Project Tutorial

COMP4611 Tutorial 7

Oct 29th-Nov 2nd

OUTLINE

- Objectives
- Project Task I
  - Task description
- Project Task II
  - Task description
  - Skeleton code
- Project Task III
  - Task description
- Deliverables
- Submission & Grading

Objectives

- To have hands-on experiments with the branch prediction using SimpleScalar

- To design your own branch predictor for higher prediction accuracy

Background

- Branch predictor types in SimpleScalar (**alpha**)
  - Taken or not-taken
  - Perfect predictor
  - 2-bit predictor
  - 2-level adaptive predictor

- Specifying the branch predictor type
  - Option: *-bpred <type>*
  - *<type>*: *nottaken, taken, perfect, bimod, 2lev*

Background

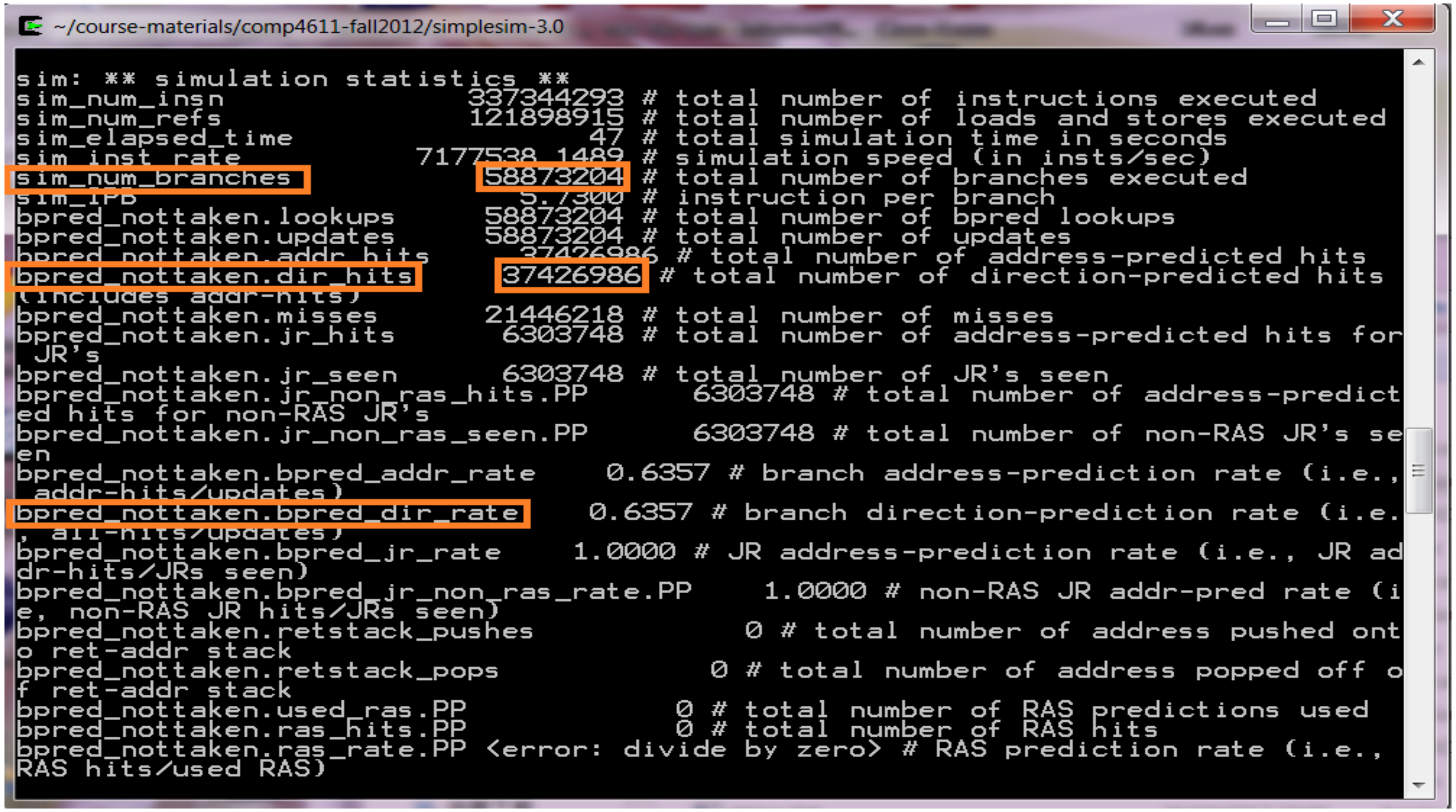
- Configuring taken/nottaken predictor
  - Option: -*bpred taken*, -*bpred nottaken*
  - Command line example for "-*bpred taken*":

```
./sim-bpred -bpred taken benchmarks/cc1.alpha -O
benchmarks/1stmt.1
```

  - Command line example for "-*bpred nottaken*":

```
./sim-bpred -bpred nottaken benchmarks/cc1.alpha
-O benchmarks/1stmt.i
```

SIMULATING NOT-TAKEN PREDICTOR


~/course-materials/comp4611-fall2012/simplesim-3.0

```
sim: ** simulation statistics **
sim_num_insn               337344293 # total number of instructions executed
sim_num_refs               121898915 # total number of loads and stores executed
sim_elapsed_time                  47 # total simulation time in seconds
sim_inst_rate           7177538.1489 # simulation speed (in insts/sec)
sim_num_branches            58873204 # total number of branches executed
sim_IPB                       5.7300 # instruction per branch
bpred_nottaken.lookups      58873204 # total number of bpred lookups
bpred_nottaken.updates      58873204 # total number of updates
bpred_nottaken.addr_hits      37426986 # total number of address-predicted hits
bpred_nottaken.dir_hits       37426986 # total number of direction-predicted hits
 (includes addr-hits)
bpred_nottaken.misses       21446218 # total number of misses
bpred_nottaken.jr_hits       6303748 # total number of address-predicted hits for
 JR's
bpred_nottaken.jr_seen       6303748 # total number of JR's seen
bpred_nottaken.jr_non_ras_hits.PP         6303748 # total number of address-predict
ed hits for non-RAS JR's
bpred_nottaken.jr_non_ras_seen.PP         6303748 # total number of non-RAS JR's se
en
bpred_nottaken.bpred_addr_rate      0.6357 # branch address-prediction rate (i.e.,
 addr-hits/updates)
bpred_nottaken.bpred_dir_rate      0.6357 # branch direction-prediction rate (i.e.
, all-hits/updates)
bpred_nottaken.bpred_jr_rate      1.0000 # JR address-prediction rate (i.e., JR ad
dr-hits/JRs seen)
bpred_nottaken.bpred_jr_non_ras_rate.PP      1.0000 # non-RAS JR addr-pred rate (i
e, non-RAS JR hits/JRs seen)
bpred_nottaken.retstack_pushes              0 # total number of address pushed ont
o ret-addr stack
bpred_nottaken.retstack_pops              0 # total number of address popped off o
f ret-addr stack
bpred_nottaken.used_ras.PP            0 # total number of RAS predictions used
bpred_nottaken.ras_hits.PP            0 # total number of RAS hits
bpred_nottaken.ras_rate.PP <error: divide by zero> # RAS prediction rate (i.e.,
RAS hits/used RAS)
```

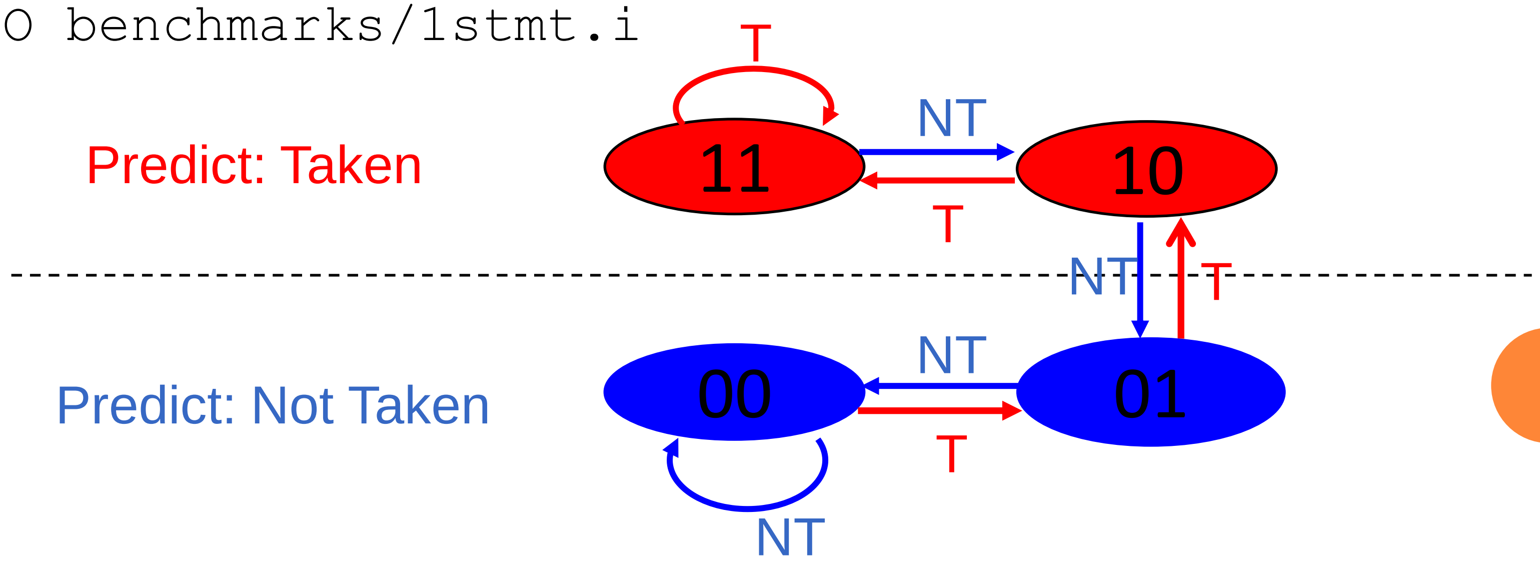
Branch prediction accuracy:

$$\text{bpred\_dir\_rate} = \frac{\text{total number of direction−predicted hits}}{\text{total number of branches executed}}$$

BACKGROUND

- Configuring 2-bit predictor
  - *-bpred:bimod <size>*
  - <*size*>: the size of direct-mapped branch target buffer (BTB)
  - Command line example:

```
./sim-bpred -bpred:bimod 2048 benchmarks/cc1.alpha
-O benchmarks/1stmt.i
```



Project Task I

- Evaluation of 2-bit dynamic branch predictor
  - Varied number of table entries (**512, 1024, 2048**)
  - Benchmark: **Go** (Alpha)
  - Configuration parameters for Go: **2 17**
  - Input file for Go: **2stone9.in**
  - **Branch prediction accuracy** (bpred_dir_rate) and **command lines** to be included in the project report
  - Command line example:

```
./sim-bpred -redir:prog results/go-2bit-512-2-
17.progout -redir:sim results/go-2bit-512-2-
17.simout -bpred:bimod 512 benchmarks/go.alpha
2 17 benchmarks/2stone9.in
```

Project Task II

- Write a 3-bit branch predictor on SimpleScalar
    - Implementation
    - Evaluation

- Guideline
    - Skeleton code at http://course.cse.ust.hk/comp4611/comp4611_project.tar.gz
    - Extract the package and compile it by typing "**make config-alpha**" and then "**make**"
    - Fill in the missing code in ***bpred.c*** and recompile

CODE GLIMPSE

- Source code that implements the branch predictor is in ***simplesim-3.0/***
  - *sim-bpred.c*: simulating a program with configured branch prediction instance
  - *brpred.c*: implementing the logic of several branch predictors
  - *brpred.h*: defining the structure of several branch predictors
  - *main.c*: simulating a program on SimpleScalar

Code Glimpse

- bpred.h
  - *bpred_class*: branch predictor types
  - *bpred_t*: branch predictor structure
  - *bpred_dir_t*: branch direction structure
  - *bpred_update_t*: branch state update structure (containing predictor state counter)
  - *bpred_btb_ent_t*: entry structure in a BTB

Code Glimpse

- Predictor's state counter

```
struct bpred_update_t {
  char *pdir1;          /* direction-1 predictor counter */
  char *pdir2;          /* direction-2 predictor counter */
  char *pmeta;                    /* meta predictor counter */
  struct {              /* predicted directions */
      unsigned int ras    : 1;     /* RAS used */
      unsigned int bimod  : 1;   /* bimodal predictor */
      unsigned int twolev : 1;   /* 2-level predictor */
      unsigned int meta   : 1;   /* meta predictor (0..bimod /
1..2lev) */
  } dir;
};
```

Code Glimpse

- bpred.c
  - *bpred_create*: create a new branch predictor instance
  - *bpred_dir_create*: create a branch direction instance
  - *bpred_lookup*: predict a branch target
  - *bpred_dir_lookup*: predict a branch direction
  - *bpred_update*: update an entry in BTB

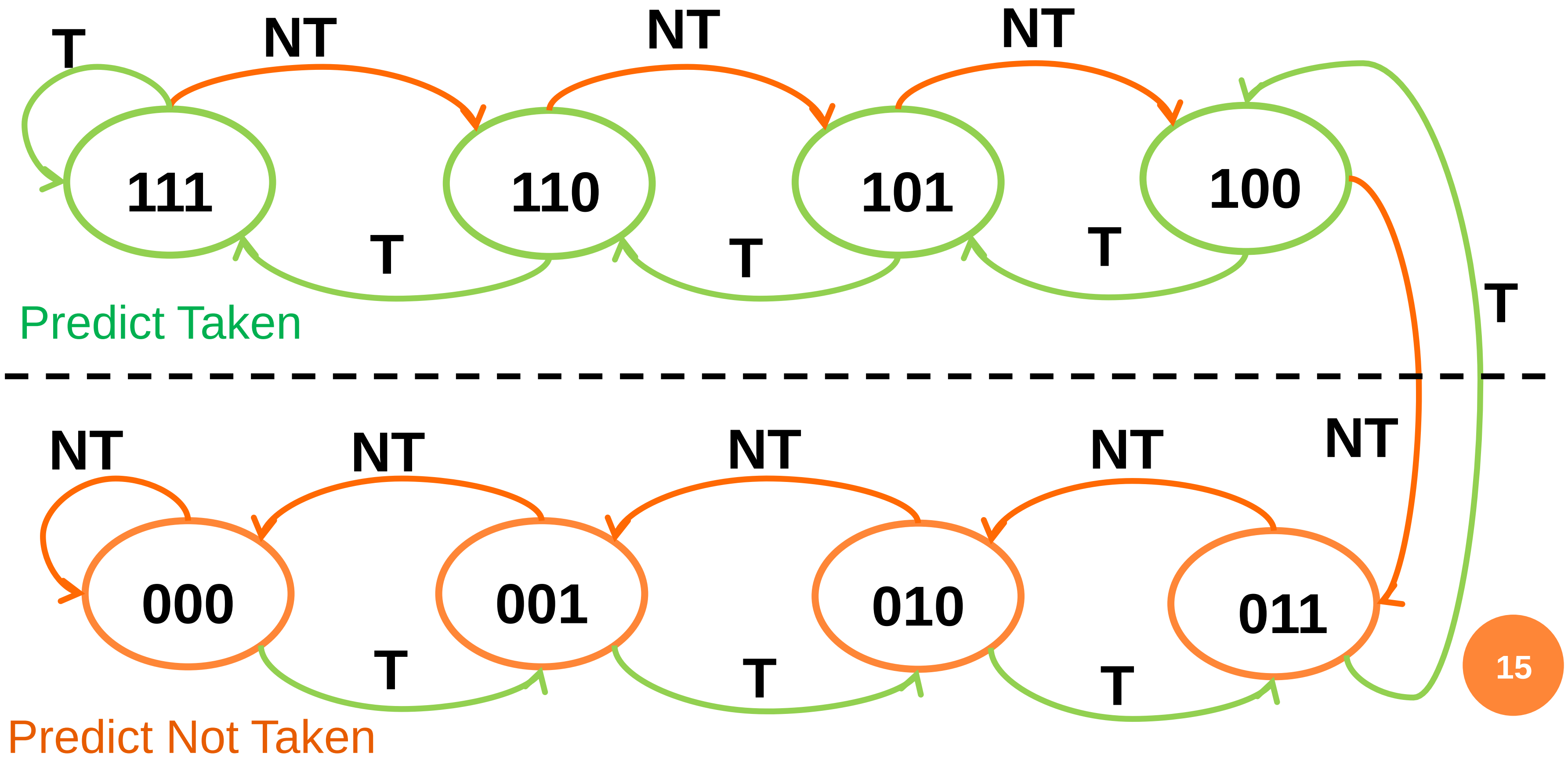
Code Glimpse

- Workflow of branch prediction

```
main
    sim_check_options
        bpred_create
            bpred_dir_create
            allocate BTB
    sim_main
        bpred_lookup
        bpred_update
            pred->dir_hits
            *dir_update_ptr->pdir1
```

3-bit Branch Predictor

A 3-bit branch predictor has 8 states in total



IMPLEMENTATION

- Configuring 3-bit predictor
  - *-bpred:tribit <size>*
  - <*size*>: the size of direct-mapped BTB
  - Command line example:

```
./sim-bpred -v -redir:prog results/go-3bit-
2048-2-17.progout -redir:sim results/go-3bit-
2048-2-17.simout -bpred:tribit 2048
benchmarks/go.alpha 2 17
benchmarks/5stone21.in
```

  - Command must include verbose option –v

Skeleton Code

- *branch_lookup()* in bpred.c

```
/* comp4611 3-bit predict saturating cntr pred (dir mapped) */
 if (pbtb == NULL)  {
    if (pred->class != BPred3bit) {
       return ((*(dir_update_ptr->pdir1) >= 2)? /* taken */ 1 : /* not taken */
0);
    }
    else {
       // code to be filled in here
    }
 }
 else {
    ……
 }
 /**********************************************************/
```

Skeleton Code

- *branch_update()* in bpred.c

```
/* comp4611 3-bit predict saturating cntr pred (dir mapped) */
 if (dir_update_ptr->pdir1)  {
     if (pred->class != BPred3bit)  {
           .......
      }
      else  {
            if (taken)  {
                 // code to be filled in here
            }
            else   { /* not taken */
                // code to be filled in here
            }
     }
 }
/*******************************************************/
```

Evaluation

- 3-bit predictor with the table size as **2048**
  - Benchmark: **Go** (Alpha)
  - Parameters for Go: **2 17**
  - Input file for Go: **2stone9.in**
  - **Branch prediction accuracy** and **command line** to be included in the project report
  - Output trace files (include option **–v**)
    - are the redirected program and simulation output
    - should be saved in the "results" directory
    - are as large as **a few GBs** and make sure you have sufficient disk storage for them in your PC

Project Task III

- Design and implement your own predictor
  - Use existing predictors (e.g. 2-level) or create your own predictor to achieve higher accuracy than the 2-bit predictor
  - Evaluate your predictor using **Go** (Alpha)
  - Parameters for Go: **2 17**
  - Input file for Go: **2stone9.in**
  - **Branch prediction accuracy** and **command line** to be included in the project report
  - Output trace files (include option **–v**)
    - are the redirected program and simulation output
    - should be saved in the "results" directory
    - are as large as **a few GBs** and make sure you have sufficient disk storage for them in your PC

DELIVERABLES

- Source code
  - Code for 3-bit predictor: ***bpred.c***, saved as "**3bit/bpred.c**"
  - Code for your own predictor: including ***bpred.h***, ***bpred.c***, ***sim-bpred.c, readme*** (specify your command line format) and **other relevant files**, saved under "**own/**"
  - To be submitted to CASS

- Output trace files
  - Output trace files for **3-bit predictor**
  - Output trace files for **your own predictor**
  - To be submitted separately (**not CASS**)

- Project report (no longer than 2 pages)
  - Evaluation result (**2-bit, 3-bit, your own predictor**)
  - Description of **your own predictor**
  - To be submitted to **CASS**

Grading Scheme

- 2-bit predictor (20%)
  - Correctness
- 3-bit predictor (40%)
  - Correctness
- Your own predictor (30%)
  - If correct, score = **max{0, (prediction accuracy – 0.85) * 200}**
- Project report (10%)
  - Completeness
  - Correctness
  - Clarity

Submission Guideline

- Submit your source code and report to CASS
  - Report should contain your **group ID**, each group member's **name**, **UST ID**, **email** on the first page
  - Package the code and report files in one zip file as "**comp4611_project_*groupID*.zip**"
  - Deadline: **Nov 30, 2012**
- Submit the hardcopy of your report to the homework box
  - Report should contain your **group ID**, each group member's **name**, **UST ID**, **email** on the first page
  - Deadline: **Nov 30, 2012**
- Submission of your output trace files will be informed later

References

- SimpleScalar LLC: www.simplescalar.com
- Introduction to SimpleScalar: www.ecs.umass.edu/ece/koren/architecture/Simplescalar/SimpleScalar_introduction.htm
- SimpleScalar Tool Set: http://www.ece.uah.edu/~lacasa/tutorials/ss/ss.htm

Appendix

- Branch direction structure

```
struct bpred_dir_t {
  enum bpred_class class;      /* type of predictor */
  union {
     struct {
        unsigned int size;         /* number of entries in direct-
mapped table */
        unsigned char *table;   /* prediction state table */
     } bimod;
     ……
  } config;
};
```

Appendix

- sim-bpred.c
  - *sim_main*: execute each conditional branch instruction
  - *sim_reg_options*: register command options
  - *sim_check_options*: determine branch predictor type and create a branch predictor instance
  - *pred_type*: define the type of branch predictor
  - **_nelt & *_config[]*: configure the parameters for each branch predictor

APPENDIX

- *sim_main()* in sim-bpred.c

```
if (MD_OP_FLAGS(op) & F_CTRL)  {
    if (pred) {
        pred_PC = bpred_lookup(pred, …, &update_rec, …);
        if (!pred_PC)  {
            pred_PC = regs.regs_PC + sizeof(md_inst_t);
        }
        bpred_update(pred, …, &update_rec, …);
    }
  }
  ……
  regs.regs_PC = regs.regs_NPC;
  regs.regs_NPC += sizeof(md_inst_t);
```

Appendix

- *main()* in main.c

```
sim_odb = opt_new(orphan_fn);
opt_reg_flag(sim_odb, …);
……
sim_reg_options(sim_odb);
opt_process_options(sim_odb, argc, argv);
……
sim_check_options(sim_odb, argc, argv);
……
sim_reg_stats(sim_sdb);
……
sim_main();
……
```