Research Methods in Advanced Computer Architectures

P R O J E C T S

1. Statistics regarding load value locality. It is recommended to use a list which will be indexed with the PC. Each entry of the list will contain the history of values of a certain load instruction. You might vary the history (1, 2, 4, 8).
2. Statistics regarding load value locality. It is recommended to use a list which will be indexed with the memory address. Each entry of the list will contain the history of values of a certain memory location. You might vary the history (1, 2, 4, 8).
3. Direct mapped Load Value Predictor with 4-state confidence automata. The table will be indexed with the PC. It is recommended to use an array. You might implement a Last Value Predictor which will be accessed only in the case of miss in the L1 data cache. You'll vary the number of locations (256, 512, 1024, 2048).
4. Set-associative Load Value Predictor with 1024 entries and 4-state confidence automata. The table will be indexed with the PC. You might implement a Last Value Predictor which will be accessed only in the case of miss in the L1 data cache. It is recommended to use an array. You'll vary the associativity degree (1, 2, 4, 8). The evacuations will be solved by a LRU algorithm.
5. Direct mapped Load Value Predictor with 1024 entries. The table will be indexed with the PC. You might implement a Last Value Predictor which will be accessed only in the case of miss in the L1 data cache. It is recommended to use an array. The confidence automaton will be varied.
6. Direct mapped Load Value Predictor with 1024 entries and multiple values (and confidences). The table will be indexed with the PC. You might implement a Last Value Predictor which will be accessed only in the case of miss in the L1 data cache. It is recommended to use an array. The number of values (and confidences) will be varied as follows: 1, 2, 4, 8.
7. Direct mapped Load Value Predictor with 1024 entries and 4-state confidence automata. The table will be indexed with the PC. You might implement a Last Value Predictor. It is recommended to use an array. You will compare an access only in the case of miss in the L1 data cache with an access only in the case of miss the L2 data cache.
8. Statistics regarding the reusability of MUL/DIV instructions. It is recommended to use a list. You have to implement and compare two strategies:
    a. A certain MUL/DIV instruction can occur only once (with the last "seen" operand values) within the list;
    b. A certain MUL/DIV instruction occurs for each different set of operand values.
9. Direct mapped Reuse Buffer. It is recommended to use an array. You'll vary the number of locations (256, 512, 1024, 2048).
10. Set-associative Reuse Buffer with 1024 entries. It is recommended to use an array. You'll vary the associativity degree (1, 2, 4, 8). The evacuations will be solved by a LRU algorithm.

11. Direct mapped Load Value Predictor with 1024 entries and 4-state confidence automata. You might implement a Last Value Predictor which will be accessed only in the case of miss in the L1 data cache. It is recommended to use an array. Comparative study regarding the access with the PC and an access with the memory address.
12. Direct mapped Reuse Buffer with 1024 entries. It is recommended to use an array. Comparison between the S_V and S_N schemes (see the work of Sodani & Sohi).
13. Direct mapped Reuse Buffer with 1024 entries enhanced with detection of trivial operations. It is recommended to use an array for the Reuse Buffer.
14. Direct mapped Reuse Buffer for MUL/DIV and LOAD instructions. It is recommended to use an array. You'll vary the number of entries (256, 512, 1024, 2048).
15. Direct mapped Load Value Predictor with 1024 entries and 4-state confidence automata. You might integrate into a SMT architecture a Last Value Predictor which will be accessed only in the case of miss in the L1 data cache. It is recommended to use an array. You'll vary the number of threads (1, 2, 3, 4).
16. Direct mapped Reuse Buffer with 1024 entries integrated into a SMT architecture. It is recommended to use an array. You'll vary the number of threads (1, 2, 3, 4).
17. Thermal analysis of microarchitectures with Selective Load Value Predictor.
18. Thermal Manager for a microarchitecture with Dynamic Instruction Reuse.
19. Thermal Manager for a microarchitecture with Selective Load Value Prediction.
20. Including thermal evaluation as a third objective (after performance and energy consumption) into an automatic design space exploration.
21. Evaluation of prediction techniques for loads management based on photovoltaics and energy storage systems.
22. Next location prediction in an intelligent ubiquitous computing ambient using Markov models.
23. Next location prediction in an intelligent ubiquitous computing ambient using neural networks.
24. Next location prediction in an intelligent ubiquitous computing ambient using Hidden Markov Models.
25. Integrating a hybrid branch predictor into the M-Sim.
26. Developing efficient predictors for unbiased branches.

INDICATIONS (for 1-20)

Installing M-Sim

1. Create your own folder which must be visible from the Linux Terminal.
2. Copy into that folder m-sim_v2.0.tar and spec2000binaries.zip

3. Uncompress the m-sim_v2.0 archive using the following command:

   tar –xvf m-sim_v2.0.tar
4. Unzip the spec2000binaries.zip archive.
5. Open a Terminal in Linux and compile M-Sim using the following command:

   make
6. If you get an error in machine.h on line 224, solve it by moving the instructions between lines 229-237 before line 223.
7. Copy the contents of spec2000bats into the m-sim_v2.0 folder.
8. Try to run the simulator through the following command:

   ./run_all.sh
9. Create a results folder in m-sim_v2.0
10. If you receive the "permission denied" message the following steps must be performed:

    10.1. If after the ls –l command the sh files are not seen as applications apply:

    chmod +x *.sh

    10.2. From all the sh files, including run_all.sh, you have to remove the cd/home/… commands.
11. Try again to start the simulation by using the following command:

    ./run_all.sh

After any change of the source code it is necessary to apply the following commands:

make clean

make

Defining the structure and the functions

The structure and the functions must be defined at the beginning of the sim-outorder.c source file.

Dynamic Instruction Reuse

For dynamic instruction reuse the Reuse Buffer (RB) structure is:

```
struct RBLocation
{
        md_addr_t tag;          // the higher part of the buffered instruction's PC
        qword_t srcval1;        // first source value
        qword_t srcval2;        // second source value
        qword_t res;            // result value
};
```

For the projects which will use lists the structure will have as additional field the address of the next node.

The reuse buffer in a multithreaded architecture can be declared as follows:

```
struct RBLocation rbuff[10][10000];  // maximum 10 threads and 10000 entries per thread
```

The declaration for a monothreaded architecture is:

```
struct RBLocation rbuff[10000];        // maximum 10000 entries
```

The function(s) can be implemented based on the pseudocode given below:

```
Foreach MUL or DIV
        nMulDiv++
        If MISS in the RB (TAG not found)
                Evacuation using LRU (only in case of set-associative table)
                Update the (evacuated) entry with the actual TAG, operand values and result
        If HIT in the RB (TAG found)
                If at least one operand value is different
                        Update operand values and result
                If the operand values are the same
                        nReusedMulDiv++
Reuse_Degree = (nReusedMulDiv / nMulDiv) * 100 [%]
```

Parameters:

- Instruction address (PC): rs->PC (type: md_addr_t);
- Operand value 1: rs->val_ra (type: qword_t);

- Operand value 2: rs->val_rb (type: qword_t);
- Thread ID: rs->context_id (type: int);

All the addresses must be shifted to right by 2 before indexation. The tag and index can be computed as follows:

- Index = (Address>>2)%N;
- TAG = (Address>>2)/N;

where N is the number of entries in a direct mapped table or the number of sets in associative tables.

Dynamic Load Value Prediction

For dynamic load value prediction the Load Value Prediction Table (LVPT) structure is the following:

```
struct LVPTLocation
{
        md_addr_t tag;          // the higher part of the Load instruction's PC
        byte_t counter;         // 2-bit saturating counter
        qword_t value;          // Load value
};
```

For the projects which will use lists the structure will have as additional field the address of the next node.

The prediction table in a multithreaded architecture can be declared as follows:

```
struct LVPTLocation lvpt[10][10000];      // maximum 10 threads and 10000 entries per thread
```

The declaration for a monothreaded architecture is:

```
struct LVPTLocation lvpt[10000];      // maximum 10000 entries.
```

The function(s) can be implemented based on the pseudocode given below:

```
Foreach long latency LOAD
        If MISS in the LVPT (TAG not found)
                Evacuation using LRU (only in case of set-associative table)
                Update the (evacuated) entry with the actual TAG, value and confidence=0
        If HIT in the LVPT (TAG found)
                If PREDICTABLE
                        nPredictions++
                        If current_value == LVPT_value
                                nCorrectPredictions++
                                Increase the confidence counter
                        If current_value != LVPT_value
                                Update LVPT_value
                                Decrease the confidence counter
                If UNPREDICTABLE
                        If current_value == LVPT_value
                                Increase the confidence counter
                        If current_value != LVPT_value
                                Update LVPT_value
                                Decrease the confidence counter
Prediction_Accuracy = (nCorrectPredictions / nPredictions) * 100 [%]
```

Parameters:

- Instruction address (PC): rs->PC (type: md_addr_t);
- Memory address: rs->addr (type: md_addr_t);
- Load value: rs->val_ra (type: qword_t);
- Thread ID: rs->context_id (type: int);

All the addresses must be shifted to right by 2 before indexation. The tag and index can be computed as follows:

- Index = (Address>>2)%N;
- TAG = (Address>>2)/N;

where N is the number of entries in a direct mapped table or the number of sets in associative tables.

Where to call the functions and how to select the instructions?

Dynamic Instruction Reuse

In sim-outorder.c, selection function, after rs->queued = FALSE

```
/* selecting integer instructions */
if((MD_OP_FUCLASS(rs->op) != NA) && (MD_OP_FLAGS(rs->op) & F_ICOMP))
{
        switch(rs->op)
        {
                /* integer MUL instructions without immediate operand */
                case UMULH:
                case MULL:
                case MULQ:
                        /* call function(s) to reuse integer MUL without imm. op. */
                        break;
                /* integer MUL instructions with one immediate operand */
                case UMULHI:
                case MULLI:
                case MULQI:
                        /* call function(s) to reuse integer MUL with imm. op. */
                        break;
        }
}
/* selecting floating point instructions */
if((MD_OP_FUCLASS(rs->op) != NA) && (MD_OP_FLAGS(rs->op) & F_FCOMP))
{
        switch(rs->op)
        {
                /* floating point MUL instructions without immediate operand */
                case MULS:
                case MULT:
                        /* call function(s) to reuse floating point MUL without imm. op. */
                        break;
                /* floating point DIV instructions without immediate operand */
                case DIVS:
```

```
            case DIVT:
                /* call function(s) to reuse floating point DIV without imm. op. */
                break;
        }
}
```

Selective Load Value Prediction

In sim-outorder.c, selection function, after rs->exec_lat=load_lat

If we access the prediction table only for loads with miss in DL1:

```
if(load_lat>cache_dl1_lat)
{
        /* call function(s) to predict LOAD */
}
```

If we access the prediction table only for loads with miss in DL2:

```
if(load_lat>cache_dl2_lat)
{
        /* call function(s) to predict LOAD */
}
```

How to change the latencies?

Dynamic Instruction Reuse

- Reused MUL/DIV: rs->exec_lat = 1 (cycle)
- Unreused MUL/DIV: rs->exec_lat unchanged

Load Value Prediction

- Correctly predicted LOAD: rs->exec_lat = 3 (cycles)

- Mispredicted LOAD: rs->exec_lat += 7 (cycles)
- Unpredicted LOAD: rs->exec_lat unchanged

Changing the technology, the frequency and the voltage

Technology – 80nm

In power.h

```
#define FUDGEFACTOR 10.0
```

In cacti\def.h

```
#define FUDGEFACTOR 10.0
```

Frequency – 1.2 GHz

In power.h

```
#define MHz 1200e6
```

Voltage – 1V

In power.h

```
#define Vdd (1 * VSCALE)
```

In cacti\def.h

```
#define Vdd 1
```

Command line arguments

Example

```
int parameter = 0; // global declaration in sim-outorder.c
…
sim_reg_options(…
{
        …
```

```
        opt_reg_int(odb, “-parameter_name”, “parameter description”, &parameter, /*default
                value */ 0, TRUE, NULL);
        …
}
```

Printing the results in the *.res file

Example – we print out the global variables a and b and also (a/b)*100

```
static counter_t a = 0; // global declaration in sim-outorder.c
static counter_t b = 0; // global declaration in sim-outorder.c
…
sim_reg_stats(…
{
        …
        stat_reg_counter(sdb, “a”, “description of a”, &a, /*initial value*/ 0, /*format*/ NULL);
        stat_reg_counter(sdb, “b”, “description of b”, &b, /*initial value*/ 0, /*format*/ NULL);
        stat_reg_formula(sdb, “c”, “description of c”, “(a/b)*100”, NULL);
        …
}
```

The structure of the final report

- Cover
- Introduction
- State of the art in the area of your project
- Implementation
- Experimental Results
- Conclusions
- References

The final reports will be submitted on a CD in PDF format no later than the last week of the semester (before the session).