PROJECT 1: BRANCH PREDICTION

CE6304: COMPUTER ARCHITECTURE

Department of Electrical and Computer Engineering

Submitted By:
Srivishnu Srinivas (sxs220196)
Prathamesh Gadad (psg220003)

PART 1: PROBLEM DISCUSSION

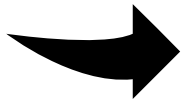
Setting Up The Server

- For this particular project, we've utilized Gem5, a specific version of the software installed on the UTD server, where all the required dependencies have been properly set up(No-Machine).
- Copied the Gem5 file from sever to our Local Directory using the command:
- “cp–rf /usr/local/gem5 /home/eng/s/sxs220196/CA_Project”.
- Used scons command to compile:
- “scons build/X86/gem5.opt”.
- we downloaded the Benchmark Files to our local Directory using the link Provided below,
- “GitHub - timberjack/Project1_SPEC: SPEC benchmark programs for CE6304 Project 1”.
- For this Project the given Benchmarks are :
- 1) 456.hmmer
- 2) 458.sjeng

What is Branch Prediction?

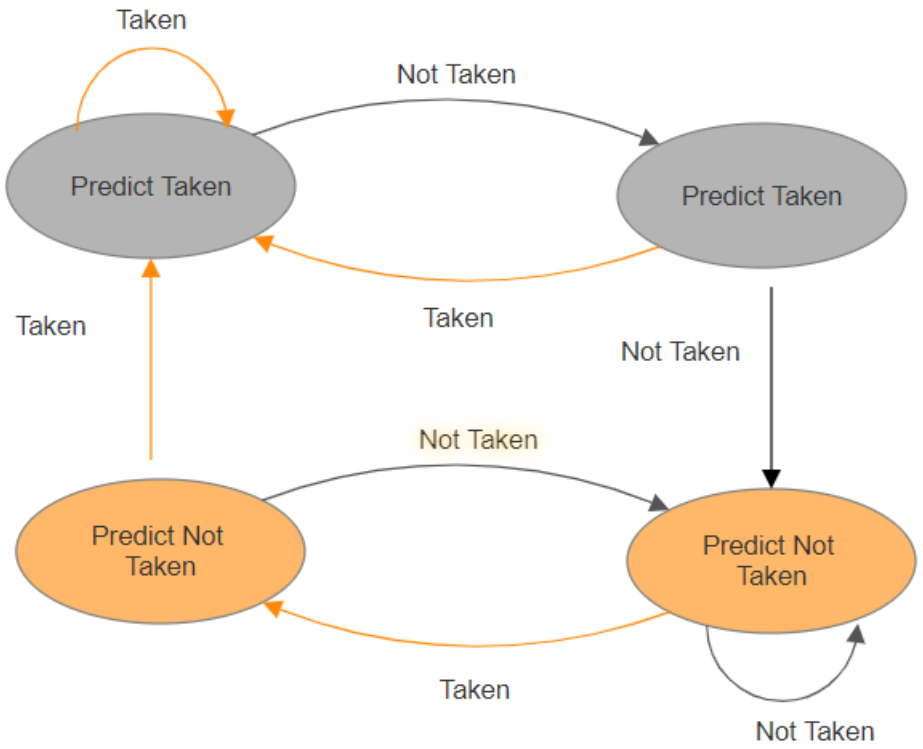
- A branch predictor is an electronic circuit that uses informed estimation to predict the likely outcome of a branching instruction before it actually takes place. The primary objective of a branch predictor is to enhance the flow of instructions within a pipeline. In modern pipelined microprocessor designs, such as the x86 architecture, branch predictors play a crucial role in achieving high-performance execution.

- The 3 Types of Branch Predictors are :
  - 2bit_local predictor.
  - Bi_mode Predictor.
  - Tournament predictor.



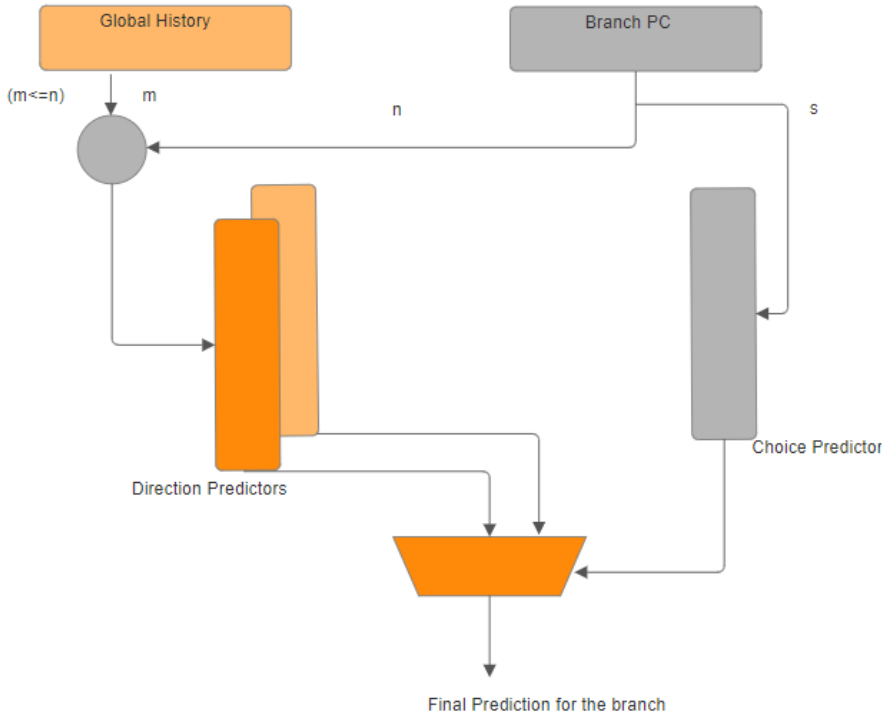
2_BIT Local Predictor:

The predictor only updates its prediction after making two consecutive incorrect forecasts. This is done by maintaining a prediction buffer with two bits, resulting in four distinct states. Two of these states signify a prediction of the branch being taken, while the other two states indicate a prediction of the branch not being taken.



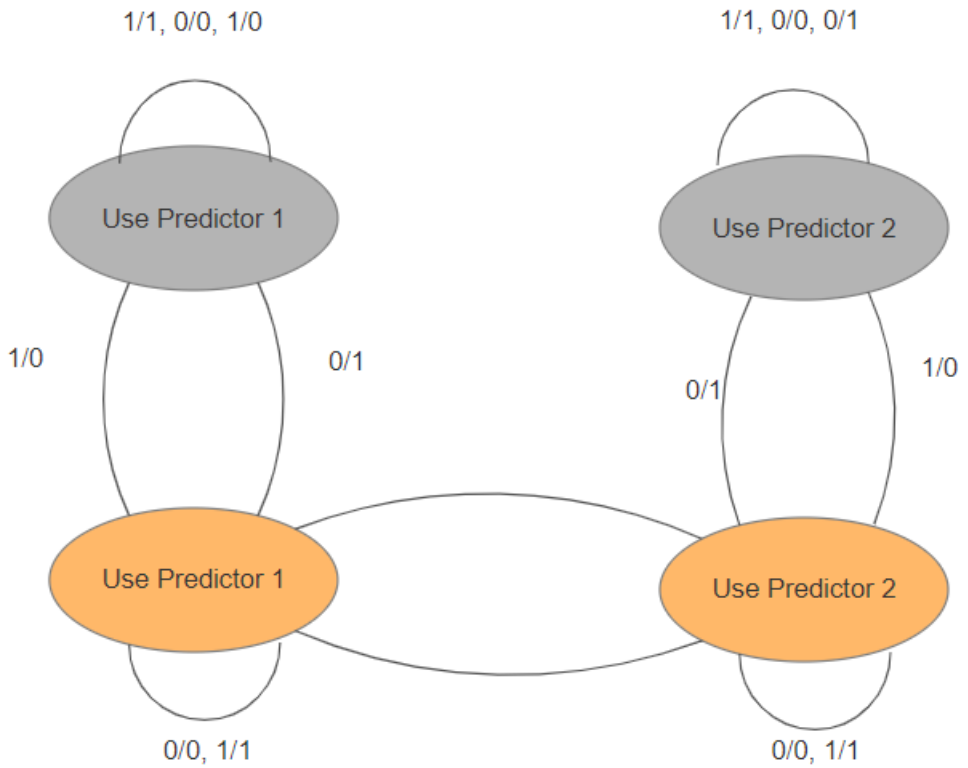
BI_MODE Predictor:

The primary purpose is to eliminate harmful aliasing in global history indexing techniques.



Tournament Predictor:

It is a combination of a local predictor with a global predictor.



PART 2: RESULTS OF config.ini

- To obtain the result in the "config.ini" file, you must make modifications to the "BaseSimpleCPU.py" file located in the specified path.

  “cd /home/eng/s/sxs220196/CA_Project/src/cpu/simple/BaseSimpleCPU.py”

- After the modification we need to compile the processor again using the command:

  “scons build/X86/gem5.opt”

- Used the following command to test “HELLOWORLD” program,

  “./build/X86/gem5.opt ./configs/example/se.py -c ./tests/test-progs/hello/bin/x86/linux/hello”

```
{ce6304:~/CA_Project} ./build/X86/gem5.opt ./configs/example/se.py -c ./tests/test-progs/hello/bin/x86/linux/
gem5 Simulator System.  http://gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 compiled Oct 21 2023 03:23:05
gem5 started Oct 21 2023 14:15:49
gem5 executing on ce6304.utdallas.edu, pid 112386
command line: ./build/X86/gem5.opt ./configs/example/se.py -c ./tests/test-progs/hello/bin/x86/linux/hello

Global frequency set at 1000000000000 ticks per second
warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (512 Mbytes)
0: system.remote_gdb.listener: listening for remote gdb #0 on port 7000
**** REAL SIMULATION ****
info: Entering event queue @ 0.  Starting simulation...
Hello world!
Exiting @ tick 5984500 because target called exit()
```

- The output can be viewed in ”m5out” folder in CA_Project.

  Path : “/home/eng/s/sxs220196/CA_Project”.

  vi config.ini

```
[system.cpu.branchPred]
type=TournamentBP
BTBEntries=4096
BTBTagSize=16
RASSize=16
choiceCtrBits=2
choicePredictorSize=8192
eventq_index=0
globalCtrBits=2
globalPredictorSize=8192
instShiftAmt=2
localCtrBits=2
localHistoryTableSize=2048
localPredictorSize=2048
numThreads=1
```

PART 3: MODIFICATION OF SOURCE CODE

Adding Extra Resulting Parameter in The Stats.Txt File (Output File)

The new parameters are :

- BTBMissPct
- numBranchMispredPercent = (numBranchMispred/numBranches)*100

- To add BTBMissPct parameter we need to edit the below files :
  - bpred_unit.hh
  - bpred_unit.cc

- To add numBranchMispredPercent parameter we need to edit the below files
  - exec_context.hh
  - base.cc

We can edit the file using “vi” command.

BTB Miss Pct :

- Edit the file to add the **Function** :
  - “cd /home/eng/s/sxs220196/CA_Project/src/cpu/pred”
  - “vi bpred_unit.hh”

```
/** Stat for number of BP lookups. */
Stats::Scalar lookups;
/** Stat for number of conditional branches predicted. */
Stats::Scalar condPredicted;
/** Stat for number of conditional branches predicted incorrectly. */
Stats::Scalar condIncorrect;
/** Stat for number of BTB lookups. */
Stats::Scalar BTBLookups;
/** Stat for number of BTB hits. */
Stats::Scalar BTBHits;
/** Stat for number of times the BTB is correct. */
Stats::Scalar BTBCorrect;
/** Stat for percent times an entry in BTB found. */
Stats::Formula BTBHitPct;
/**stat for BTBMissPct. */
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;
```

BTB Miss Pct :

- Edit the file to add the **Formula** :
  - “cd /home/eng/s/sxs220196/CA_Project/src/cpu/pred”
  - “vi bpred_unit.cc”

```
BTBHitPct
    .name(name() + ".BTBHitPct")
    .desc("BTB Hit Percentage")
    .precision(6);
BTBHitPct = (BTBHits / BTBLookups) * 100;

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

usedRAS
    .name(name() + ".usedRAS")
    .desc("Number of times the RAS was used to get a target.")
    ;
```

Branch Miss Pred Percent :

- Edit the file to add the **Function** :
  - “cd /home/eng/s/sxs220196/CA_Project/src/cpu/simple”
  - “vi exec_context.hh”

```
/// @{
/// Total number of branches fetched
Stats::Scalar numBranches;
/// Number of branches predicted as taken
Stats::Scalar numPredictedBranches;
/// Number of mispredicted branches
Stats::Scalar numBranchMispred;
///Misprediction Percentage
Stats::Formula numBranchMispredPercent;
/// @}
```

Branch Miss Pred Percent :

- Edit the file to add the **Formula**:
  - “cd /home/eng/s/sxs220196/CA_Project/src/cpu/simple”
  - “vi base.cc”

```
    t_info.numBranchMispred
        .name(thread_str + ".BranchMispred")
        .desc("Number of branch mispredictions")
        .prereq(t_info.numBranchMispred);

    t_info.numBranchMispredPercent
        .name(thread_str + ".numBranchMispredPercent")
        .desc("Number of Branch Mispred Percent");
    t_info.numBranchMispredPercent = (t_info.numBranchMispred / t_info.numBranches) * 100;
```

- After the compilation we can see the updated parameter's in "stats.txt" file and verify.

  "~m5out/stats.txt"

```
system.cpu.branchPred.BTBHitPct              36.773547                       # BTB Hit Percentage
system.cpu.branchPred.BTBMissPct             63.226453                       # BTB Miss Percentage
system.cpu.branchPred.usedRAS                  105                           # Number of times the RAS was used to get a target.
system.cpu.branchPred.RASInCorrect              75                           # Number of incorrect RAS predictions.

system.cpu.BranchMispred                       477                           # Number of branch mispredictions
system.cpu.numBranchMispredPercent           36.218679                       # Number of Branch Mispred Percent
system.cpu.op_class::No_OpClass                 1      0.01%      0.01% # Class of executed instruction
```

PART 4: BRANCH PREDICTION EXPLORATION

- In Part-2, we observed that it's possible to alter the branch predictor type by making changes to the "BaseSimpleCPU.py" file. This involves adding parameters to various files like "bpred_unit.hh," "bpred_unit.cc," "exec_context.hh," and "base.cc."

- Steps need to be followed to do Branch Exploration:
  - Changing the Parameter value in“BranchPredictor.py”
  - Recompile using the command“scons build/X86/gem5.opt”
  - Then, run using the command“sh runGem5.sh”, for respective Benchmark.

BranchPredictor.py

- Changing the Parameter value in“BranchPredictor.py”
  - “cd /home/eng/s/sxs220196/CA_Project/src/cpu/pred”
  - “vi BranchPredictor.py”

- BTB Entries :

```
class BranchPredictor(SimObject):
    type = 'BranchPredictor'
    cxx_class = 'BPredUnit'
    cxx_header = "cpu/pred/bpred_unit.hh"
    abstract = True

    numThreads = Param.Unsigned(1, "Number of threads")
    BTBEntries = Param.Unsigned(4096, "Number of BTB entries")
    BTBTagSize = Param.Unsigned(16, "Size of the BTB tags, in bits")
    RASSize = Param.Unsigned(16, "RAS size")
    instShiftAmt = Param.Unsigned(2, "Number of bits to shift instructions by")
```

Local BP:

Change the “localPredictorSize”,

```
class LocalBP(BranchPredictor):
    type = 'LocalBP'
    cxx_class = 'LocalBP'
    cxx_header = "cpu/pred/2bit_local.hh"

    localPredictorSize = Param.Unsigned(1024, "Size of local predictor")
    localCtrBits = Param.Unsigned(2, "Bits per counter")
```

BiMode BP:

Change the “globalPredictorSize” and “choicePredictorSize”,

```
class BiModeBP(BranchPredictor):
    type = 'BiModeBP'
    cxx_class = 'BiModeBP'
    cxx_header = "cpu/pred/bi_mode.hh"

    globalPredictorSize = Param.Unsigned(2048, "Size of global predictor")
    globalCtrBits = Param.Unsigned(2, "Bits per counter")
    choicePredictorSize = Param.Unsigned(2048, "Size of choice predictor")
    choiceCtrBits = Param.Unsigned(2, "Bits of choice counters")
```

Tournament BP:

Change the “localPredictorSize”, “globalPredictorSize”, and “choicePredictorSize”.

```
class TournamentBP(BranchPredictor):
    type = 'TournamentBP'
    cxx_class = 'TournamentBP'
    cxx_header = "cpu/pred/tournament.hh"

    localPredictorSize = Param.Unsigned(2048, "Size of local predictor")
    localCtrBits = Param.Unsigned(2, "Bits per counter")
    localHistoryTableSize = Param.Unsigned(2048, "size of local history table")
    globalPredictorSize = Param.Unsigned(8192, "Size of global predictor")
    globalCtrBits = Param.Unsigned(2, "Bits per counter")
    choicePredictorSize = Param.Unsigned(8192, "Size of choice predictor")
    choiceCtrBits = Param.Unsigned(2, "Bits of choice counters")
```

runGem5:

“runGem5.sh” for 456.hmmer is shown below,

```
# -- an example to run SPEC 429.mcf on gem5, put it under 429.mcf folder --
export GEM5_DIR=/home/eng/s/sxs220196/CA_Project
export BENCHMARK=/home/eng/s/sxs220196/CA_Project/benchmarks/456.hmmer/src/benchmark
export ARGUMENT=/home/eng/s/sxs220196/CA_Project/benchmarks/456.hmmer/data/bombesin.hmm
time $GEM5_DIR/build/X86/gem5.opt -d ~/m5out $GEM5_DIR/configs/example/se.py -c $BENCHMARK -o $ARGUMENT -I 5000000
--cpu-type=atomic --caches --l2cache --l1d_size=128kB --l1i_size=128kB --l2_size=1MB --l1d_assoc=2 --l1i_assoc=2 --
l2_assoc=1 --cacheline_size=64
~
```

runGem5:

“runGem5.sh” for 458.sjeng is shown below,

```
export GEM5_DIR=/home/eng/s/sxs220196/CA_Project
export BENCHMARK=/home/eng/s/sxs220196/CA_Project/benchmarks/458.sjeng/src/benchmark
export ARGUMENT=/home/eng/s/sxs220196/CA_Project/benchmarks/458.sjeng/data/test.txt
time $GEM5_DIR/build/X86/gem5.opt -d ~/m5out $GEM5_DIR/configs/example/se.py -c $BENCHMARK -o $ARGUMENT -I 5000000 --cpu-
type=atomic --caches --l2cache --l1d_size=128kB --l1i_size=128kB --l2_size=1MB --l1d_assoc=2 --l1i_assoc=2 --l2_assoc=1 -
-cacheline_size=64
~
```

Automation Exploration:

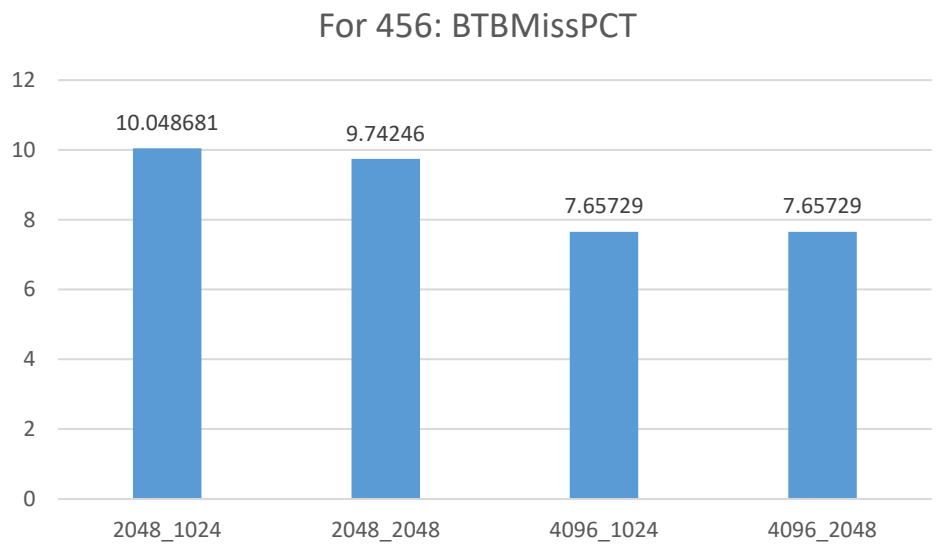
- we've adjusted the specified parameters and assigned them their respective values. The results and their explanations can be found in the subsequent slides.
- We have used python for Automation.
- Automation is a time-saving approach because it allows us to explore various potential combinations of branch predictors using a single script. This script can modify all the branch predictor parameter values for each iteration. In contrast, manual exploration would consume a significant amount of time due to the multitude of possible combinations. The results of this automated process are graphically presented in the following slides.

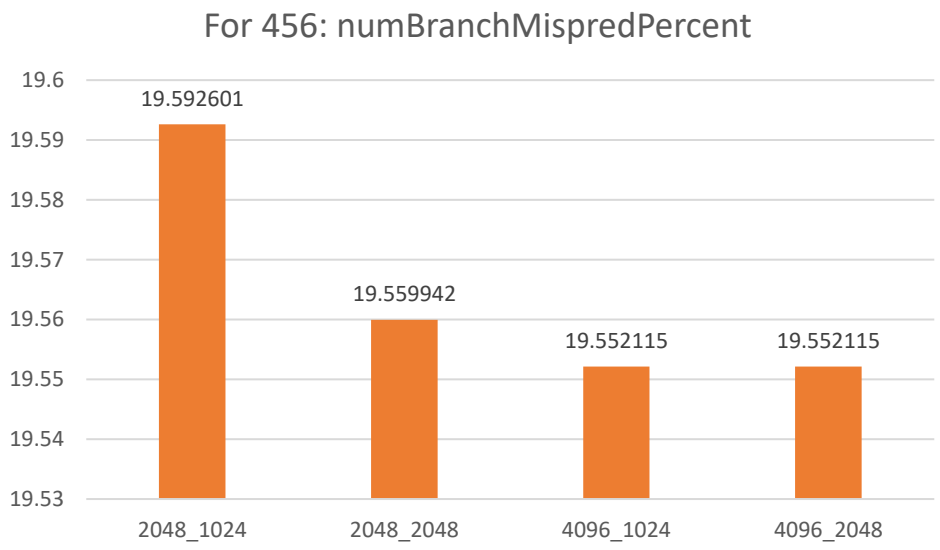
Parameters	LocalBP()	BiModeBP()	TournamentBP()
BTBEntries	2048 -> 4096	2048 -> 4096	2048 -> 4096
localPredictorSize	1024 -> 2048		1024 -> 2048
globalPredictorSize		2048 -> 4096 -> 8192	4096 -> 8192
choicePredictorSize		2048 -> 4096 -> 8192	4096 -> 8192

Steps to Automate The Process (Automation Algorithm):

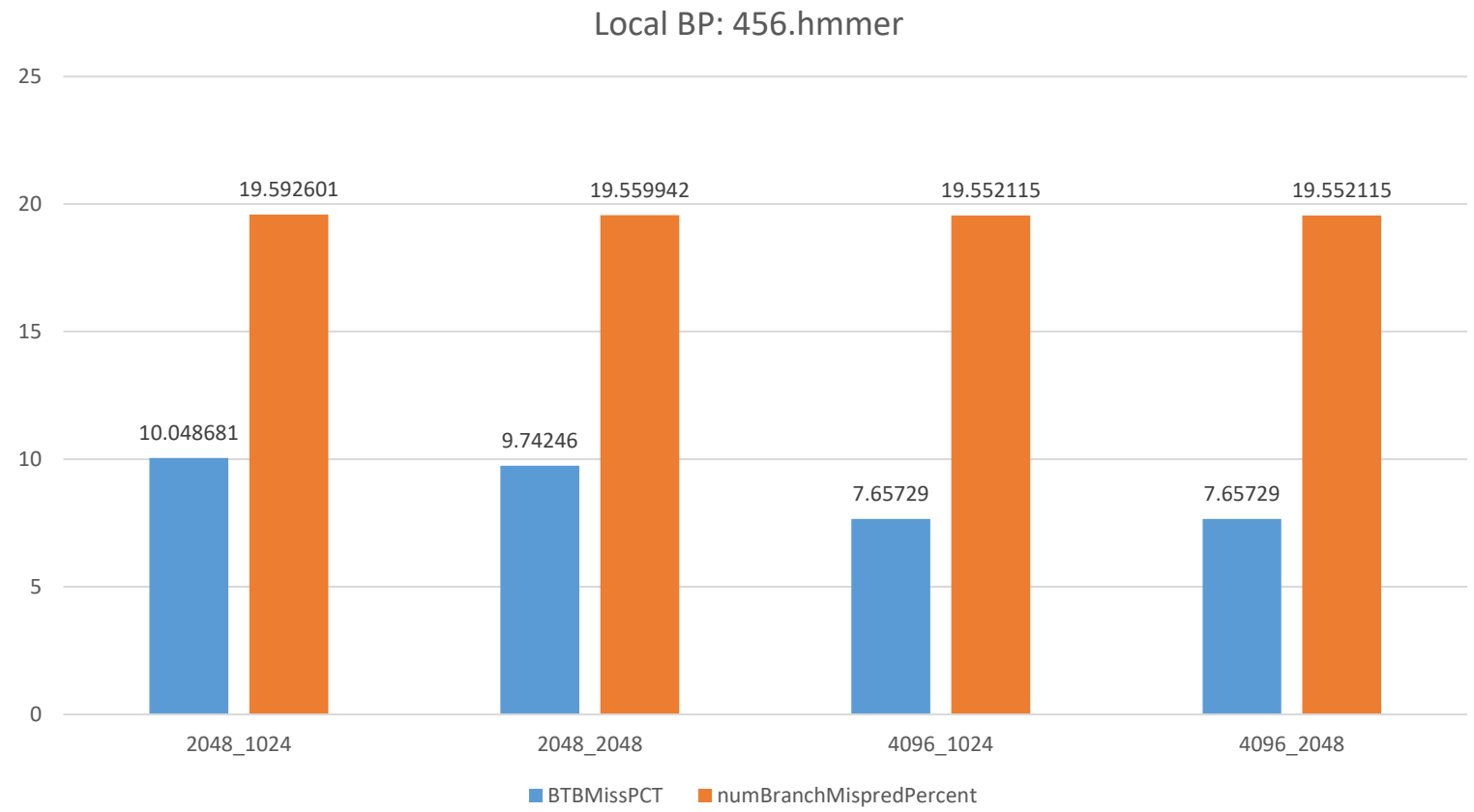
- Using Python's "subprocess" and "itertools" libraries to automate tasks.
- Modifications were made to the branch predictor type in the "BaseSimpleCPU.py" file, adjusting necessary arguments.
- The "BranchPredictor.py" script was utilized to modify parameters for all possible combinations for each predictor.
- Following these changes, the simulation configuration was rebuilt using the "scons build/X86/gem5.opt" command.
- The "runGem5.sh" script was executed within each benchmark's folder, specifying the output location.
- For each iteration, the command "sh runGem5.sh" was repeated to conduct the simulation.
- A total of 76 combinations of benchmarks were simulated, and the respective code files have been uploaded to the e-Learning platform for reference.

LocalBP Exploration for 456.hmmer:

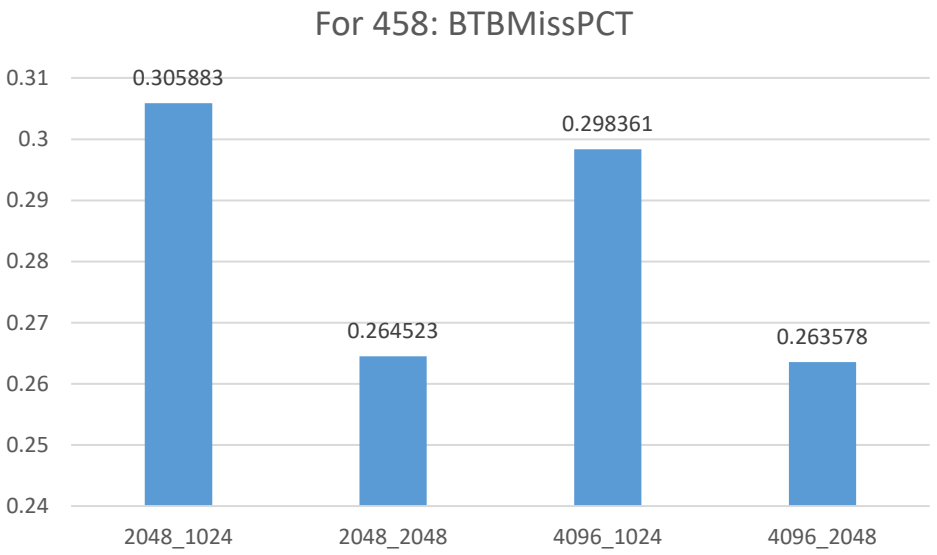


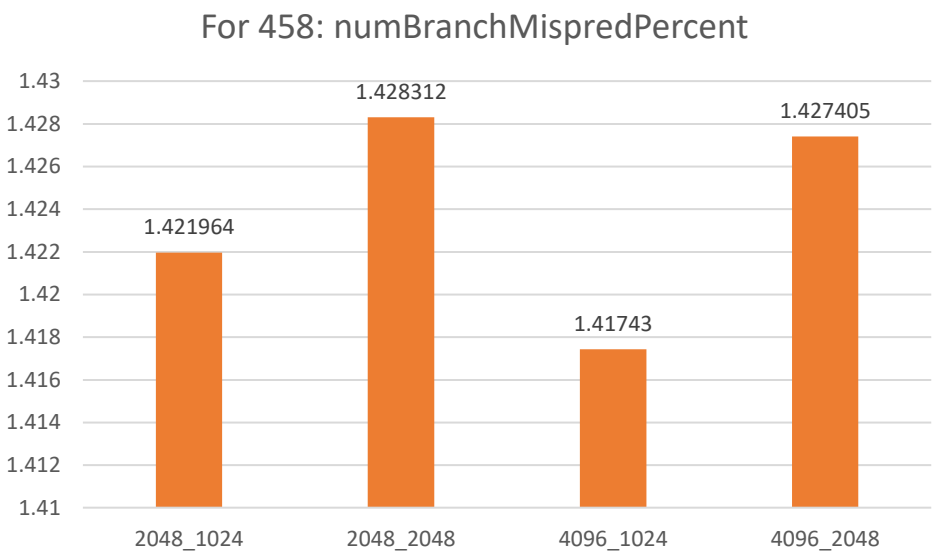


LocalBP Exploration for 456.hmmer:

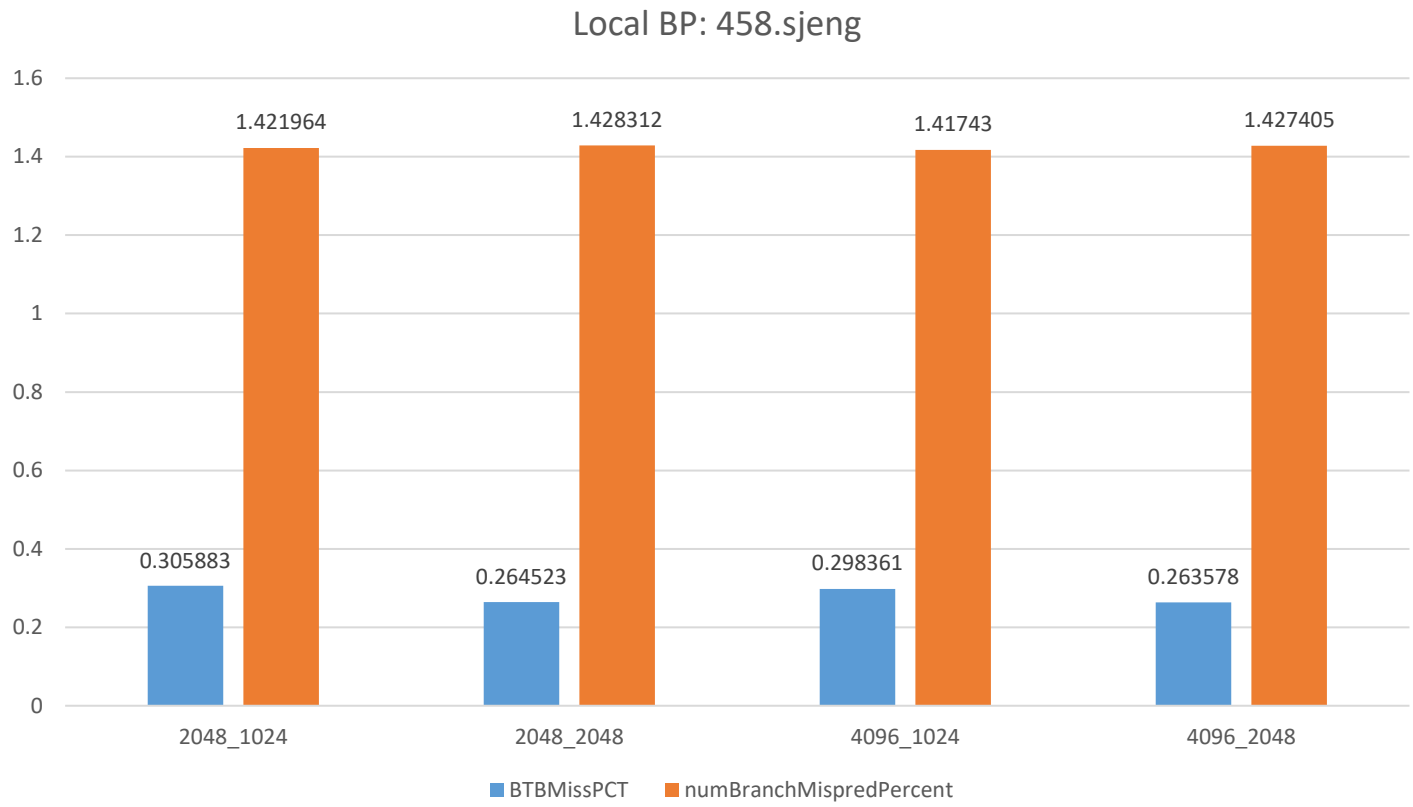


LocalBP Exploration for 458.sjeng:

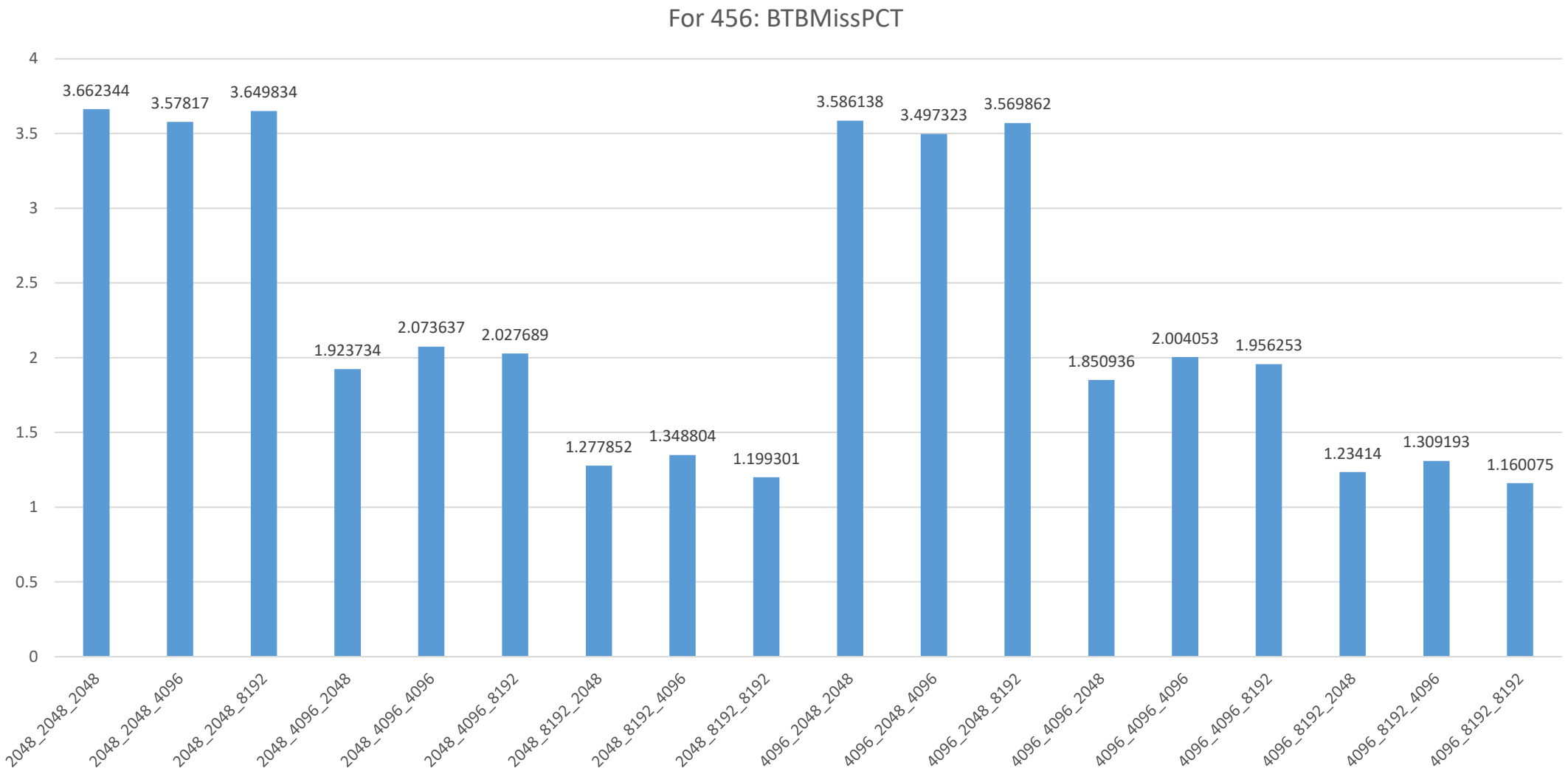




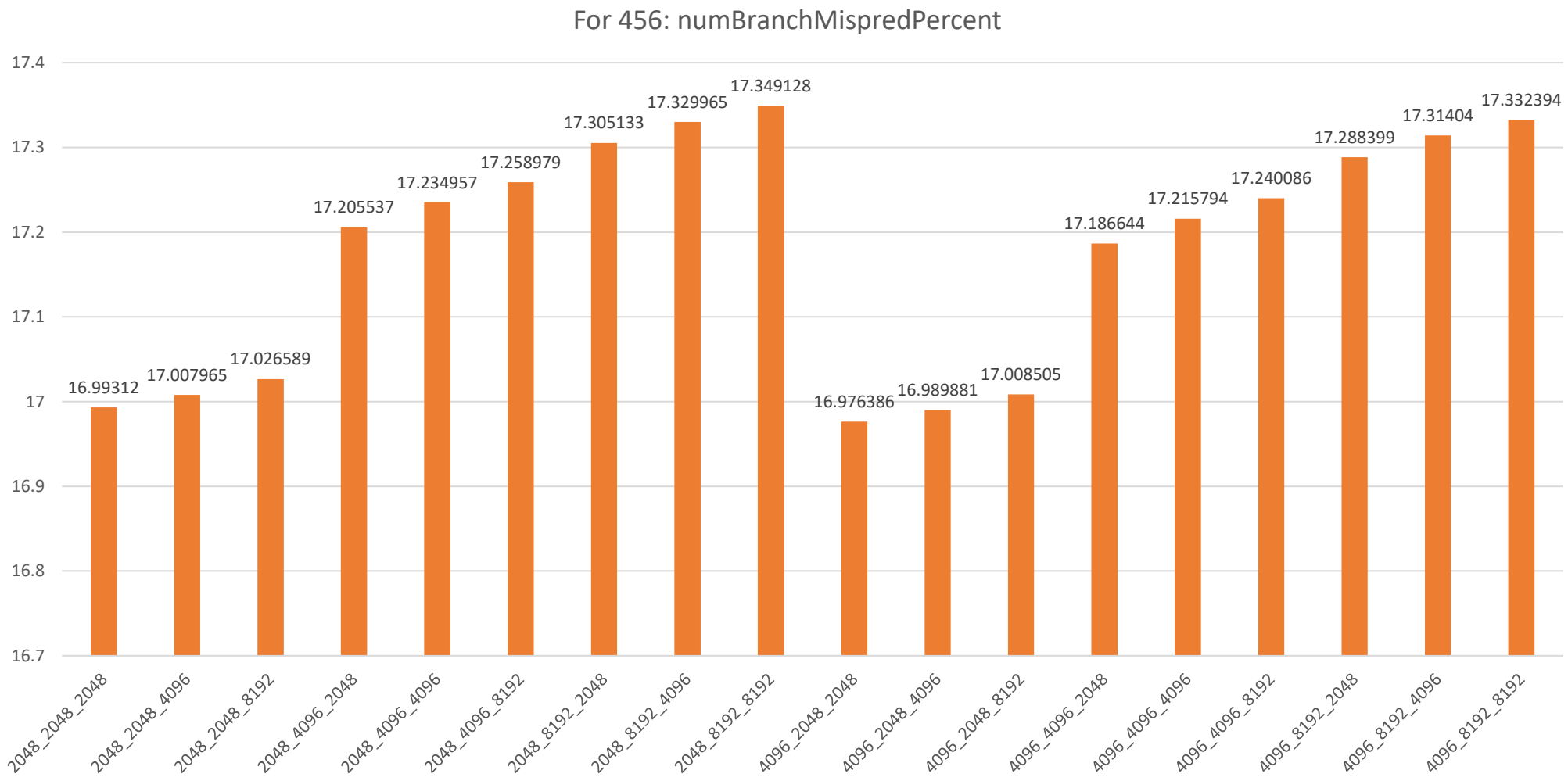
LocalBP Exploration for 458.sjeng:



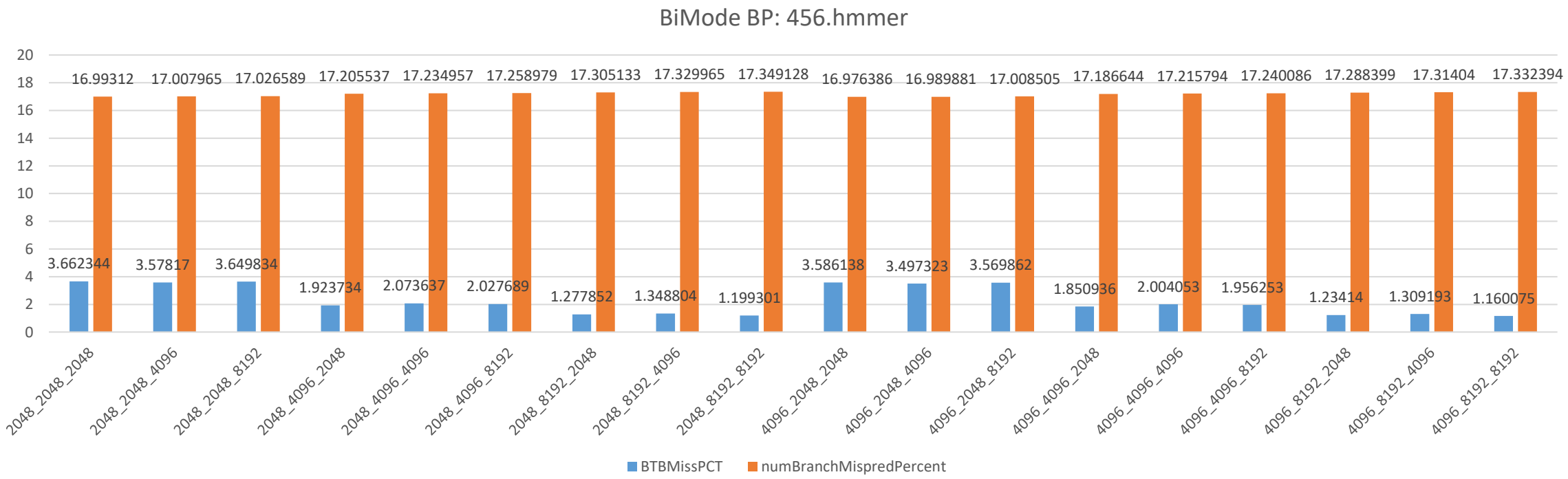
BiModeBP Exploration for 456.hmmer:



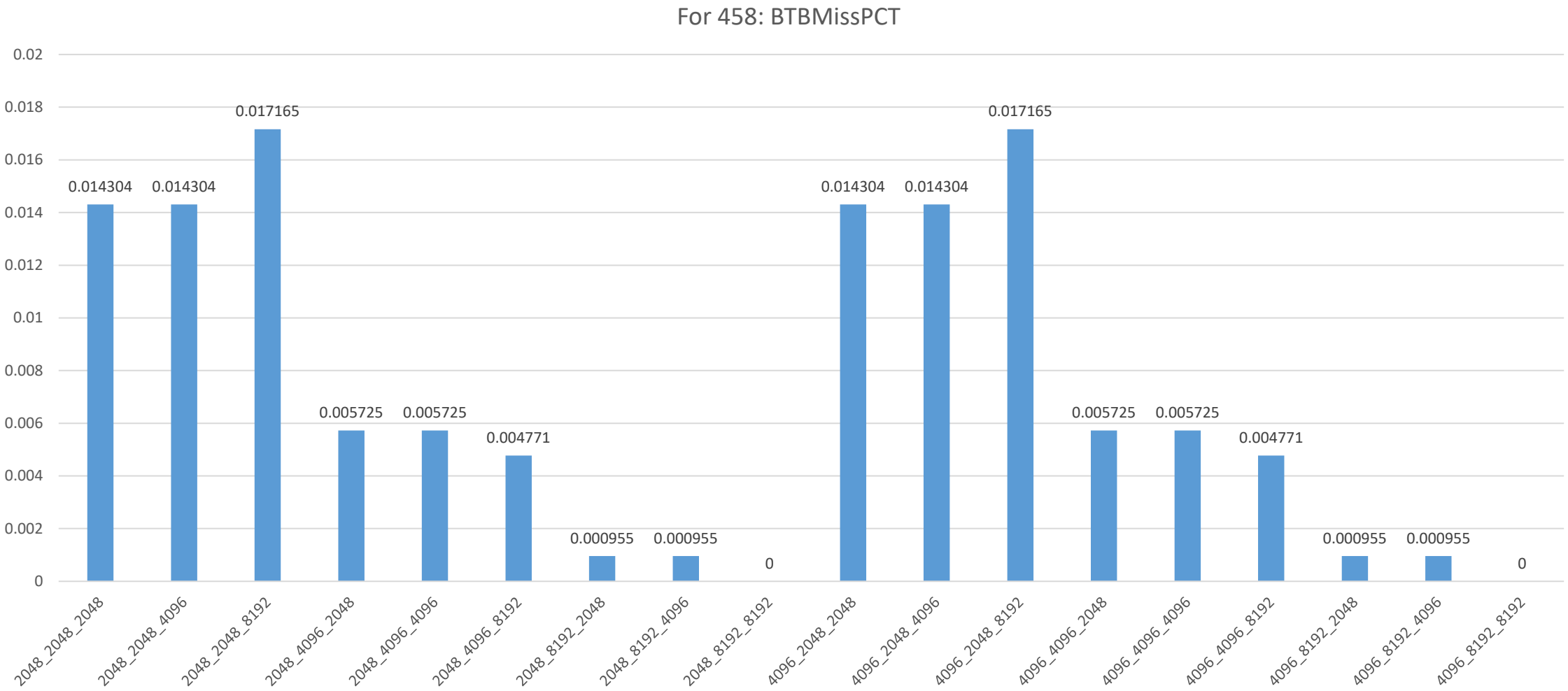
BiModeBP Exploration for 456.hmmer:



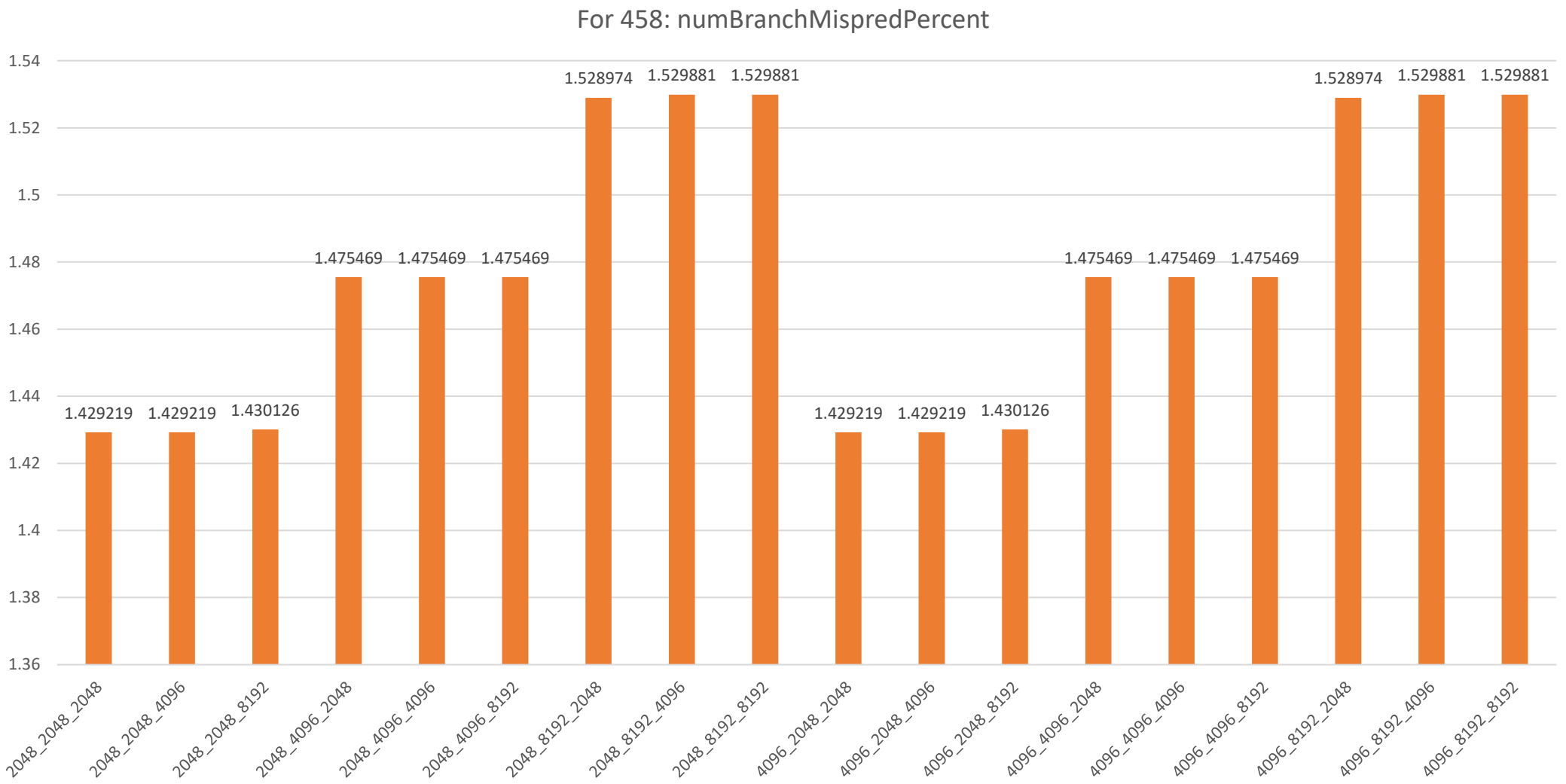
BiModeBP Exploration for 456.hmmer:



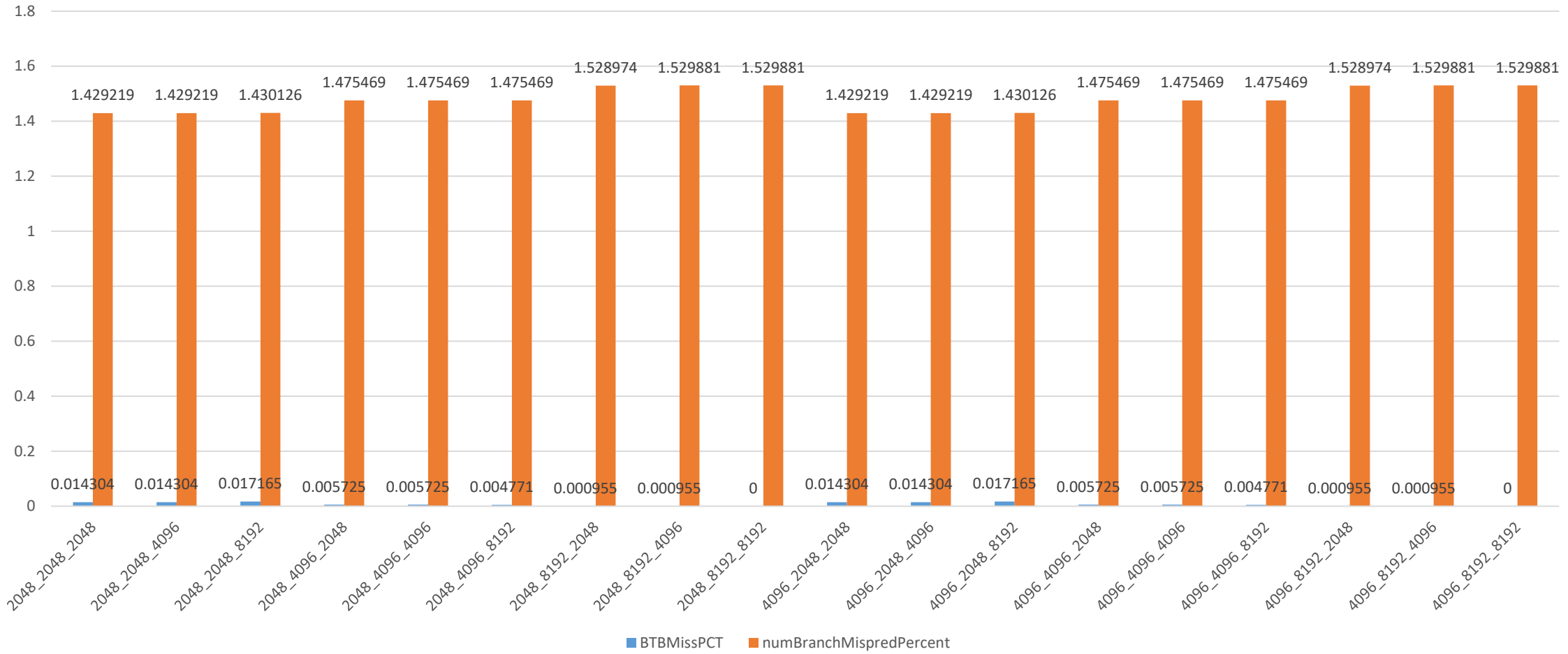
BP Exploration for 458.sjeng:



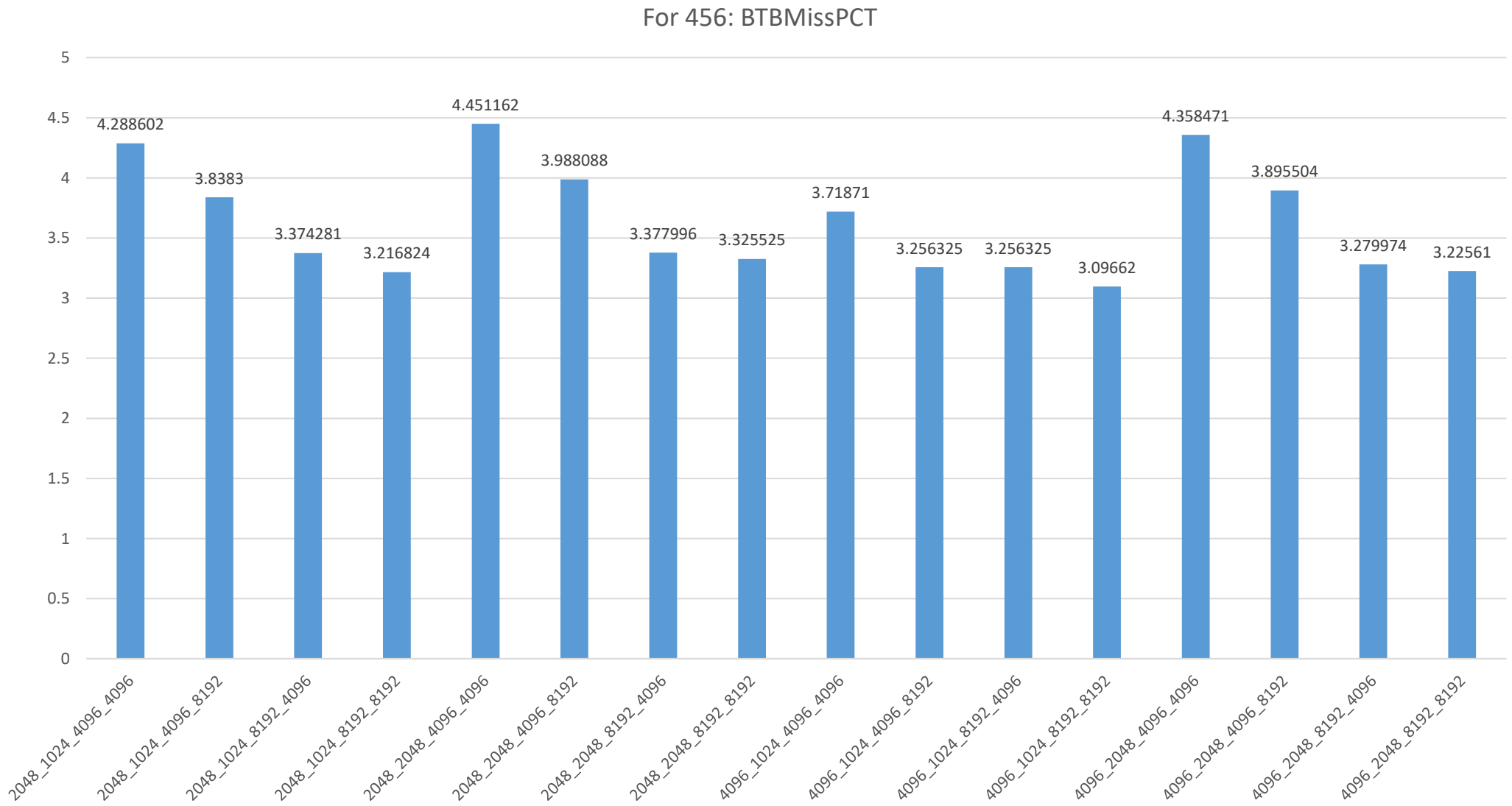
BP Exploration for 458.sjeng:



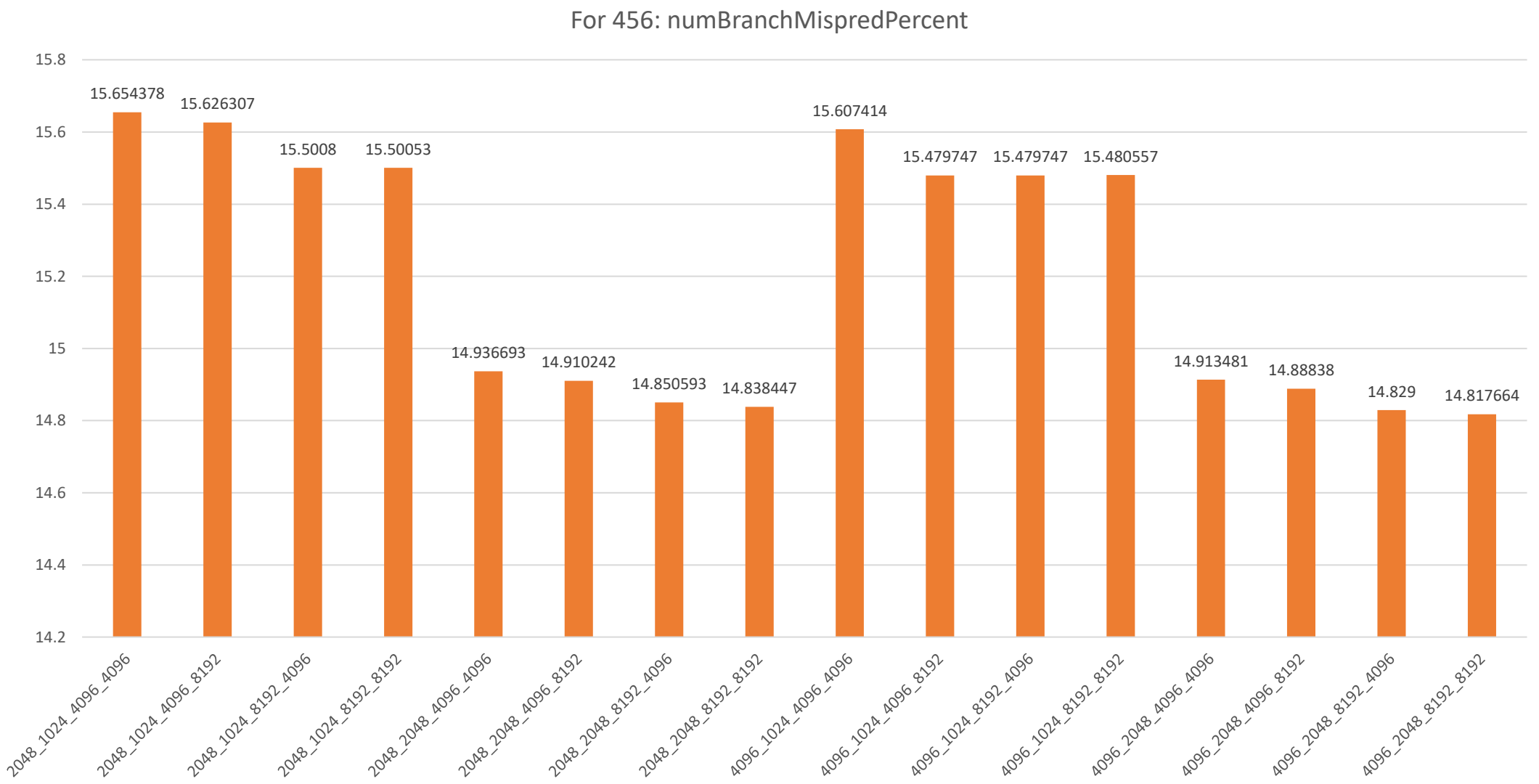
BP Exploration for 458.sjeng:



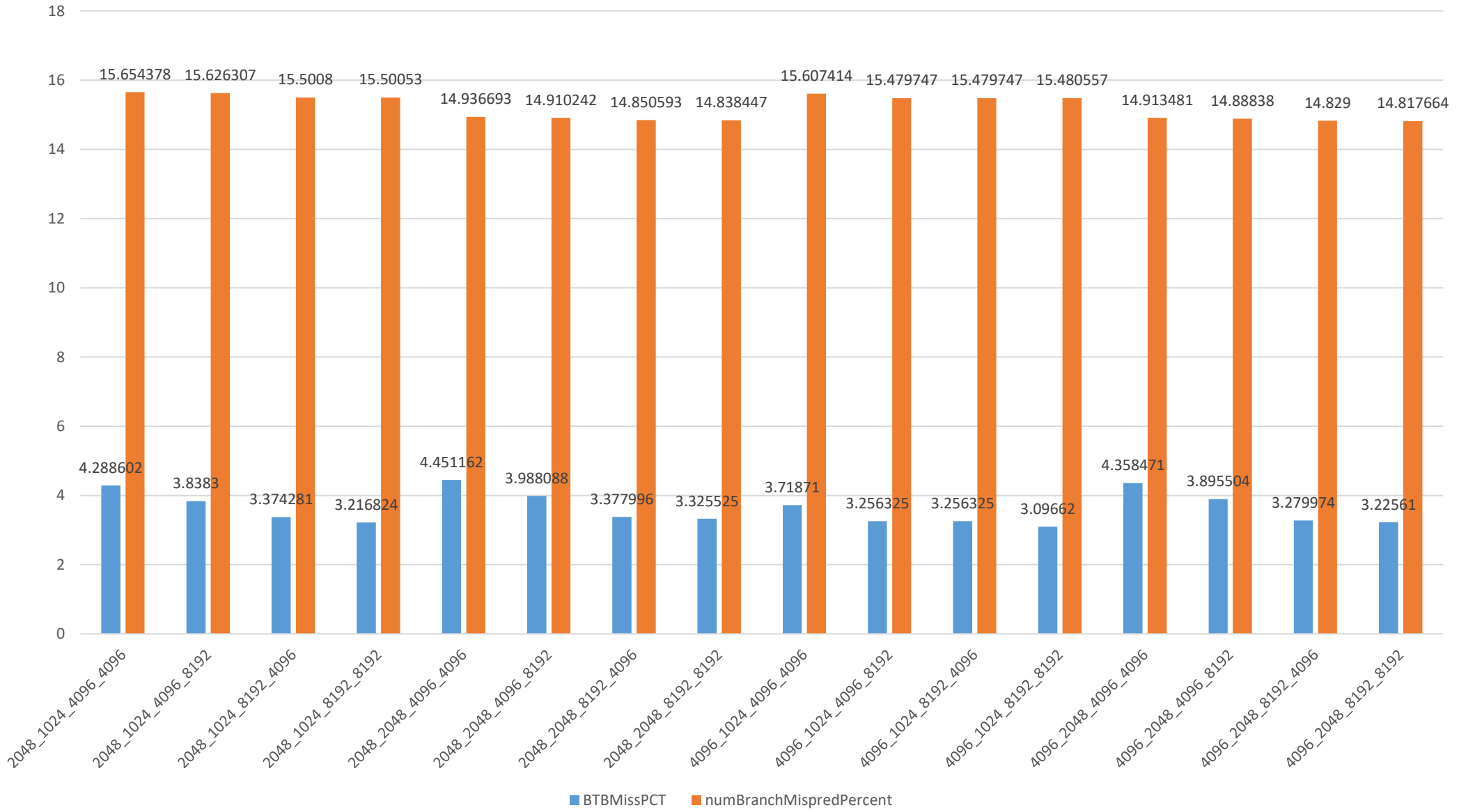
TournamentBP Exploration for 456.hmmer:



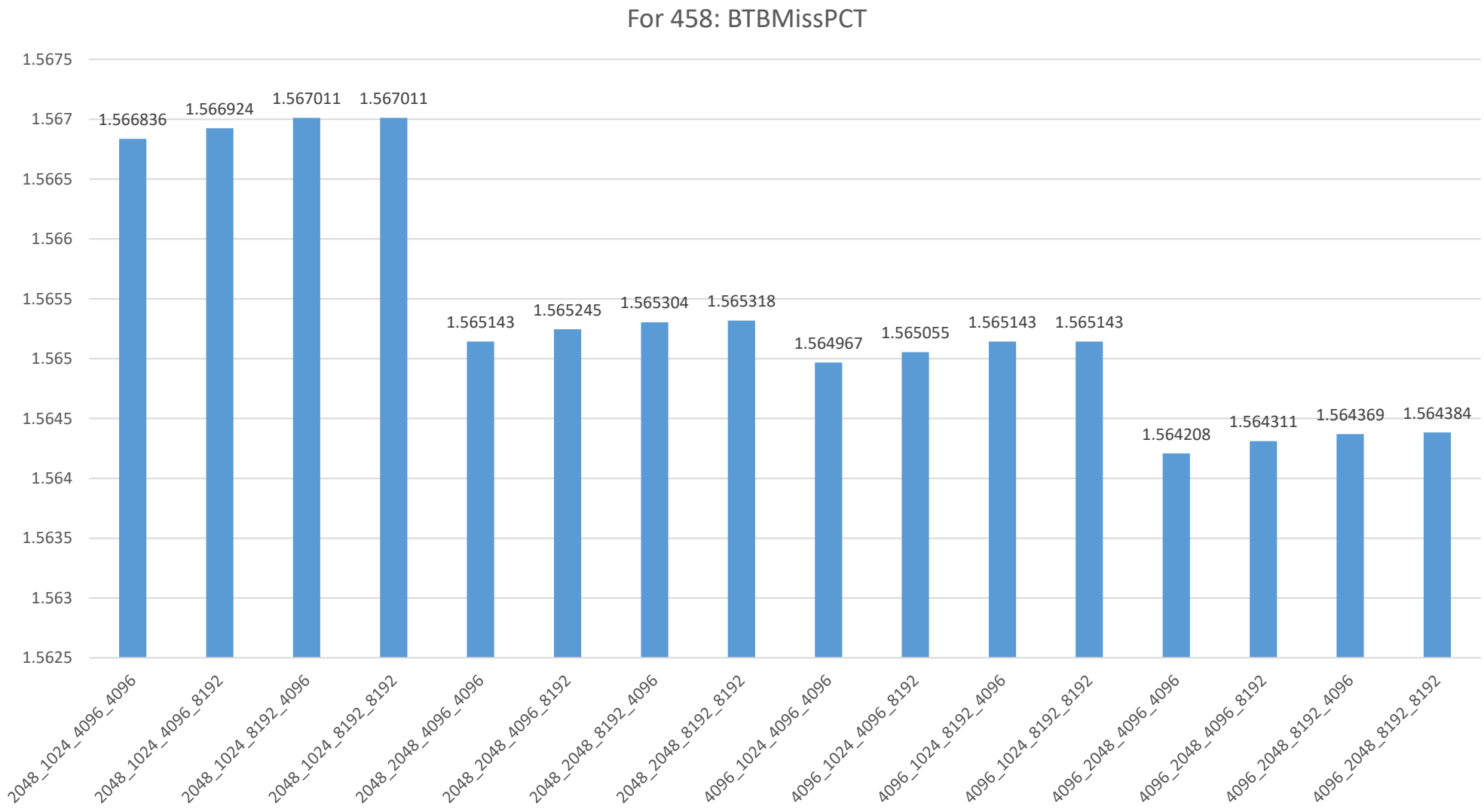
TournamentBP Exploration for 456.hmmer:



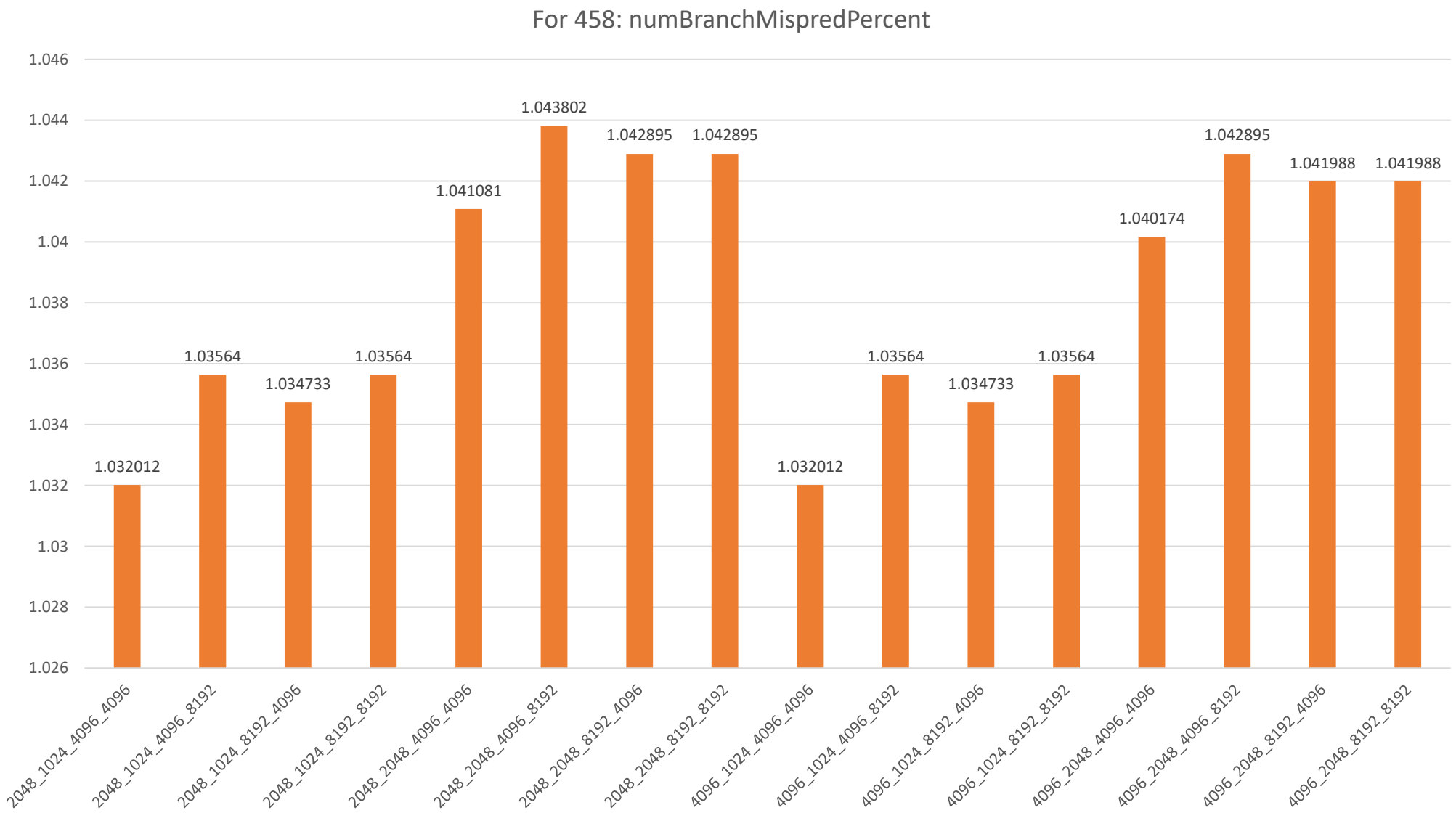
TournamentBP Exploration for 456.hmmer:



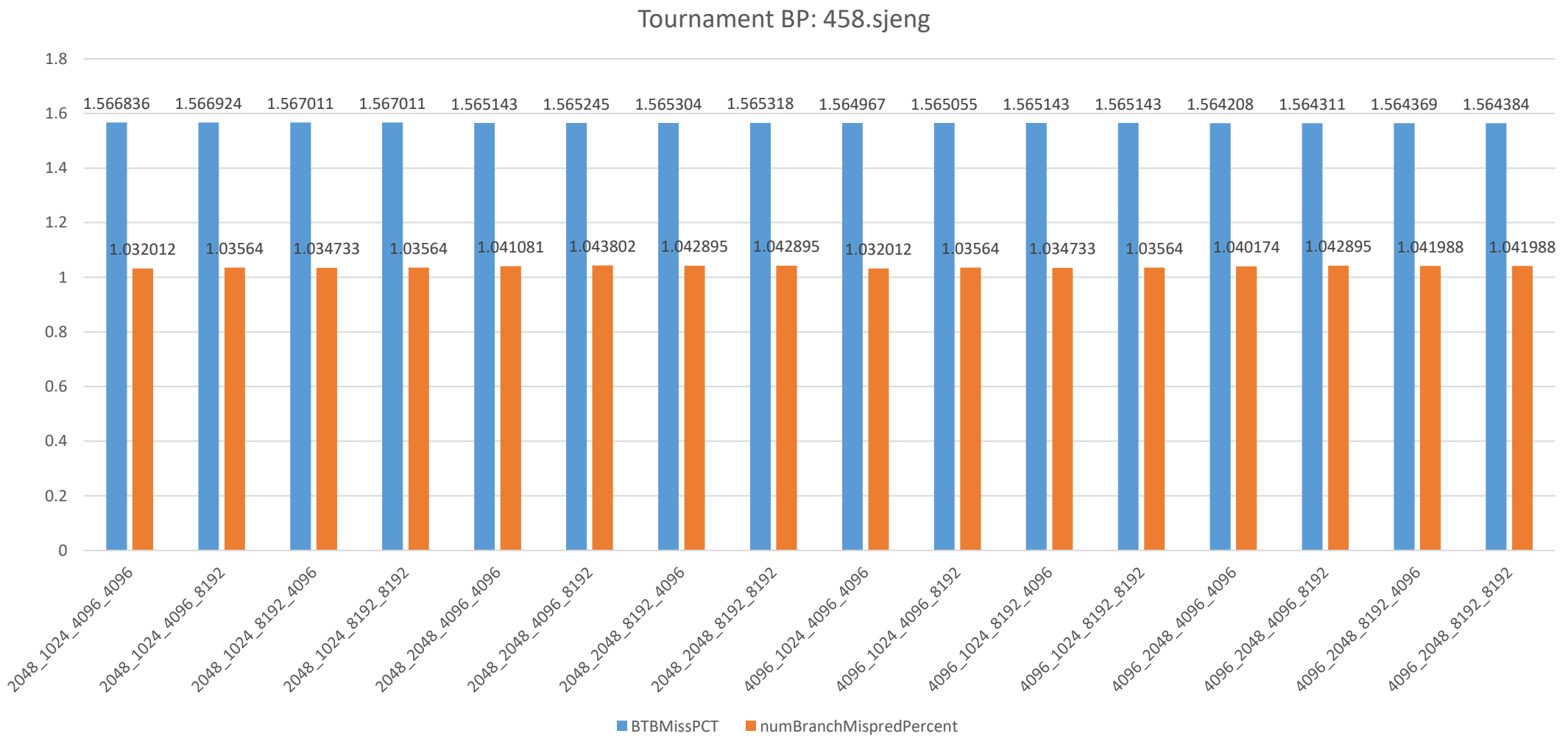
TournamentBP Exploration for 458.sjeng:



TournamentBP Exploration for 458.sjeng:



TournamentBP Exploration for 458.sjeng:



Observations Made From Graph For 456.hmmer:

- **Local Branch Prediction (BP):**

  When we enhance the BTBEntries and localPredictorSize, there's a slight decrease in both BTBMissPCT and numBranchMispredPercent.

- **BiMode Branch Prediction (BP):**

  If we increase BTBEntries, localPredictorSize, and globalPredictorSize, there's a slight variation in the beginning and significant decrease for BTBMissPCT and increase in numBranchMispredPercent.

- **Tournament Branch Prediction (BP):**

  When we increase BTBEntries, localPredictorSize, globalPredictorSize, and choicePredictorSize, we notice there's a slight variation in the beginning and at end BTBMissPCT and numBranchMispredPercent values remain relatively constant.

Observations Made From Graph For 458.sjeng:

- **For Local Branch Prediction (BP):**

  When we increase BTBEntries and localPredictorSize, there's a slight variations in both BTBMissPCT and numBranchMispredPercent

- **For BiMode Branch Prediction (BP):**

  If we increase BTBEntries, localPredictorSize, and globalPredictorSize, there’s a slight variation in the beginning and significant decrease for BTBMissPCT and increase in numBranchMispredPercent.

- **For Tournament Branch Prediction (BP):**

  Increasing BTBEntries, localPredictorSize, globalPredictorSize, and choicePredictorSize results in a consistent change in the linear trend for both BTBMissPCT and numBranchMispredPercent.

Conclusion:

- **Local Branch Prediction** seems to provide a relatively moderate but consistent improvement in both BTBMissPCT and numBranchMispredPercent across both benchmarks.
- **BiMode Branch Prediction** initially shows a varying impact but leads to a significant decrease in BTBMissPCT while increasing in numBranchMispredPercent.
- **Tournament Branch Prediction** presents a more consistent behavior across the given benchmark, maintaining stable trends in BTBMissPCT and numBranchMispredPercent.

➞ So, based on the above-mentioned observations, we can say that **Tournament Branch Prediction** could be considered relatively more favorable as it maintains a stable performance in terms of BTBMissPCT and numBranchMispredPercent across the different benchmarks'.

THANK YOU