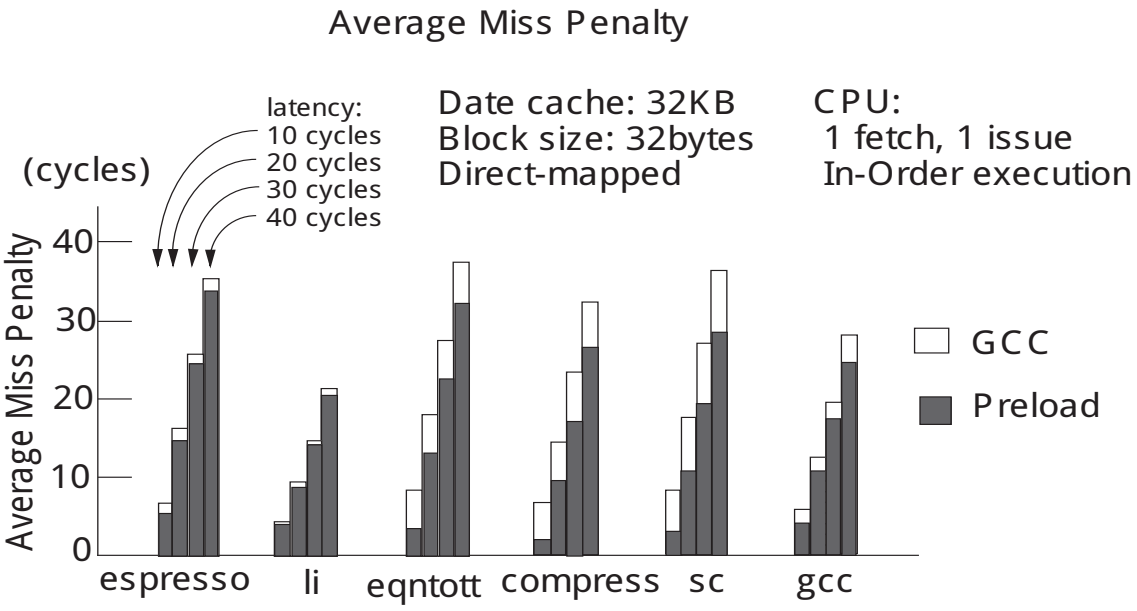


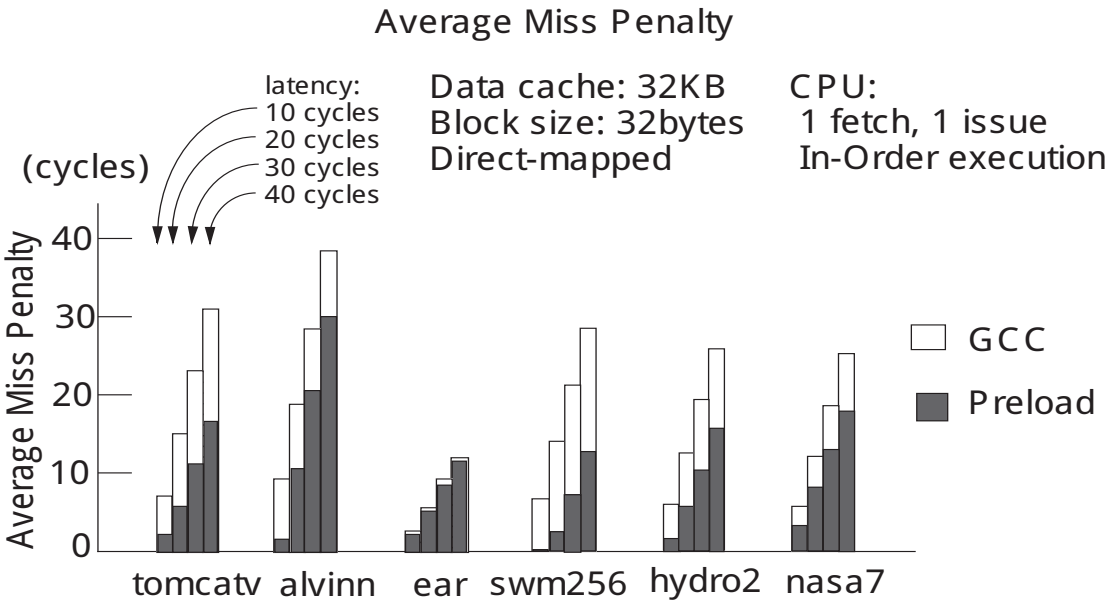
Average Miss Penalty (AMP)

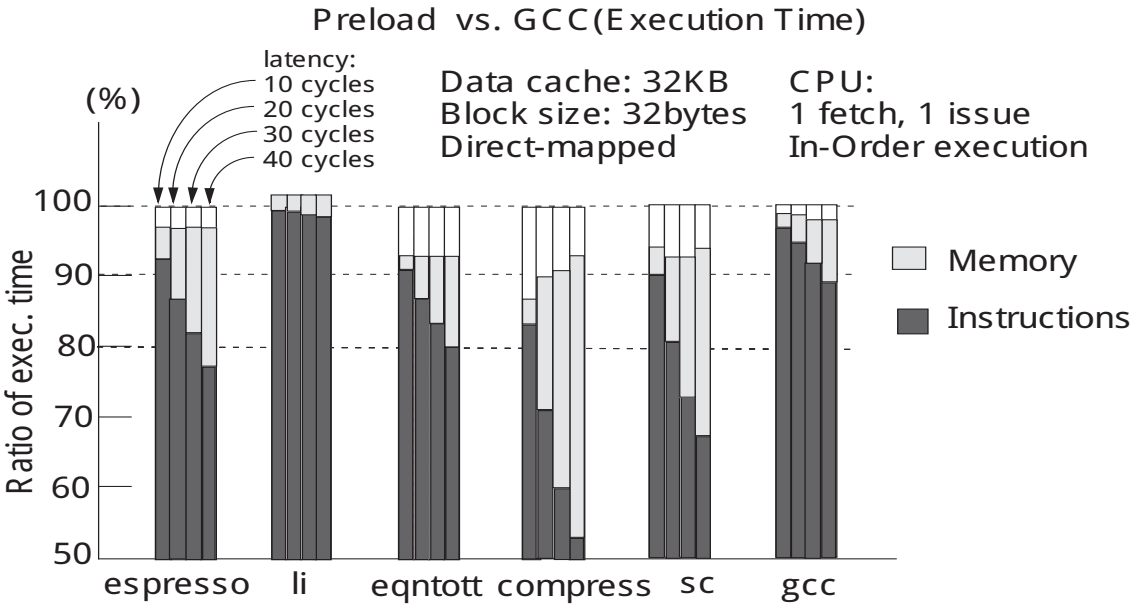
$$AMP = \frac{EC[X] - EC[ideal]}{CM[X]}$$

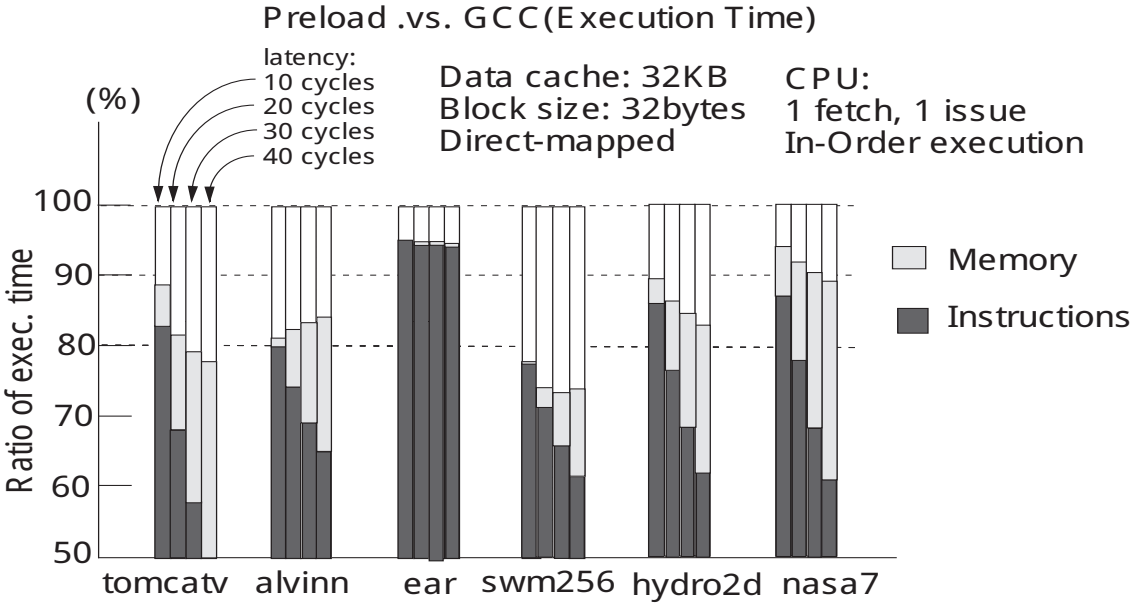
EC [X] : execution cycles
□ □ □ where cache miss latency is X cycles.

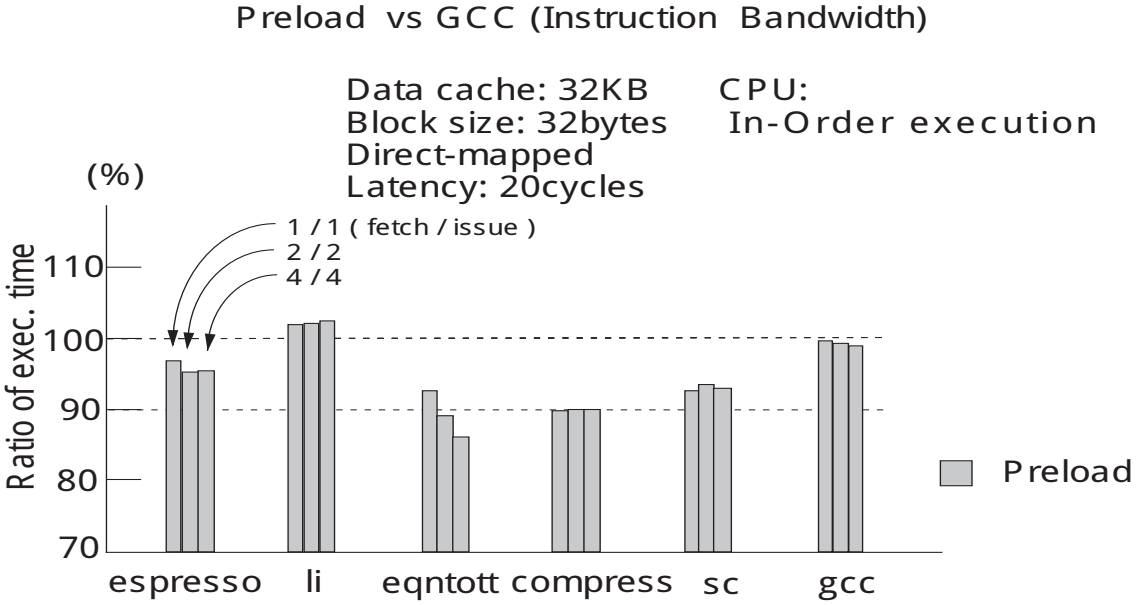
CM [X] : the number of cache missess
□ □ □ where cache miss latency is X cycles.

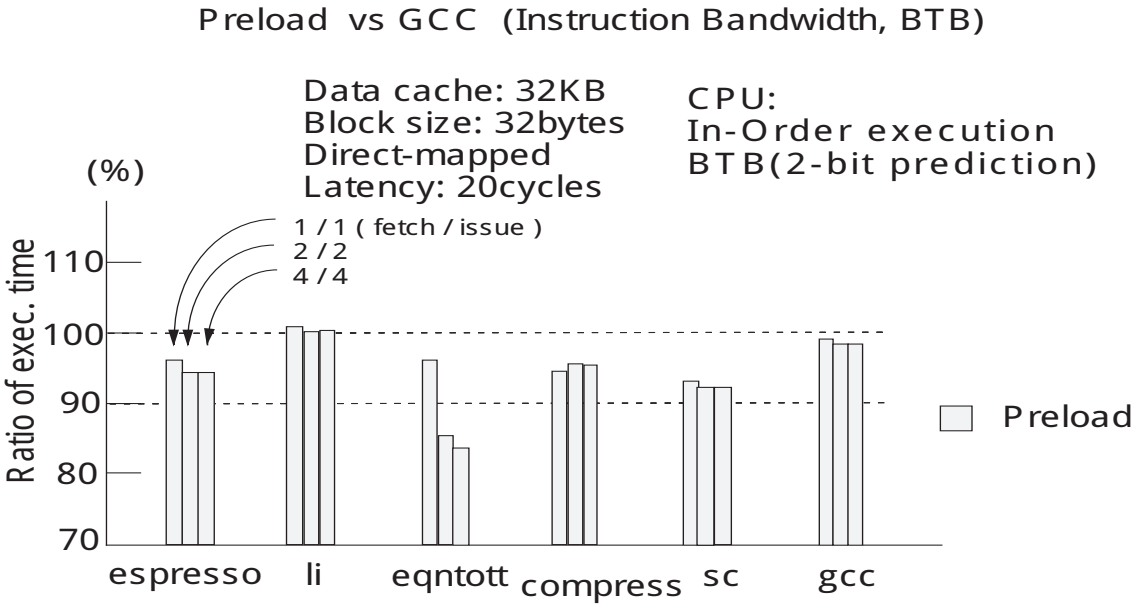

Average Miss Penalty
latency:
10 cycles
20 cycles
30 cycles
40 cycles
Date cache: 32KB
Block size: 32bytes
Direct-mapped
CPU:
1 fetch, 1 issue
In-Order execution
(cycles)
Average Miss Penalty
0
10
20
30
40
GCC
Preload
espresso
li
eqntott
compress
sc
gcc

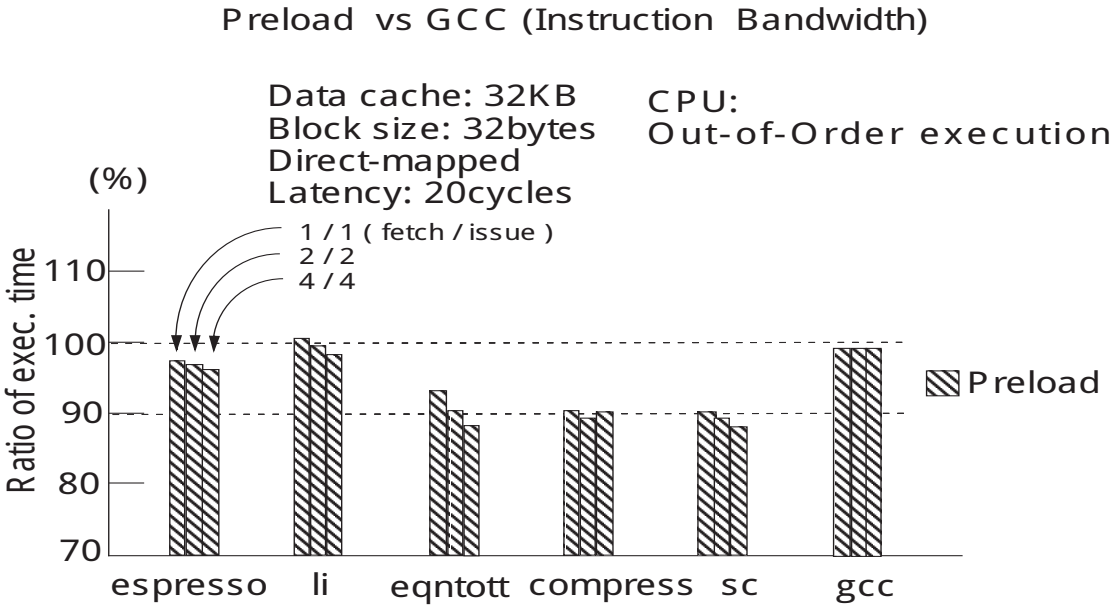

Average Miss Penalty
latency:
10 cycles
20 cycles
30 cycles
40 cycles
Data cache: 32KB
Block size: 32bytes
Direct-mapped
CPU:
1 fetch, 1 issue
In-Order execution
(cycles)
Average Miss Penalty
0
10
20
30
40
GCC
Preload
tomcatv
alvinn
ear
swm256
hydro2
nasa7

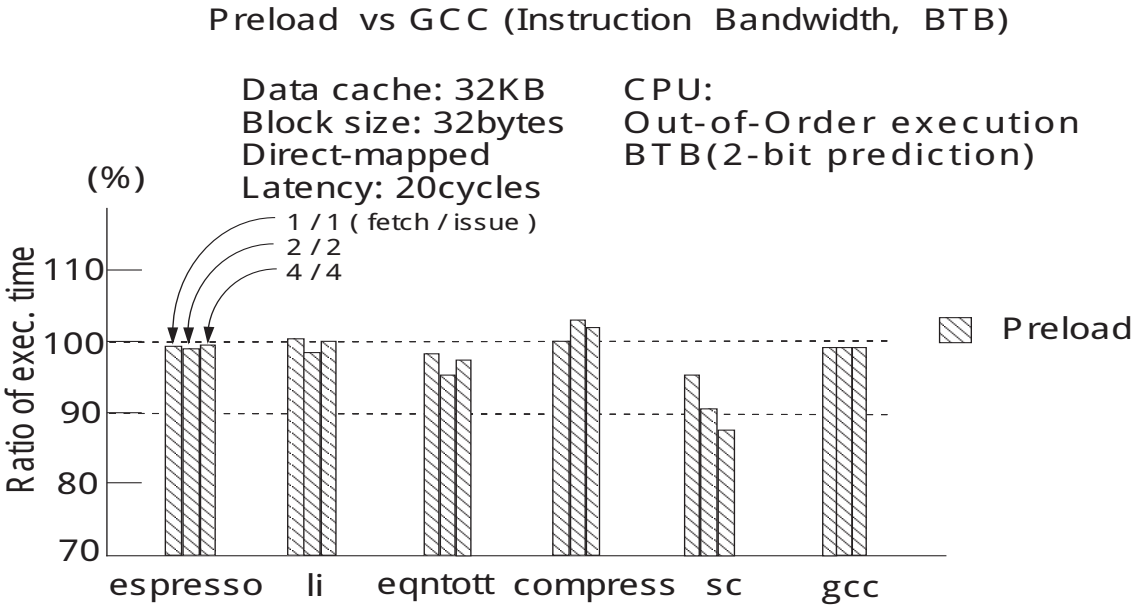

Preload vs. GCC(Execution Time)
latency:
10 cycles
20 cycles
30 cycles
40 cycles
Data cache: 32KB
Block size: 32bytes
Direct-mapped
CPU:
1 fetch, 1 issue
In-Order execution
(%)
Ratio of exec. time
100
90
80
70
60
50
Memory
Instructions
espresso
li
eqntott
compress
sc
gcc


Preload .vs. GCC(Execution Time)
latency:
10 cycles
20 cycles
30 cycles
40 cycles
Data cache: 32KB
Block size: 32bytes
Direct-mapped
CPU:
1 fetch, 1 issue
In-Order execution
(%)
Ratio of exec. time
100
90
80
70
60
50
Memory
Instructions
tomcatv
alvinn
ear
swm256
hydro2d
nasa7


Preload vs GCC (Instruction Bandwidth)
Data cache: 32KB
Block size: 32bytes
Direct-mapped
Latency: 20cycles
CPU:
In-Order execution
(%)
Ratio of exec. time
110
100
90
80
70
1 / 1 (fetch / issue)
2 / 2
4 / 4
espresso
li
eqntott
compress
sc
gcc
Preload


Preload vs GCC (Instruction Bandwidth, BTB)
Data cache: 32KB
Block size: 32bytes
Direct-mapped
Latency: 20cycles
CPU:
In-Order execution
BTB(2-bit prediction)
(%)
Ratio of exec. time
110
100
90
80
70
1 / 1 (fetch / issue)
2 / 2
4 / 4
espresso
li
eqntott
compress
sc
gcc
Preload


Preload vs GCC (Instruction Bandwidth)
Data cache: 32KB
Block size: 32bytes
Direct-mapped
Latency: 20cycles
CPU:
Out-of-Order execution
(%)
Ratio of exec. time
110
100
90
80
70
1 / 1 (fetch / issue)
2 / 2
4 / 4
Preload
espresso
li
eqntott
compress
sc
gcc


Preload vs GCC (Instruction Bandwidth, BTB)
Data cache: 32KB
Block size: 32bytes
Direct-mapped
Latency: 20cycles
CPU:
Out-of-Order execution
BTB(2-bit prediction)
(%)
Ratio of exec. time
110
100
90
80
70
1 / 1 (fetch / issue)
2 / 2
4 / 4
Preload
espresso
li
eqntott
compress
sc
gcc

Conclusion

- Cache Miss Heuristics
  - List Access / Stride Access
- Global Instruction Scheduling on top of GCC
- 5 ~ 27 % Speedup when compared with GCC
- Effective on In-Order and Out-of-Order execution
- Out-of-Order and BTB may reduce
  - the effectiveness of this scheme.