THE UNIVERSITY OF TEXAS AT DALLAS
Erik Jonsson School of Engineering and Computer Science

Project 1 – Branch Prediction

CS6304 Computer Architecture

Chih-An Chang , William Chang,
CXC210017, CXC200006
Department of Computer Science

Outline

1. Branch Prediction Introduction & Background
2. Gem5 Branch Predictors
   - Introduction
   - Setup
   - Configuration
3. Gem5 Branch Predictors Results
4. Benchmark Results
5. Result Discussion

Branch Prediction
Introduction & Background

Overview

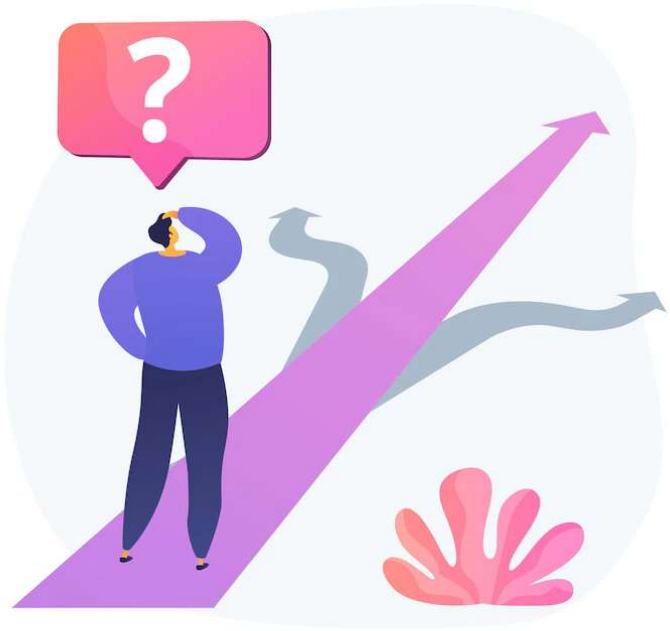
What is brach prediction?

Why it is important?

Branch Prediction

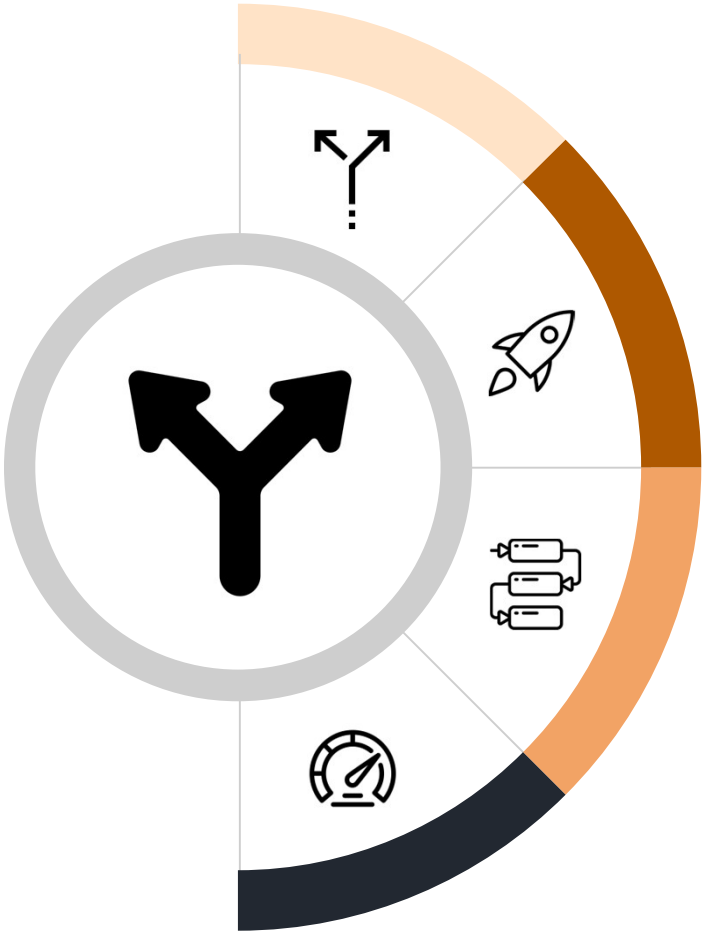
Guess the next fetch address to be used in the next cycle

3 Things to be predict at Fetch stage



- Fetched instruction is a branch ?
- (Conditional) Branch Direction
- Branch target address (if taken)

Why it so important?



01

To handle control dependencies

02

To speed execution of instructions on processors that use pipelining

03

To keep pipeline full of correct sequence of dynamic instructions

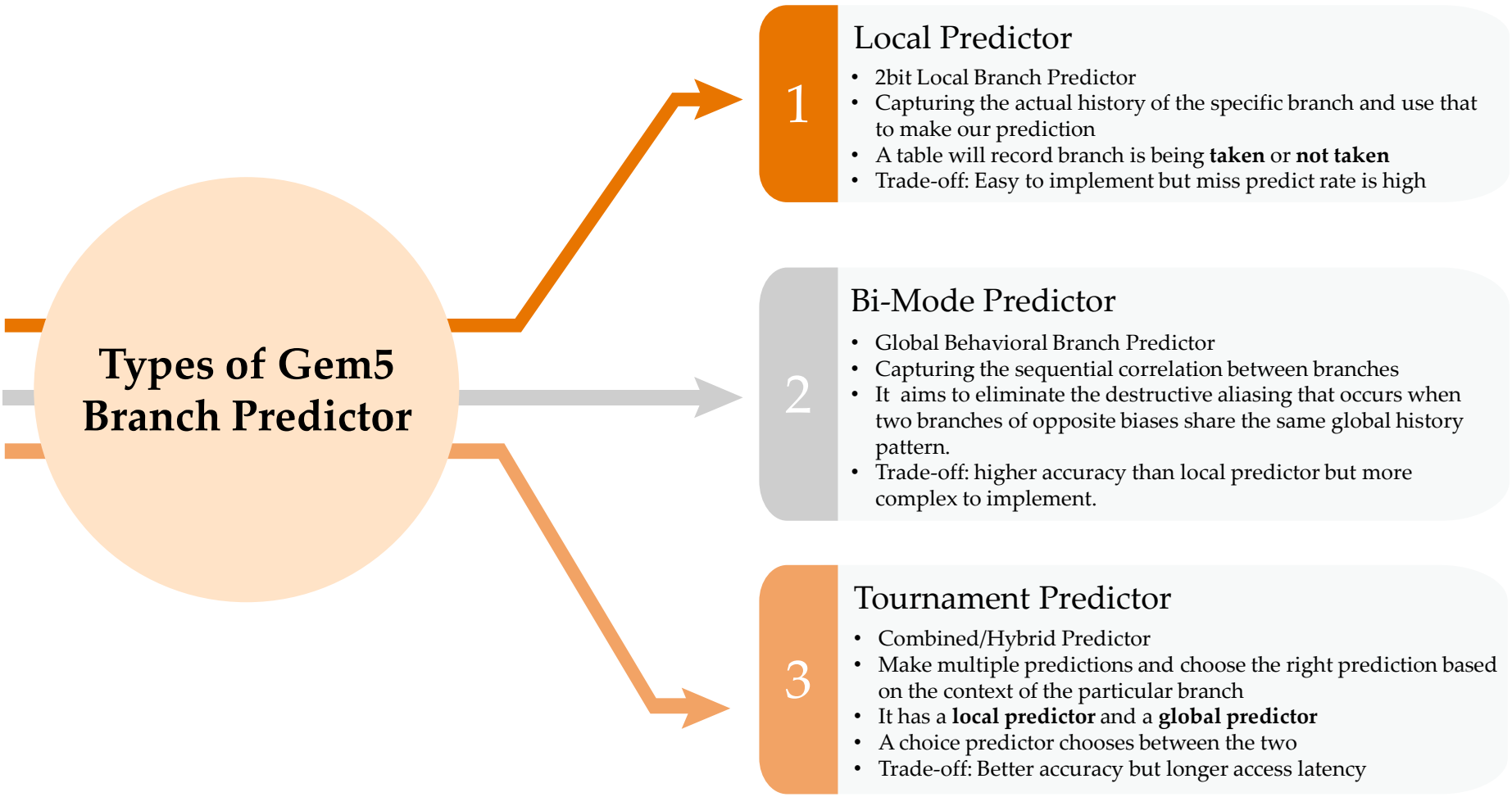
04

Very important to the performance of a deeply pipelined processor

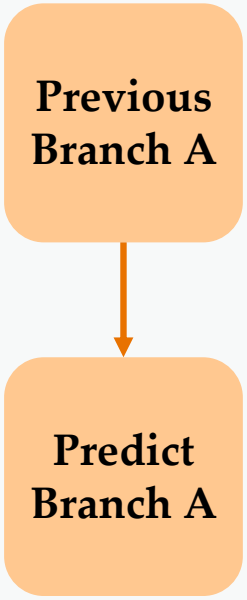
Gem5 Branch Predictors

Gem5 Simulator

- Open-source modular platform for system architecture research
- Discrete- event simulation platform with numerous models
- Most of gem5's code is written in C++ and Python.

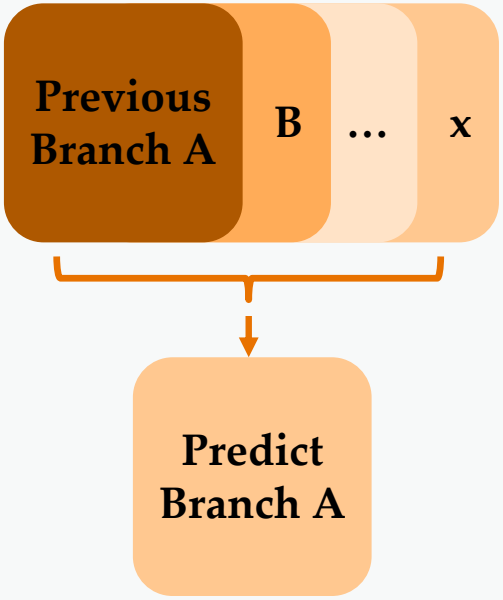

Types of Gem5 Branch Predictor
1
Local Predictor
• 2bit Local Branch Predictor
• Capturing the actual history of the specific branch and use that to make our prediction
• A table will record branch is being **taken** or **not taken**
• Trade-off: Easy to implement but miss predict rate is high
2
Bi-Mode Predictor
• Global Behavioral Branch Predictor
• Capturing the sequential correlation between branches
• It aims to eliminate the destructive aliasing that occurs when two branches of opposite biases share the same global history pattern.
• Trade-off: higher accuracy than local predictor but more complex to implement.
3
Tournament Predictor
• Combined/Hybrid Predictor
• Make multiple predictions and choose the right prediction based on the context of the particular branch
• It has a **local predictor** and a **global predictor**
• A choice predictor chooses between the two
• Trade-off: Better accuracy but longer access latency

Local Predictor



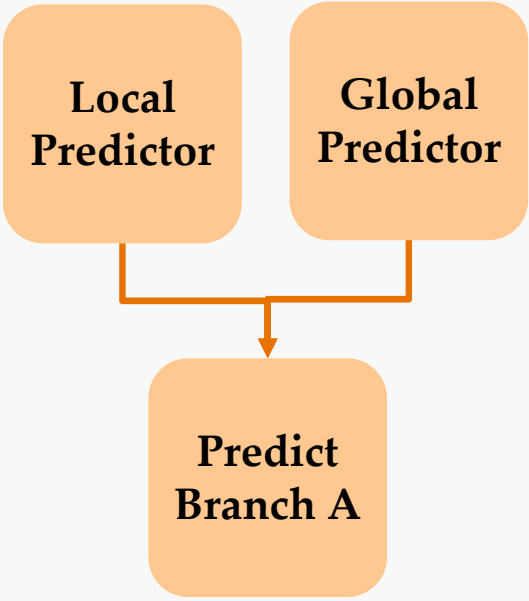
Bi-Mode Predictor

Global



Tournament Predictor

Hybrid



4 Things we have done

Setup

Set up Gem5 simulation environment

Adding BP

Adding Branch Predictor to Timing Simple CPU

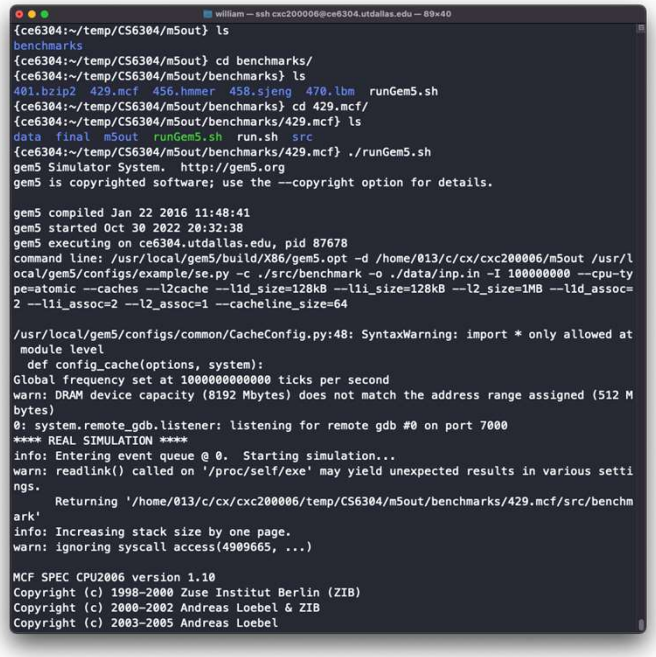
Adding Parameter

Adding extra Resulting Parameter in the Stats.txt file

BP Comparing

Running the benchmarks and comparing different branch predictor

1. Gem5 Setup



1. Login to CE6304 linux server
   - ssh <net-id>@ce6304.utdallas.edu
2. Copy gem5 folder to your local directory
   - cp –rf /usr/local/gem5 /home/<netid>/<proj_folder>
3. Downloads the Benchmarks
   - git clone https://github.com/timberjack/Project1_SPEC
4. Building Gem5 X86 with Scons
   - scons build/X86/gem5.opt –j5
5. Give permission to run the script
   - chmod +x ./runGem5.sh

2. Adding Branch Predictor

1. Go to file `$gem5/src/cpu/simple/BaseSimpleCPU.py`

2. At the bottom of file, you will see

```
branchPred = Param.BranchPredictor(Null, "Branch Predictor")
```

3. Change the "NULL" to one of following Branch Predictor

```
branchPred = Param.BranchPredictor(LocalBP(), "Branch Predictor")
branchPred = Param.BranchPredictor(BiModeBP(), "Branch Predictor")
branchPred = Param.BranchPredictor(TournamentBP(), "Branch Predictor")
```

3. Adding extra Resulting Parameter

1

Go to file `$gem5/$Gem5/src/cpu/pred/bpred_unit.cc`

2

Implement BTBMissPct

```
void
BPredUnit::regStats()
{
        ......

        /* Implement BTBMissPct */
        BTBMissPct
                .name(name() + ".BTBMissPct")
                .desc("BTB Miss Percentage")
                .precision(6);
                BTBMissPct = (1 - (BTBHits / BTBLookups)) * 100;
        /* Implement BTBMissPct */

        .....
}
```

3

Go to file `$gem5/$Gem5/src cpu pred bpred_unit.hh`

4

Add the following declaration for BTBMissPct

```
Stats::Scalar BTBCorrect;
/** Stat for percent times an entry in BTB found. */
Stats::Formula BTBHitPct;

/* Add the following declaration  */
/** Stat for percent times an entry in BTB miss. */
Stats::Formula BTBMissPct;

/** Stat for number of times the RAS is used to get a target. */
Stats::Scalar usedRAS;
/** Stat for number of times the RAS is incorrect. */
Stats::Scalar RASIncorrect;
```

5

Go to file `$gem5/$Gem5/$Gem5/src/cpu/simple/base.cc`

6

Implement BranchMispredPercent

```
void
BaseSimpleCPU::regStats()
{
        using namespace Stats;
        BaseCPU::regStats();

        for (ThreadID tid = 0; tid < numThreads; tid++) {
                . . . . .
                // Implement BranchMispredPercent
                t_info.BranchMispredPercent
                        .name(thread_str + ".BranchMispredPercent")
                        .desc("Percentage of Branch Mis-Predicts")
                        .prereq(t_info.BranchMispredPercent);
                t_info.BranchMispredPercent =
                        (t_info.numBranchMispred / t_info.numBranches)*100;
                // Implement BranchMispredPercent
        }
}
```

7

Go to file `$gem5/$Gem5/src/cpu/simple/exec_context.hh`

8

Add the following declaration for BranchMispredPercent

```
class SimpleExecContext : public ExecContext {

        public:
                . . . . . .
                // Modify
                Stats::Formula BranchMispredPercent;
                . . . . . .
}
```

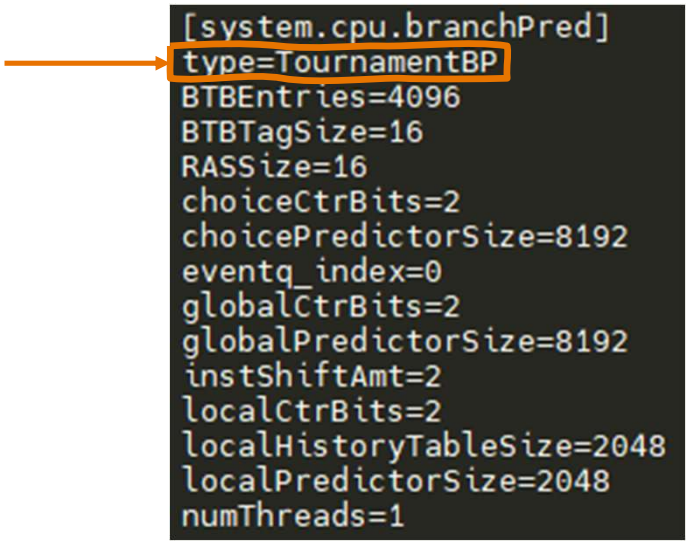
Gem5 Branch Predictors Results

Overview

Tournament Predictor

Result of CONFIG.INI FOR TournamentBP

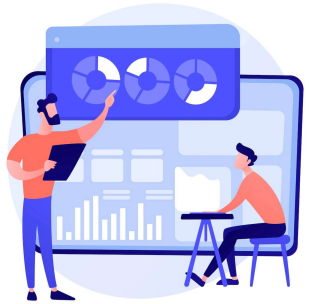
Branch Predictor type has been changed to TournamentBP



Benchmark Results

Overview

Branch Predictor Exploration



BTBMissPct and BranchMispredPercent on each BP

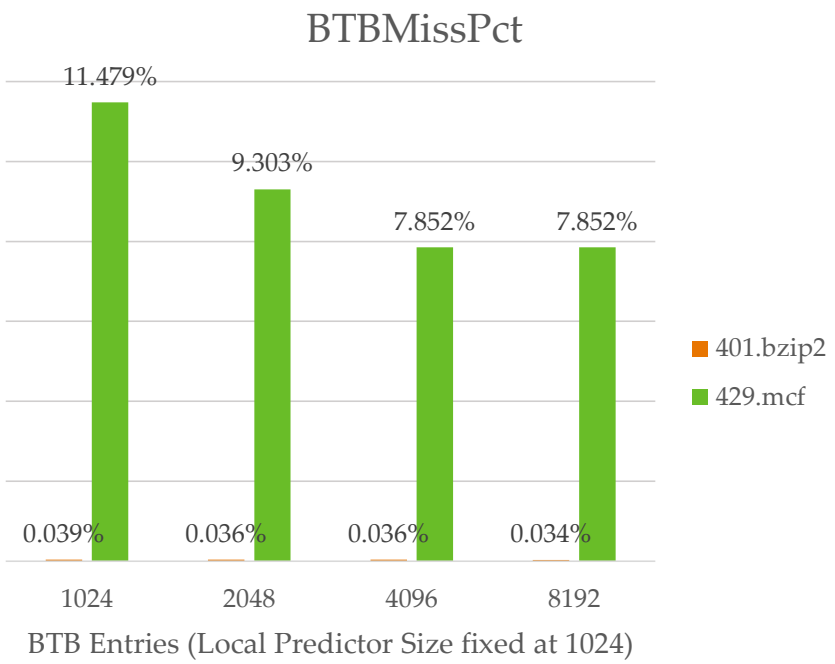
	BTBMissPct(%)		BranchMispredPercent (%)	
	401.bzip2	**429.mcf**	**401.bzip2**	**429.mcf**
local BTBEntries=2048 localPredictorSize=1024	0.0363	9.303	1.07	6.5322
TournamentBP BTBEntries=2048 localPredictorSize=1024 globalPredictorSize=4096 choicePredictorSize=4096	2.0578	31.9462	1.064	0.9698
BiModeBP BTBEntries=2048 globalPredictorSize=2048 choicePredictorSize=2048	0.0065	5.9759	1.1534	2.2625

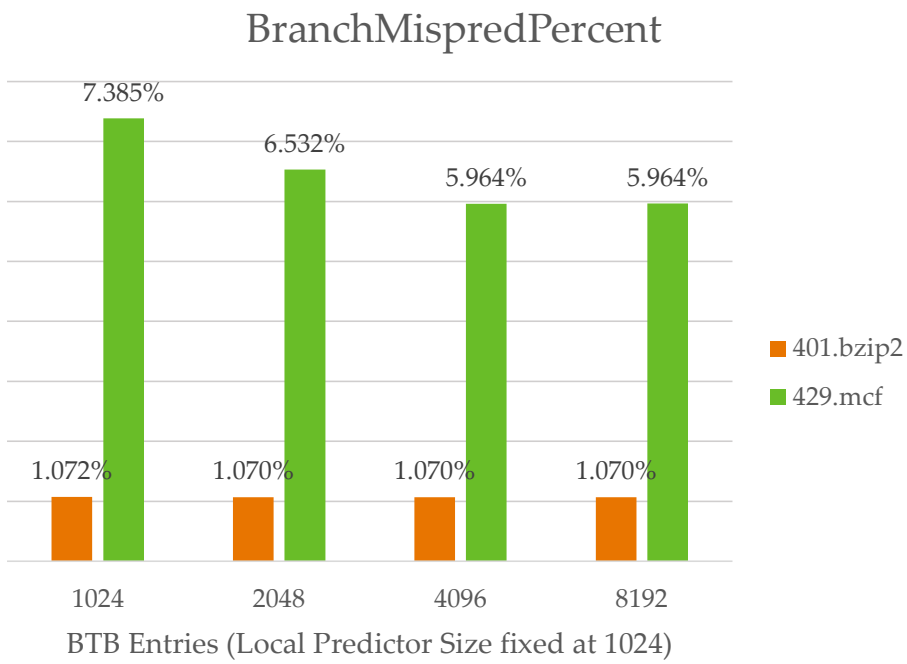
Comparing results between 3 different branch predictors

- All predictors have similar performance for 401.bzip2 benchmark
- With given parameters, the BTB Entries for TournamentBP is not sufficient. Therefore, the BTBMissPct of TournamentBP is very high
- From the BranchMispredPercent(429.mcf) point of view, TournamentBP > BiModeBP > LocalBP
- **Conclusion**:
  - Although TournamentBP has the potential to have the highest overall accuracy, it needs enough resources to make it work.
  - The choice of branch predictor is important. For some types of programs, selecting localBP can save resources while getting high accuracy.

Exploration - BTBEntries on LocalBP

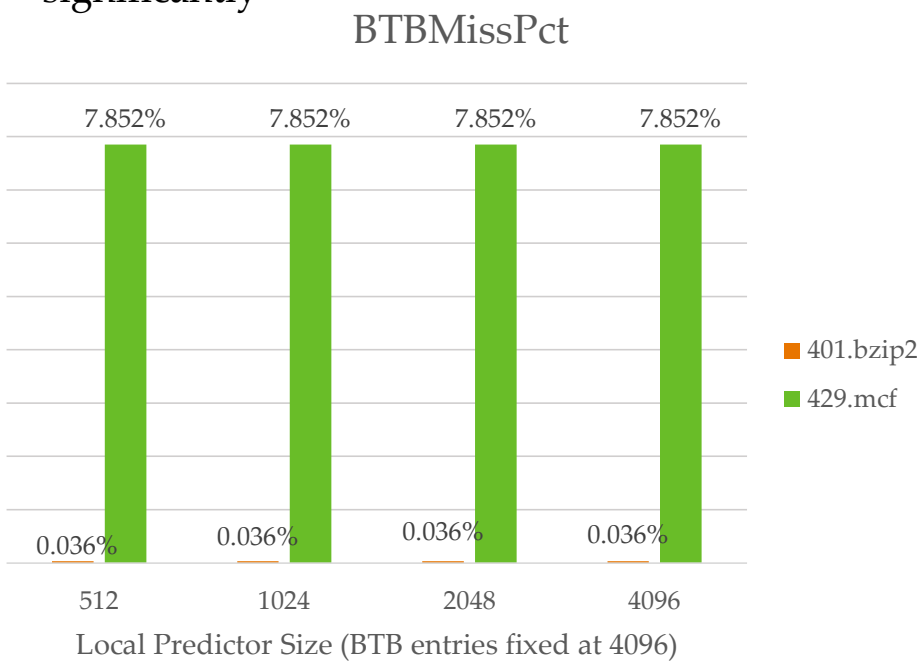
- 1024 BTB entries is enough for benchmark 401.bzip2
- Needs 4096 BTB entries for 429.mcf



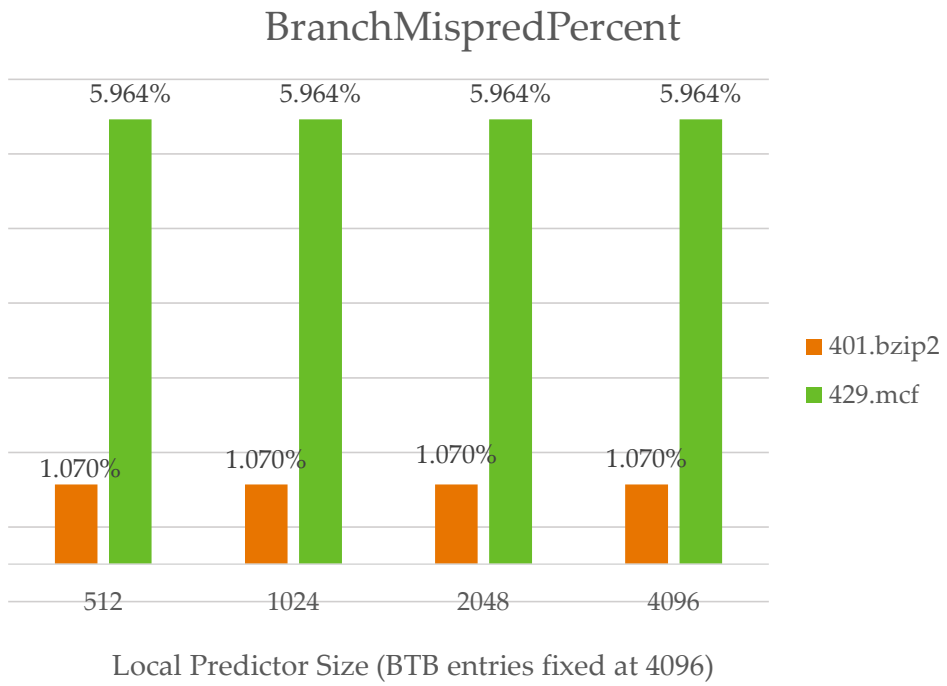


Exploration – Local Predictor size on LocalBP

- Local predictor size of 512 bytes is large enough for both 401.bzip2 and 429.mcf
- Increasing the size of local predictor size does not improve BTBMissPct nor BranchMispredPercent significantly

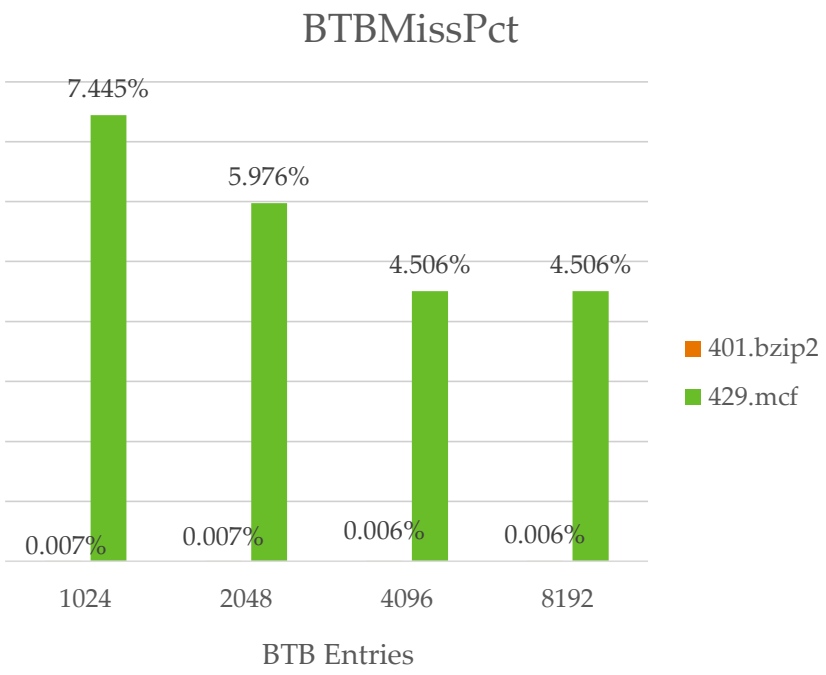


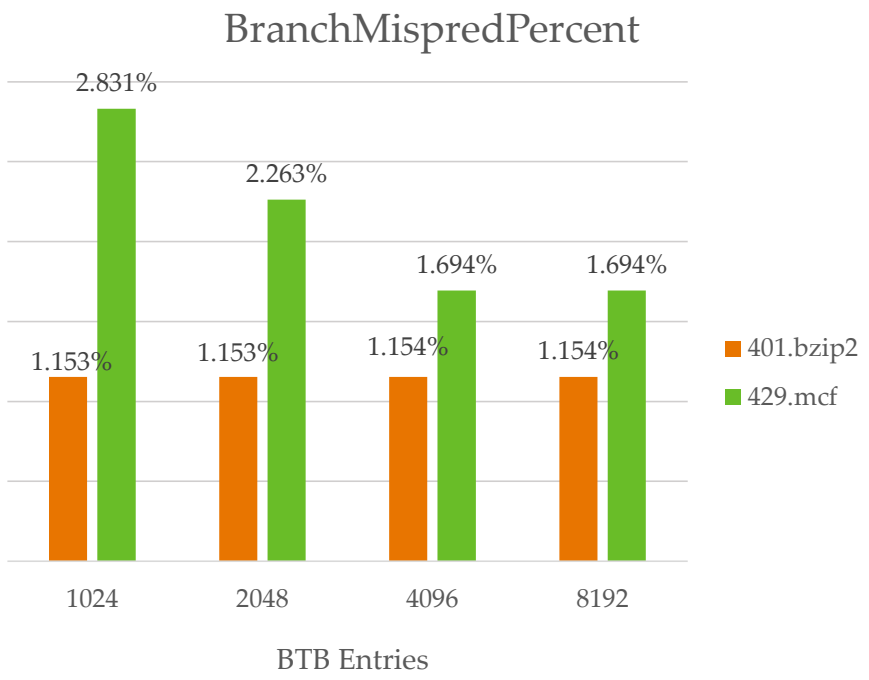
- The best configuration for local predictor would be
  - BTB entries: 4096
  - Local predictor size: 512



Exploration – BTB Entries in BiModeBP

- 1024 BTB entries is sufficient for 401.bzip2
- 429.mcf needs 4096 BTB entries

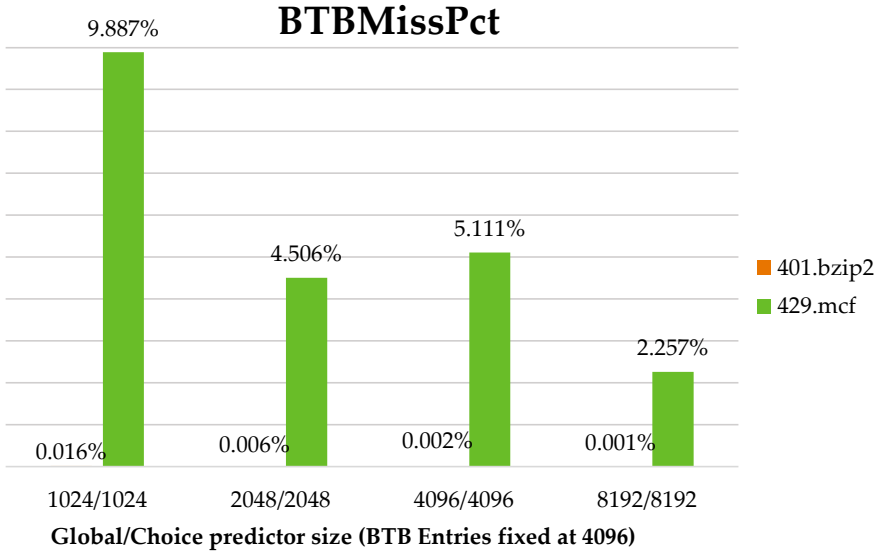


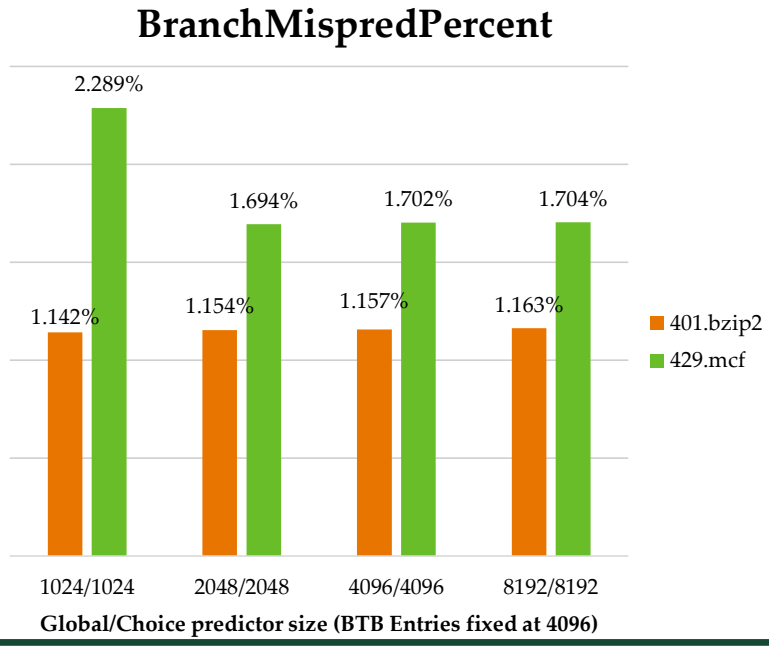


Exploration – Predictor sizes on BiModeBP

- Increasing global and choice predictor size from 1024 to 2048 greatly reduce BTBMissPct by 5.3% for 429.mcf
- Increasing global and choice predictor size after 2048 does not have significant impact.

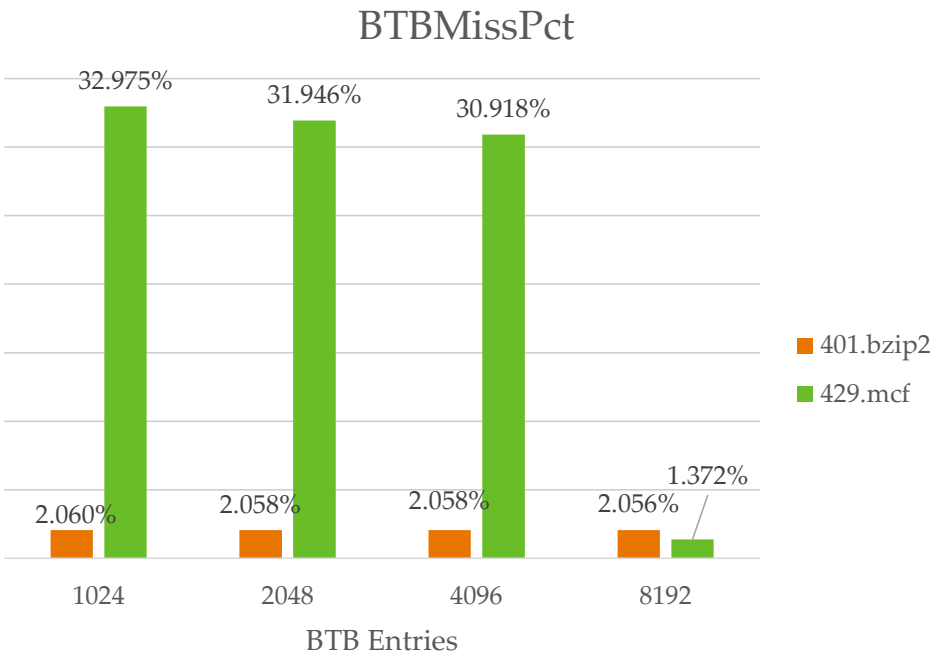
- The best configuration for TournamentBP would be
  - BTB entries: 4096
  - Global predictor size: 2048 (or 8192)
    - Chois predictor size: 2048 (or 8192)

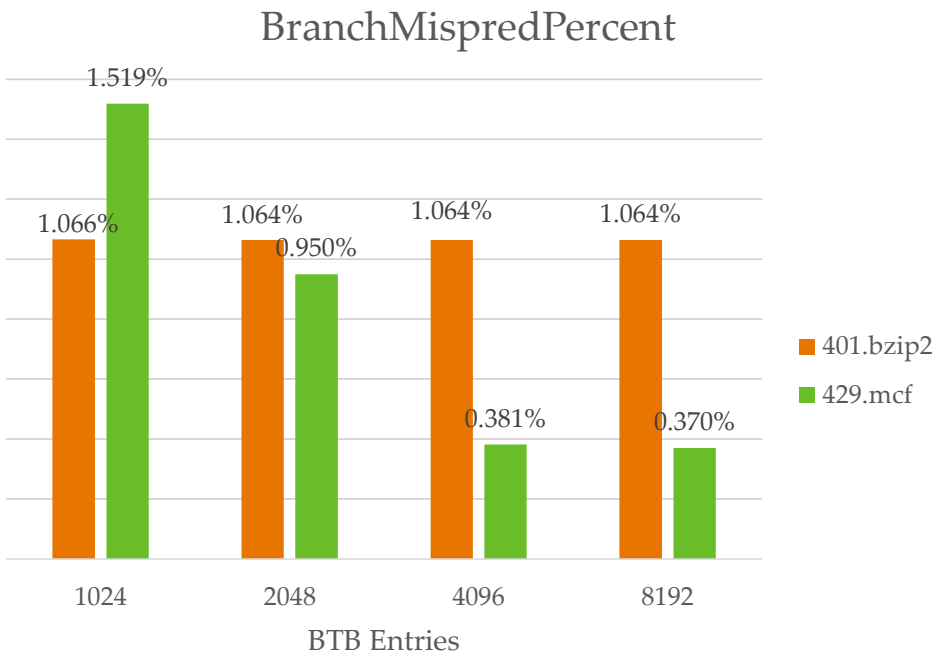




Exploration – BTB Entries on TournamentBP

- 1024 BTB entries is large enough for 401.bzip2
- Increasing BTB entries from 4096 to 8192 can significantly improve BTBMissPct

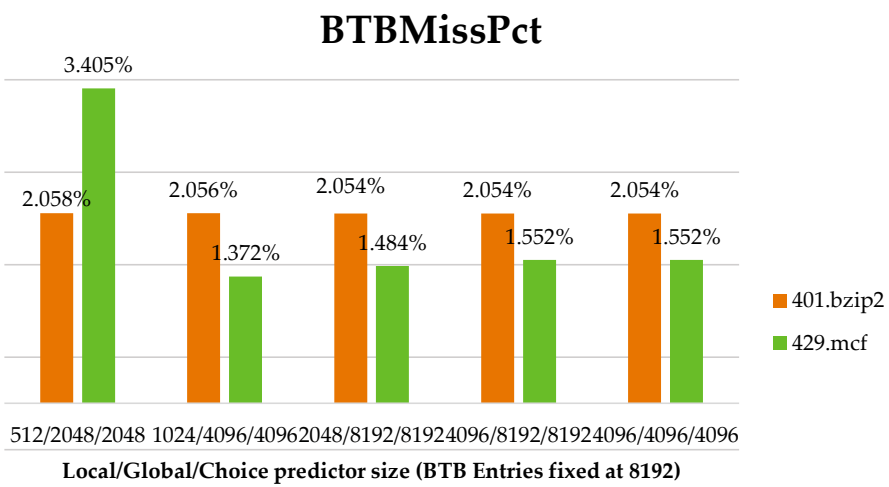


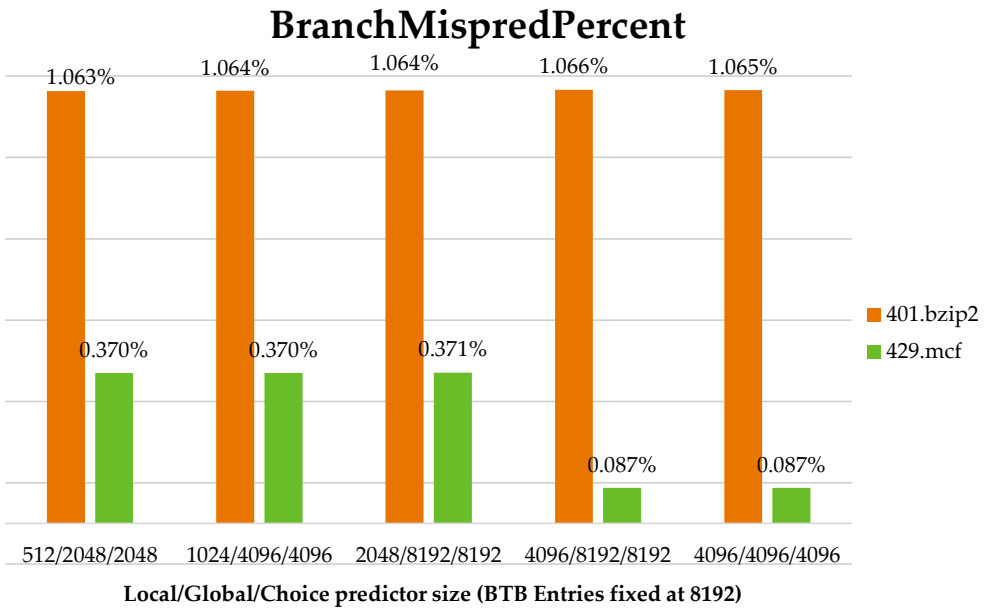


Exploration – Predictor sizes on TournamentBP

- Increasing global and choice predictor size from 2048 to 4096 reduce BTBMissPct by 1.4% for 429.mcf
- Increasing local predictor size from 2048 to 4096 reduce BranchMispredPercent by 0.29% for 429.mcf

- The best configuration for TournamentBP would be
  - BTB entries: 8192
  - Local predictor size: 4096
  - Global predictor size: 4096
  - Chois predictor size: 4096





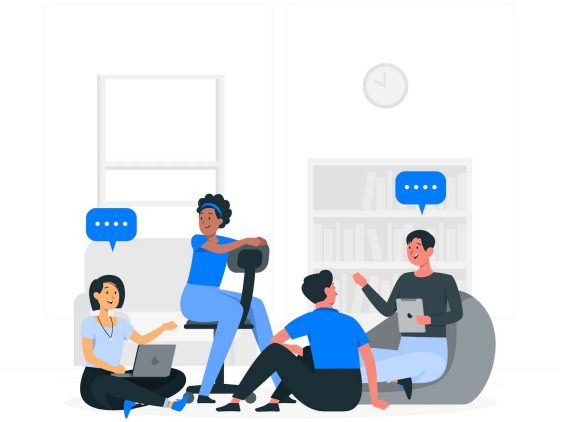
BTBMissPct and BranchMispredPercent on each BP (Comparing our new config with the given params)

BP Type	Params	BTBMissPct(%)		BranchMispredPercent (%)	
		401.bzip2	**429.mcf**	**401.bzip2**	**429.mcf**
localBP	**Given**	0.036	9.303	1.070	6.532
	New	0.036	7.852	1.070	5.964
Tournament BP	**Given**	2.058	31.946	1.064	0.970
	New	2.054	1.552	1.065	0.087
BiModeBP	**Given**	0.007	5.976	1.153	2.263
	New	0.006	4.506	1.154	1.694

Result Discussion

Overview

Conclusion



Conclusion

For benchmark 401.bzip2

LocalBP = TournamentBP > BiModeBP

Benchmark 401.bzip2 is an image compression program. Therefore, we can imagine it repeatedly runs the same piece of code for compression on different sectors of the image. The branch behavior should be more monotonous, and the local correlation should be more significant in 401.bzip2. We think that's the reason why LocalBP and TournamentBP perform better than BiModeBP in this scenario.

The number of branches in a compression program should be fewer as well. Hence, it also requires fewer BTBentries for all three branch predictors.

Conclusion

For benchmark 429.mcf

TournamentBP > BiModeBP > LocalBP

Benchmark 429.mcf runs public transportation scheduling algorithm. The complexity of the algorithm in 429.mcf should be higher than 401.bzip2. In a scheduling problem, A change of one vehicle's schedule will affect the schedule of the others, so it is more of a global correlation problem. Thus, TournamentBP and BiModeBP perform better in 429.mcf.

We imagine the architecture of a scheduling algorithm should have many if else statements to cope with different scenarios. Therefore, it is reasonable that scheduling algorithm requires more BTB entries than the compression algorithm.

Thank you

THE UNIVERSITY OF TEXAS AT DALLAS

Erik Jonsson School of Engineering and Computer Science