ATHANASIOS DELIS 03117103

ΑΘΑΝΑΣΙΟΣ ΔΕΛΗΣ

2Η ΣΕΙΡΑ ΑΣΚΗΣΕΩΝ ΠΡΟΗΓΜΕΝΑ ΘΕΜΑΤΑ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

ANSWER GIVEN IN THE ENGLISH LANGUAGE AS MEAN OF PRACTISE FOR A GLOBALIZED WORKSPACE ENVIROMENT. IF ATTEMPT NOT SUFFICIENT FOR YOUR NEEDS, SEND ME AN EMAIL TO REWRITE THE ANSWER IN THE GREEK LANGUAGE.

CONTENTS

4.1\_STATISTICS

4.2\_N-bit PREDICTORS

4.3\_BTB

4.4\_RAS

4.5\_PREDICTORS\_COMPARISON

APPENDIX A

CODE\_FOR\_PLOTS

APPENDIX B

CODE\_FOR\_BTB

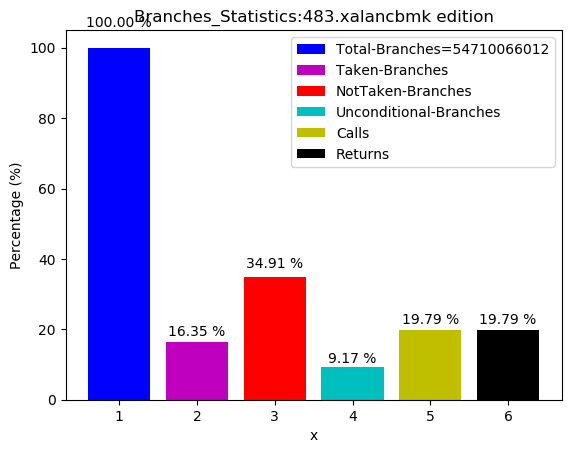
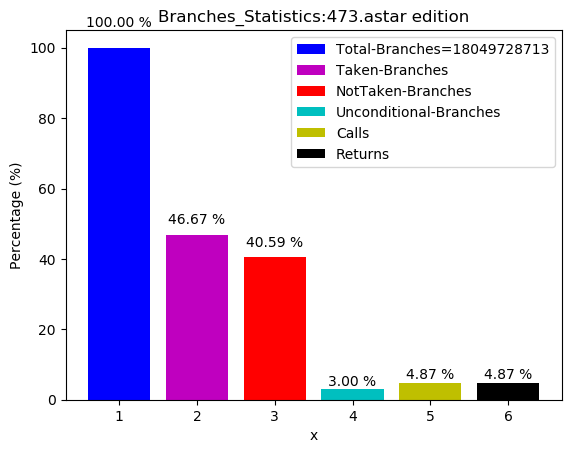
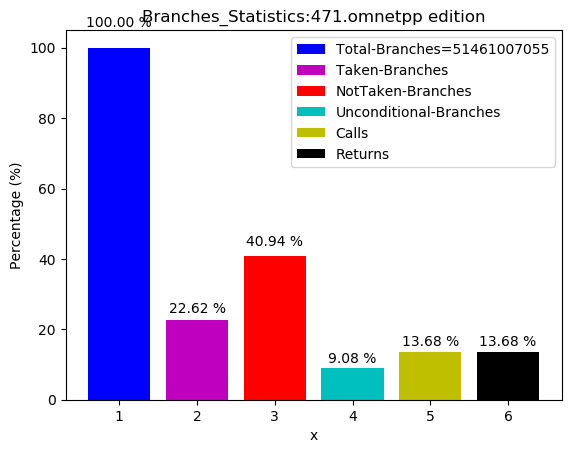
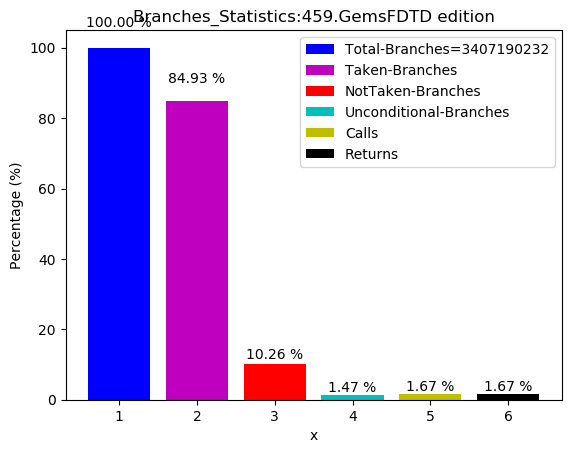
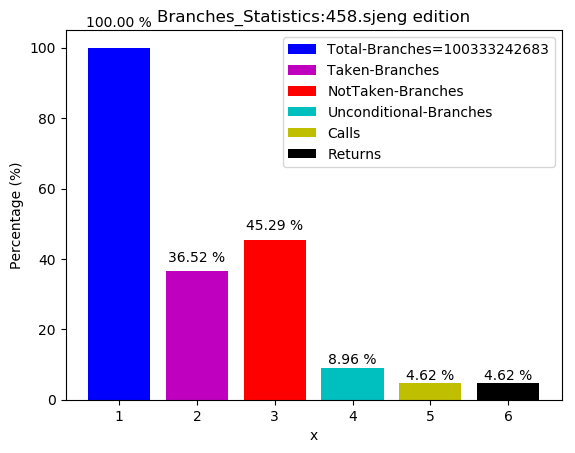
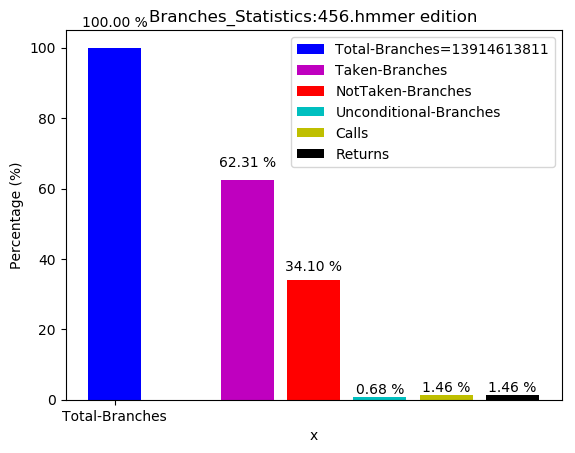
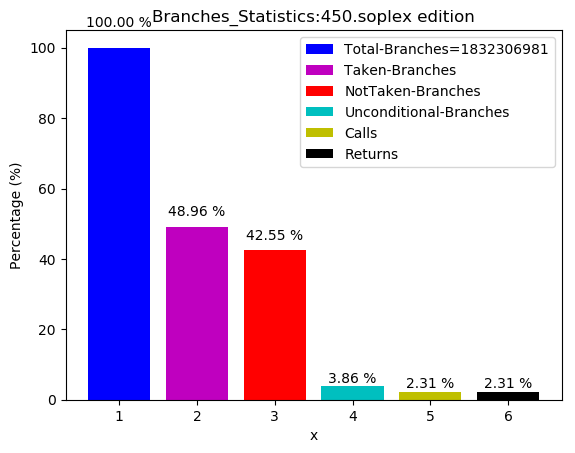
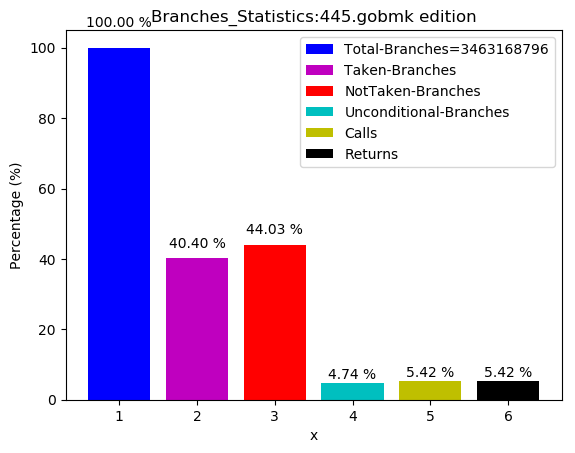
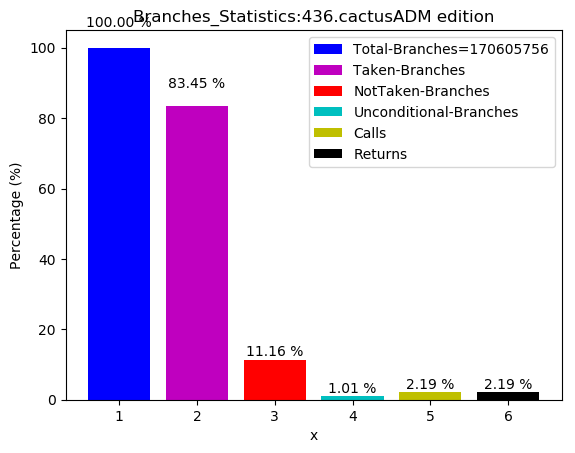
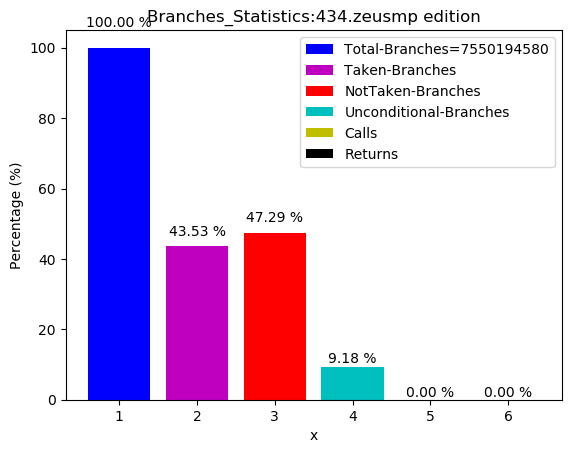
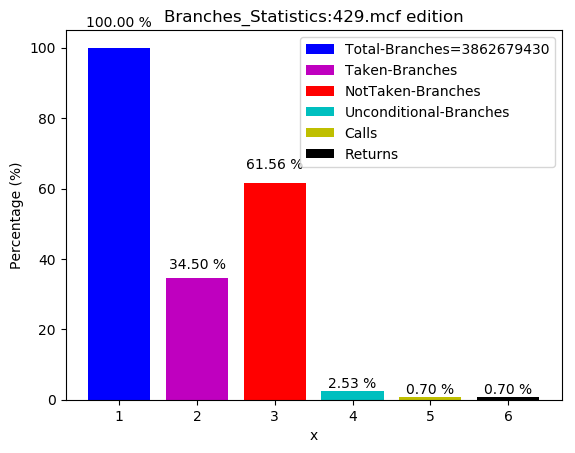
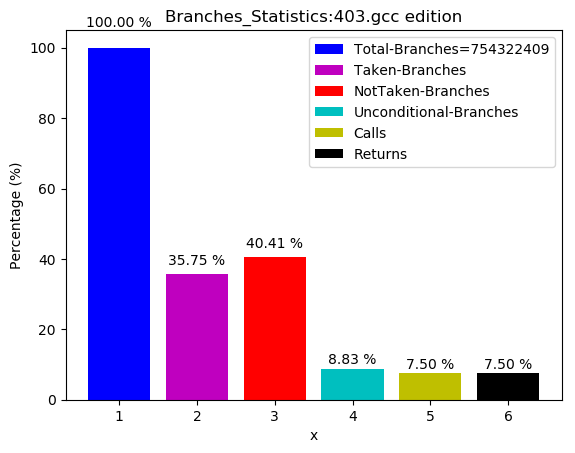
CODE\_FOR\_

APPENDIX C

CODE\_FOR\_COSTUM\_MADE\_PREDICTORS

4.1\_STATISTICS

THE MAIN PATERN WE ENCOUNTERED ACROSS ALL BENCHMARKS WAS THAT CONDITIONAL PREDICTIONS HAD GREATER NUMBERS IN COMPARISON WITH OTHER BRACHES CATEGORIES.



4.2\_N-bit PREDICTORS

AS IT WAS EXPECTED FROM THEORY IN SLIDES AND IN THE BOOK “COMPUTER ARCHITECTURE PATTERSON, HENESSY” A 2-BIT PREDICTOR HAS SIMILAR PREDICTION EFFICIENCY WITH A N-BIT PREDICTOR WITH MINIMAL DIFFERENTIATION

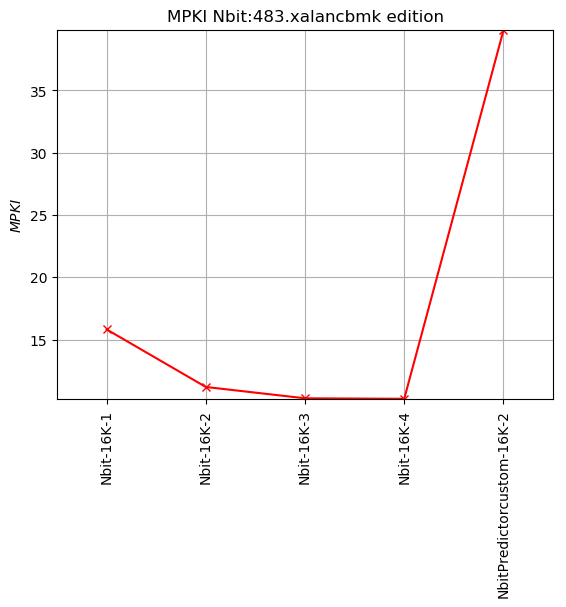
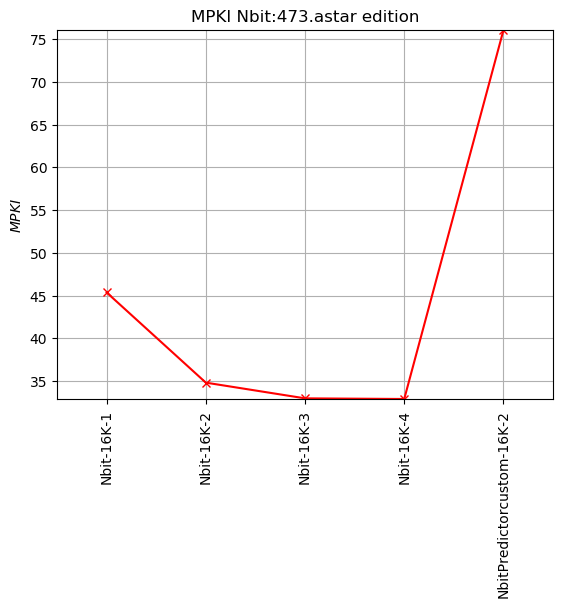
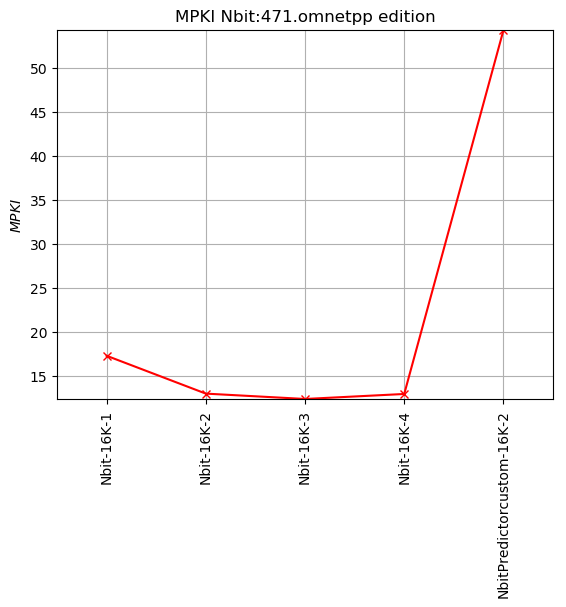
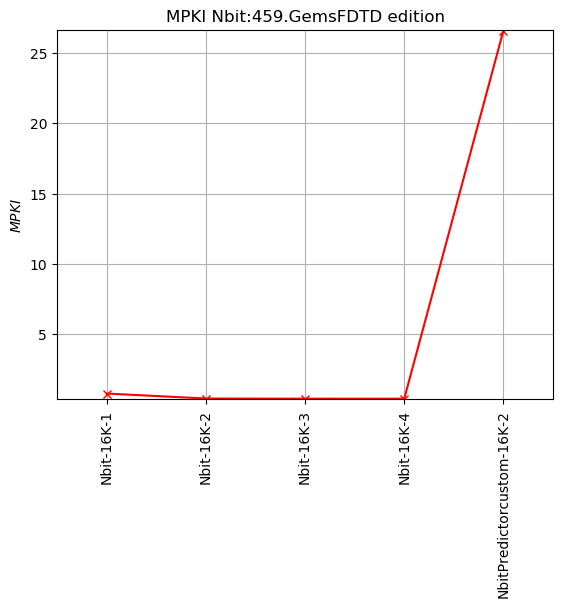
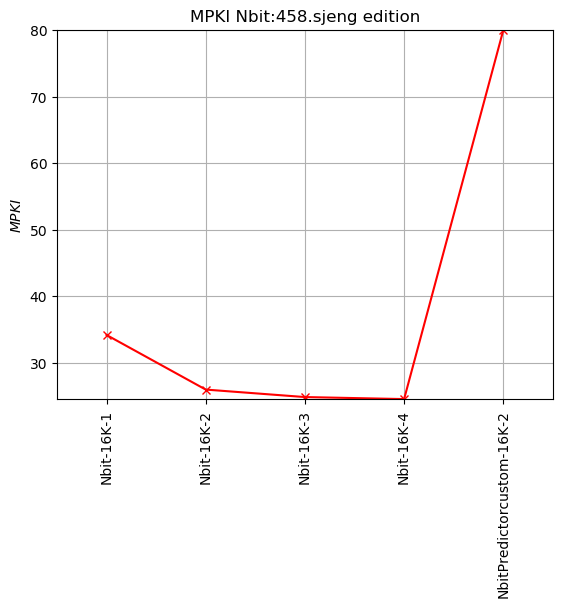
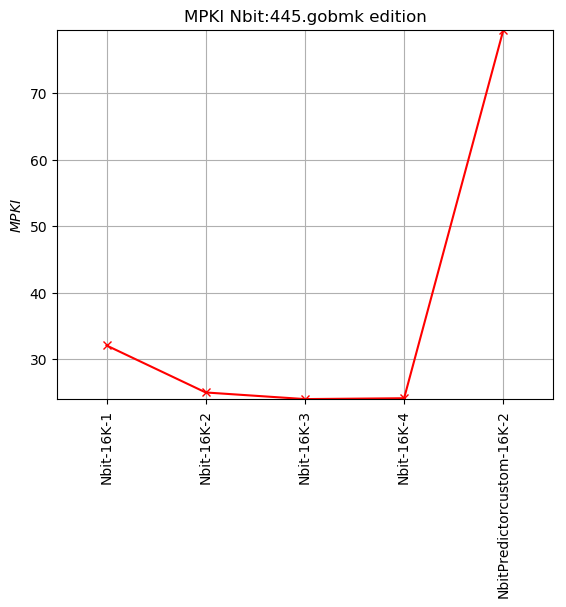
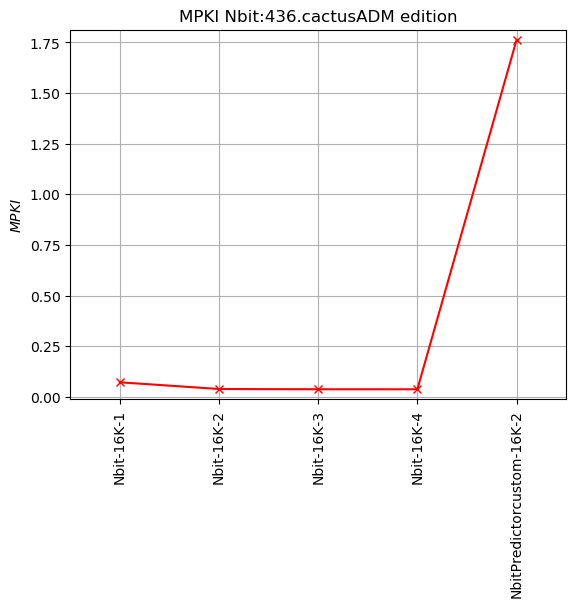
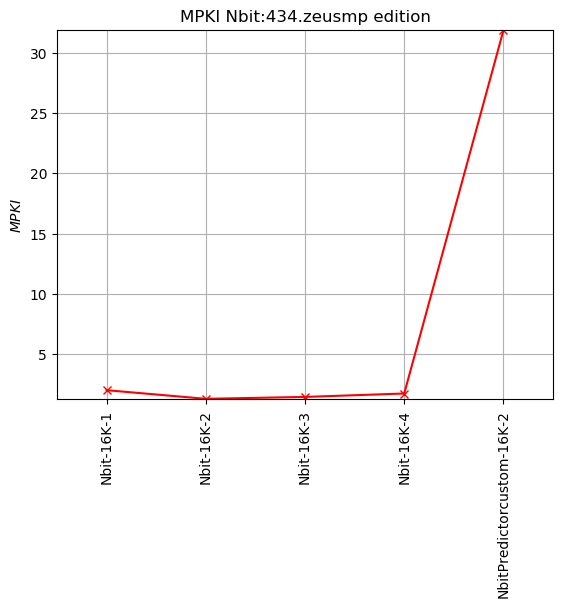
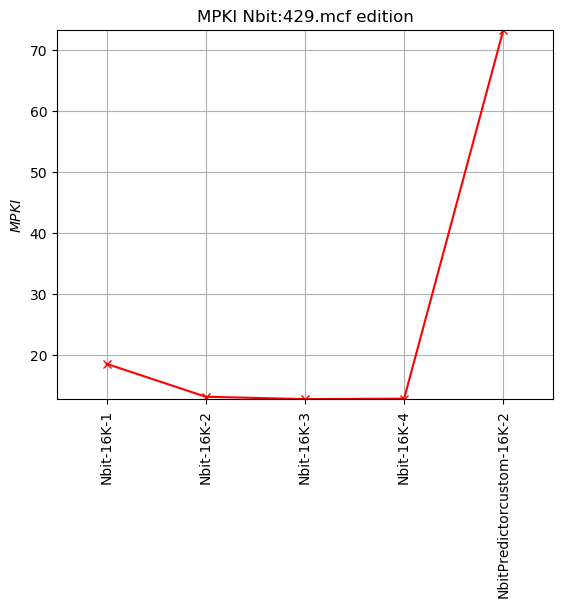
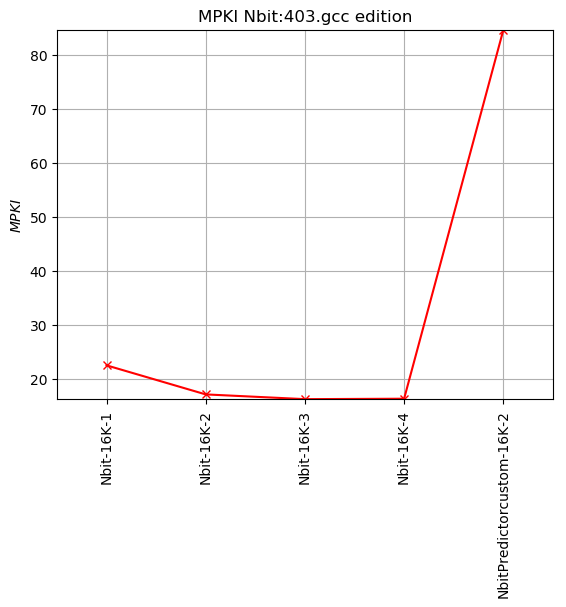
OUR COSTUM MADE PREDICTOR WAS NO BETTER THAN THE CLASSIC 2-BIT ONE. ACTUALLY IT WAS WORSE, WITH A FAR GREATER MISPREDICTION RATE.

IN THE FIRST IMPLEMANTATION WE KEEPT THE BHT AT A CERTAIN SIZE AND INCREMENTED PREDICTORS N BITS BY ADDING HARDWARE

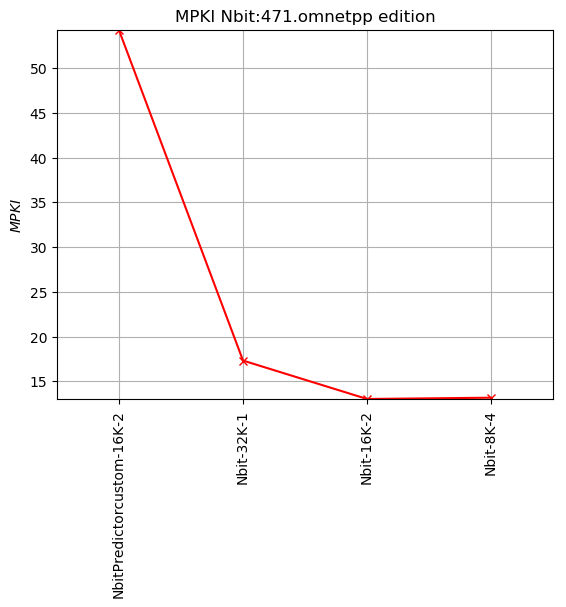
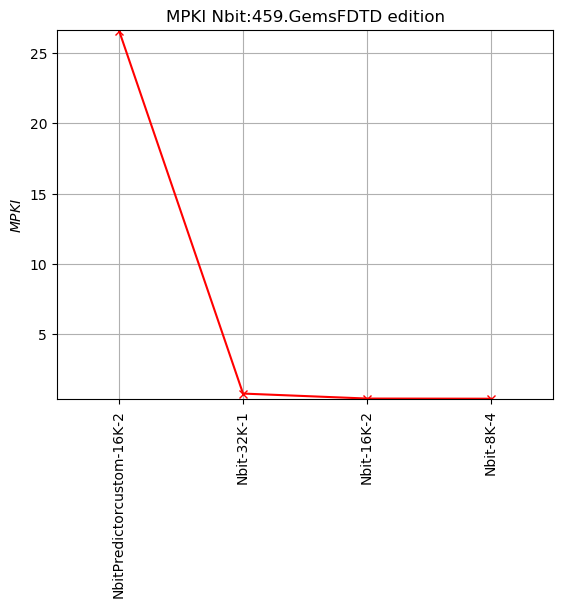
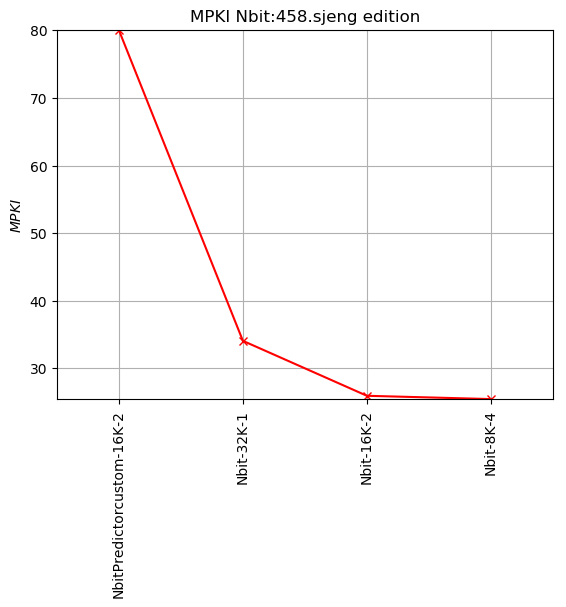
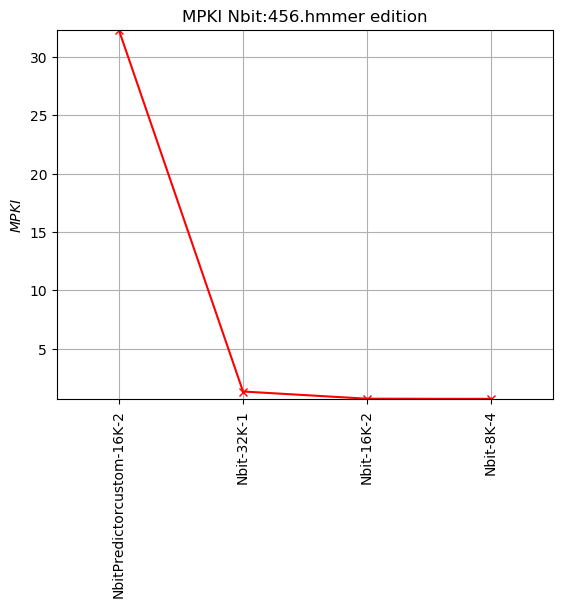
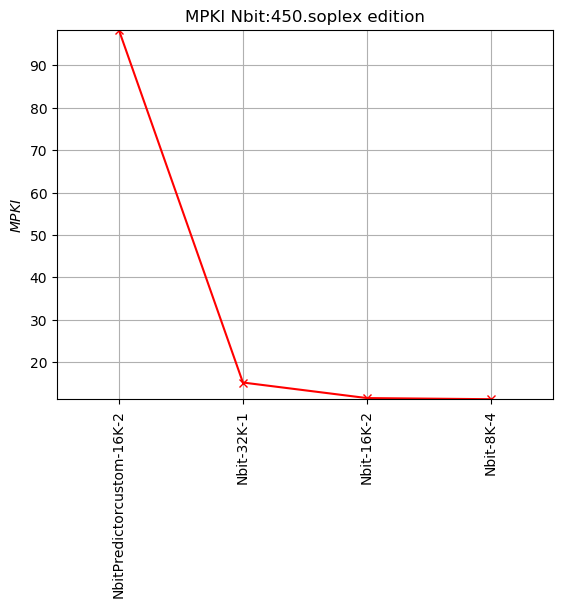
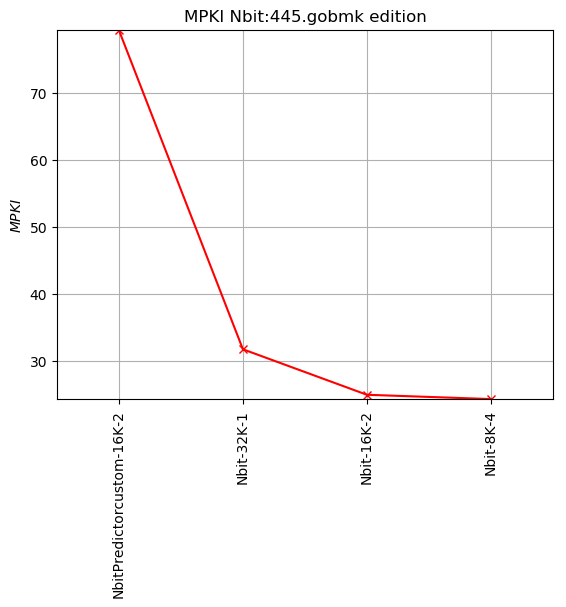
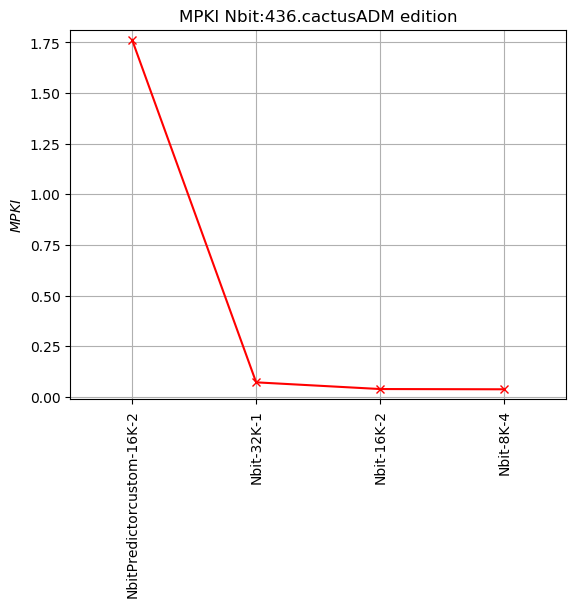
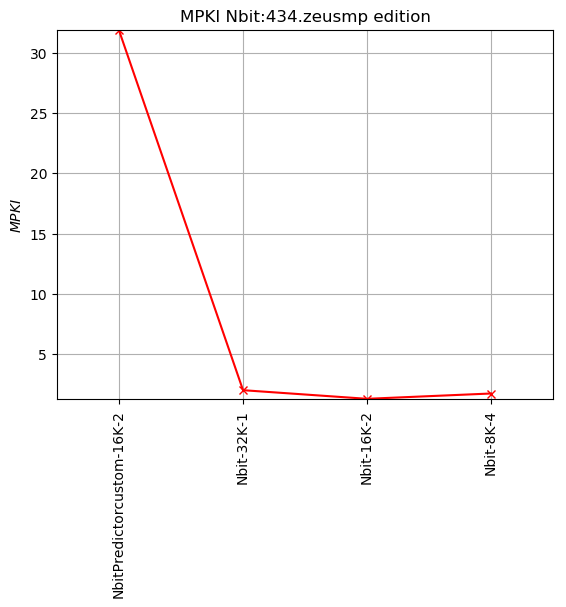
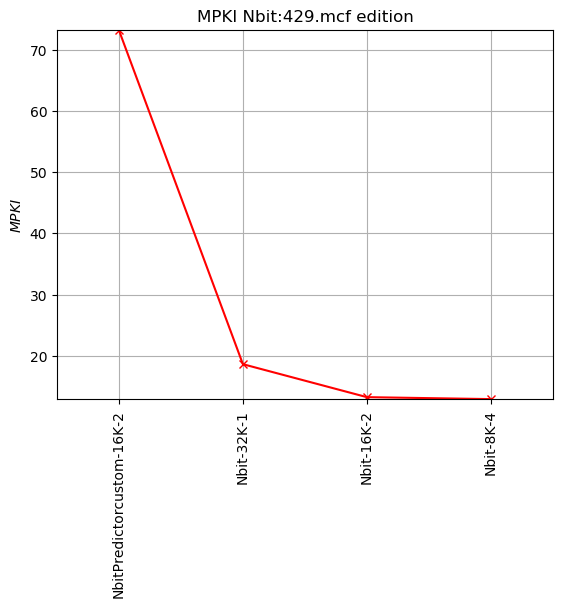
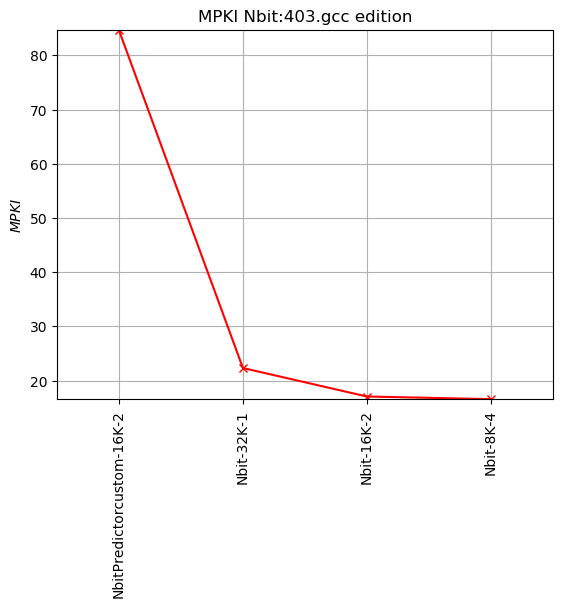
IN THE SECOND BY KEEPING STEADY SIZE IN HARDWARE WE INCREMENT BITS BY REDUCING ENTRIES

WITH THIS METHOD WE CONCLUDED THAT THE BEST PREDICTOR WAS THE 8K-4BITS, ACCORDING TO PLOTS ABSOSERVATION

4.2



4.2.1

4.3\_BTB

ACCORDING TO LECTURE SLIDES BTB IS A SMALL CACHE THAT SAVES TARGET ADRESS OF A BRACH COMMAND.

DURING THE IF STAGE , WE PREDICT FROM WHERE WE WILL BRING THE NEXT COMMAND AT THE SAME TIME WE BRING THE CURRENT COMMAND TO BE EXCUTED

BTB CONTAINS BOTH INSTRUCTION ADRESS AND PREDICTED ADRESS

AND SAVES ONLY ALREADY TAKEN BRANCHES AND JUMPS

HAVING IN MIND THE PICTURE FROM THE SLIDE 42:  
Branch-Target Buffer WE CREATED OUR OWN BTB AS A CLASS

IN CONTRAST WITH THE FIGURE WE MENTION WE USED A THREE-DIMENSIONAL MATRIX, WANTING TO REPRESENT WITH THE ADDITION OF THE 3RD DIMENSION THE ASSOSIATIVITY OF THE MATRIX, THAT IS GIVEN AS AN INPUT

OUR DIMENSION IN THE CODE REPRESENTED

1ST:THE PARTICULAR SET IN BTB THAT IT IS DESIDED BASED ON ENTRIES

2ND:A PARTICULAR REGISTRATION,WITH LENGTH DESIDED ACCORDING TO ASSOCIATIVE

3RD: INSTRUCTION ADRESS AND PREDICTED ADRESS

PREDICT METHOD RETURN TRUE IN OUR PREDICTION WAS CORRENT,IN OTHER WORDS TAKEN.

IN CASE OF NOT TAKEN BRANCH AND PREDICTION CORRECT, WE HAVE NO CHANGE

IN CASE OF TAKEN BRANCH AND PREDICTION FALSE, WE STORE THE PARTICULAR INCIDENCE TO BTB BUFFER

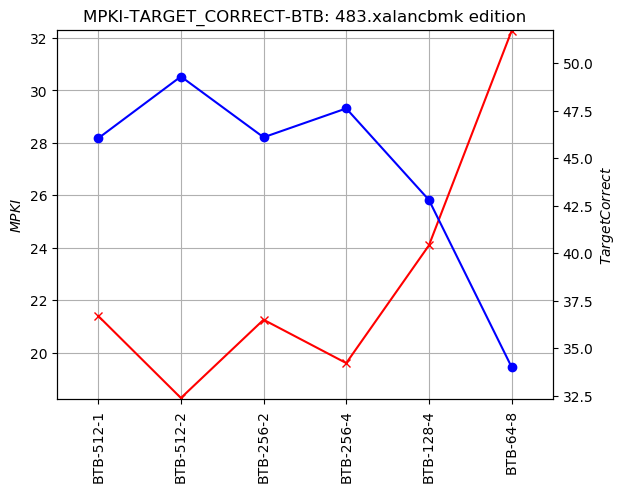
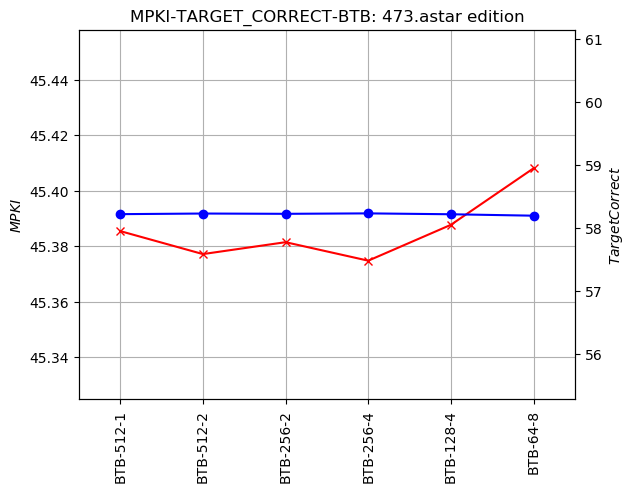
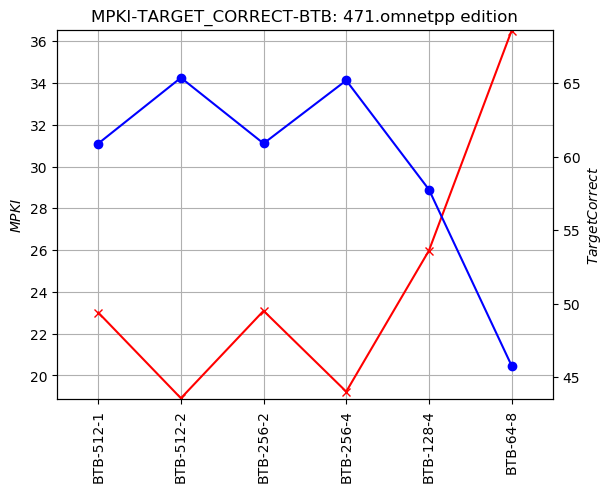
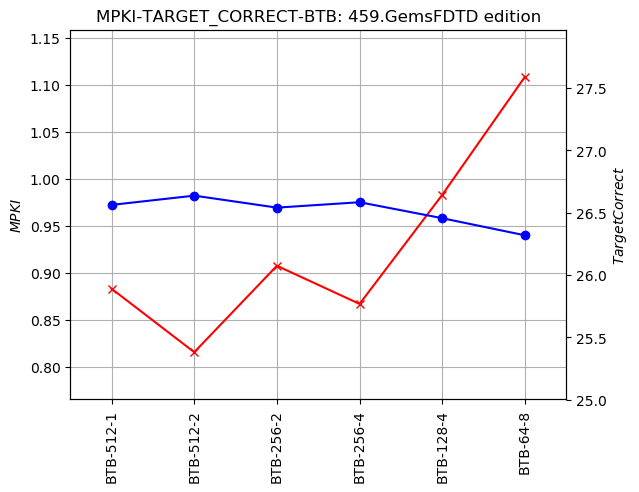
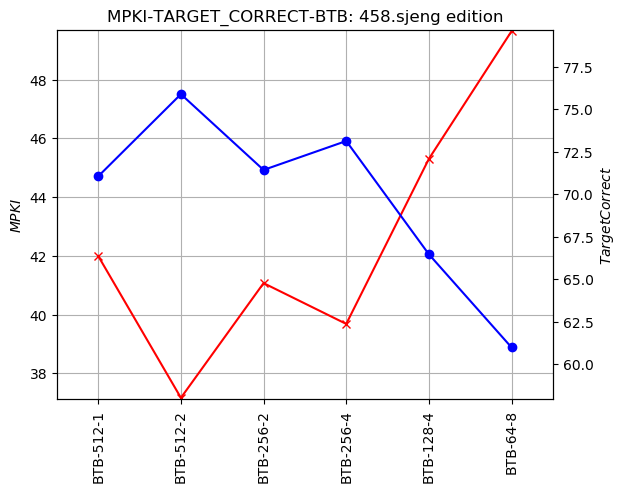
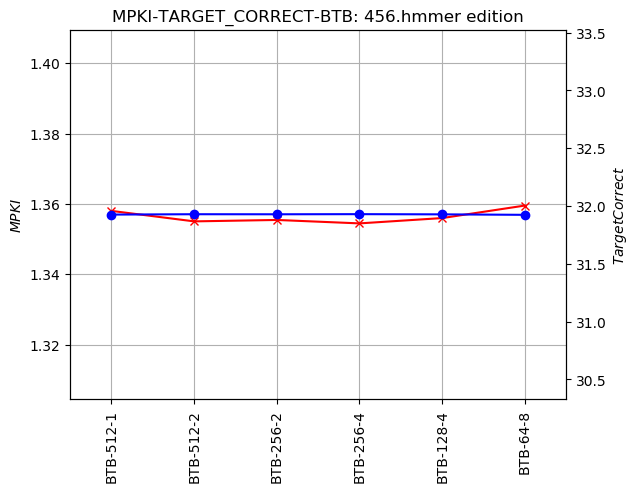
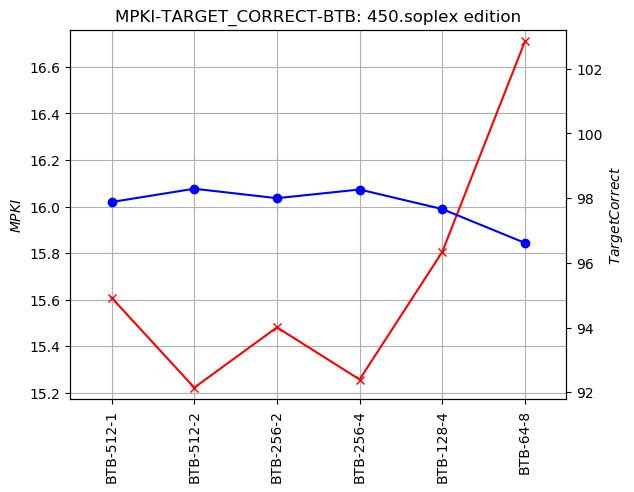
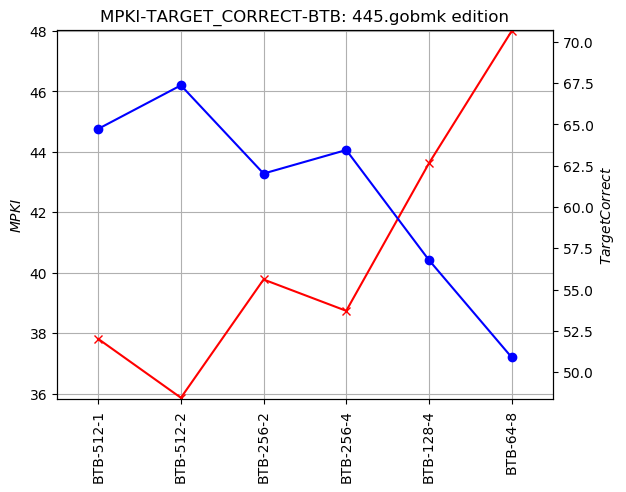
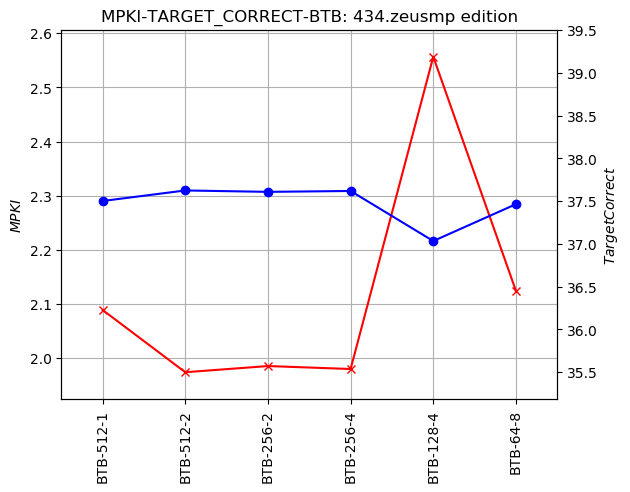
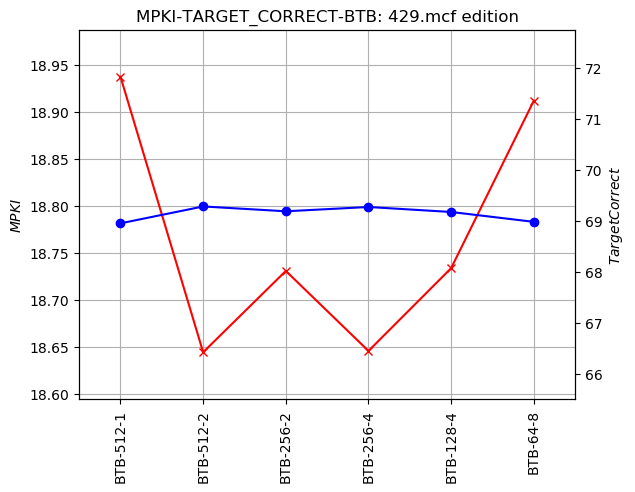
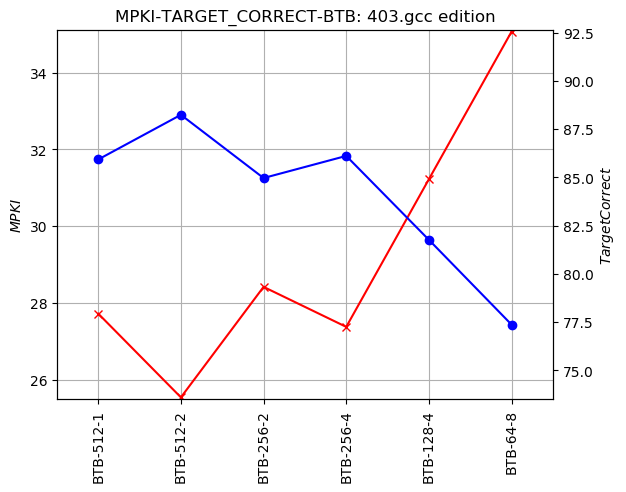
IN CASE OF NOT TAKEN BRANCH AND PREDICTION FALSE, WE HAVE TO DISCARD FROM OUR REGISTERS THIS PARTICULAR PIECE OF INFORMATION,THE REGISTRY THAT IS ACCOUNTABLE FOR THE FALSE PREDICTION

IN CASE OF TAKEN BRANCH AND PREDICTION CORRECT, WE RENEW THE PARTICULAR REGISTRY THAT IS TO BE ACCUSED OF OUR CORRECT PREICTION

OUR PLOTS PRESENT BOTH MPKI AND TARGET CORRECT/1000INSTRUCTIONS

DIRECT MAPING HAS THE GREATEST OF MISS RATES

FROM ASSOSIATIVE 2 AND BEYOND CHANGE WAS MINIMAL, ALTHOUGHT SAMEWHAT POSITIVE AS TO GREW MORE. THAT IS INTERPRETED AS MISS REDUCTION DUE TO CONFLICTS REDUCTION AND A SMALL INCREMENT IN CORRECT TARGET VALUES.



4.4\_RAS

RETURN ADDRESS STACK

IT IS INPLEMENTED AS A FILO STACK

IN THIS PART OF THE EXERSICE WE WILL EXAMINE ANOTHER ASPECT WE ENCOUNTERD DURING STATISTICS ACCUMULATIONS, AND IN PRESICION RETURNS.

IT IS A FAIR ASSUMPTION TO BE MADE THAT BENCHMARKS WITH GREATER PERCENTAGE OF RETURNS WILL PRESENT DIFFERANTIATIONS AT A GREATER EXTENT

IN PARTICULAR WE WILL MAKE A COMPARISSON BETWEEN THIS TWO TO PROVE OUR ASSUMPION

403 Returns: 56600055 and 434 Returns: 36090

Graph of the second was static close to zero from the begining

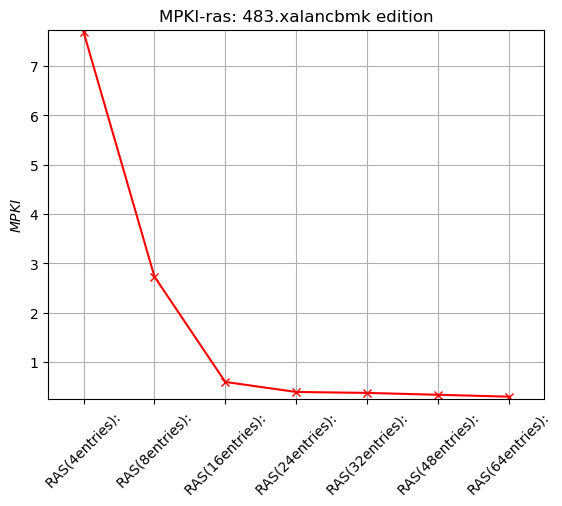
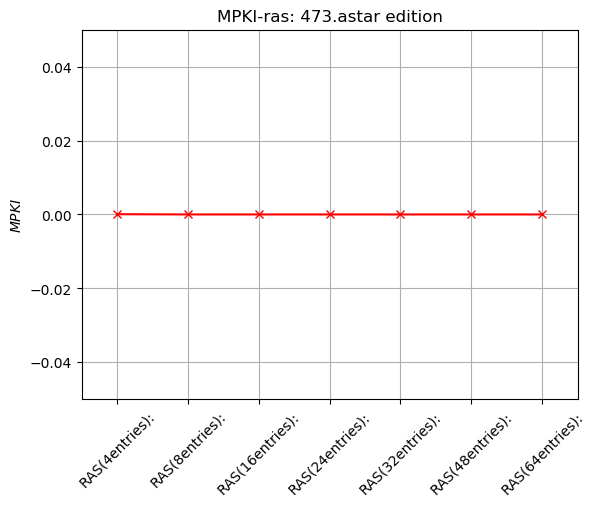
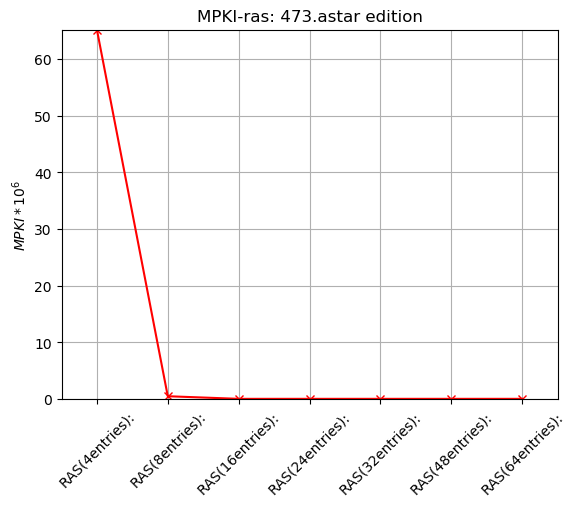
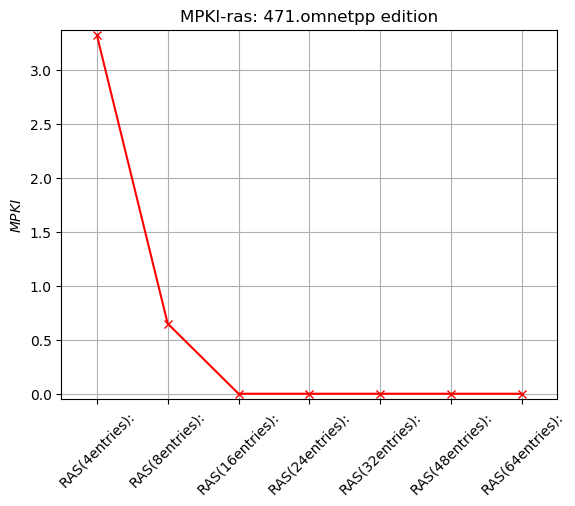
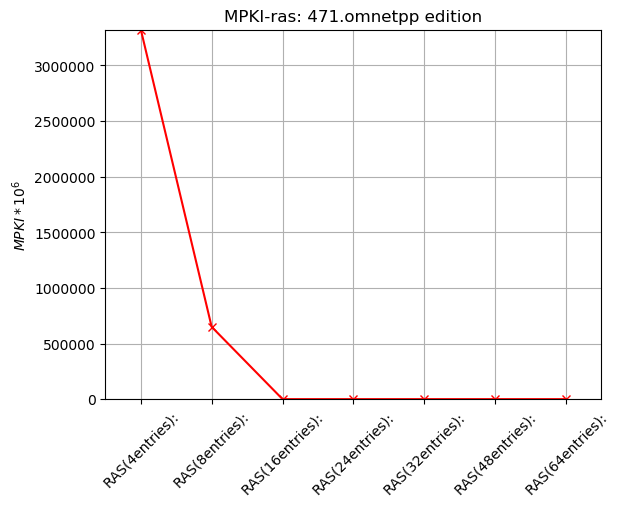
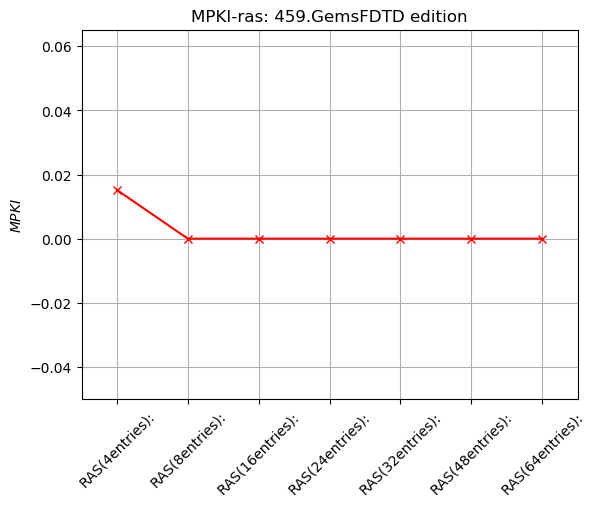
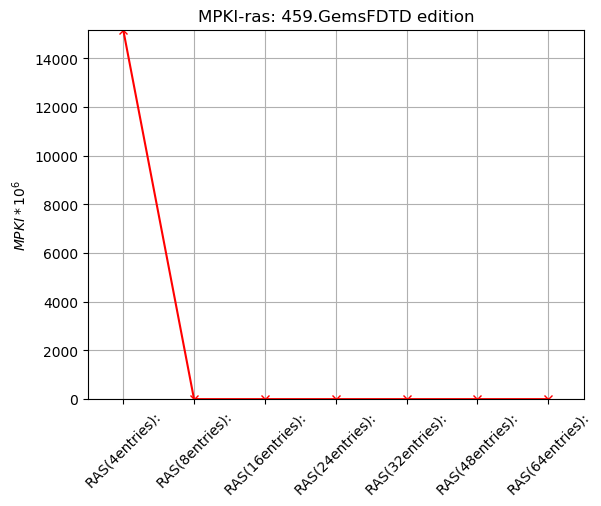
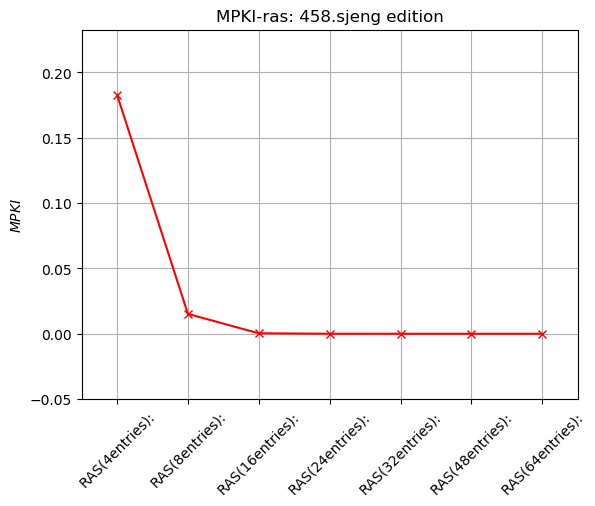
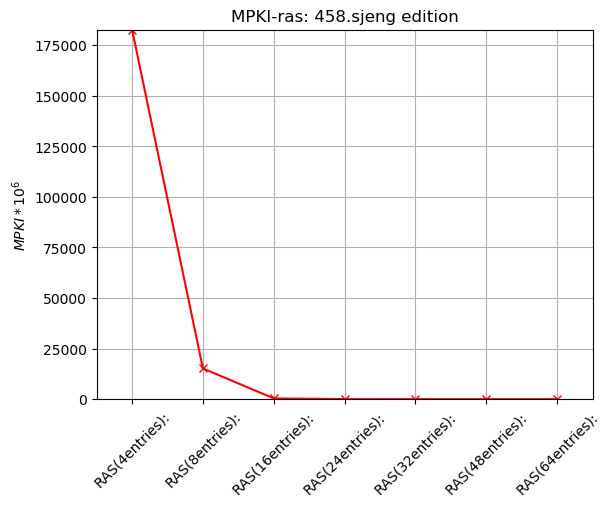
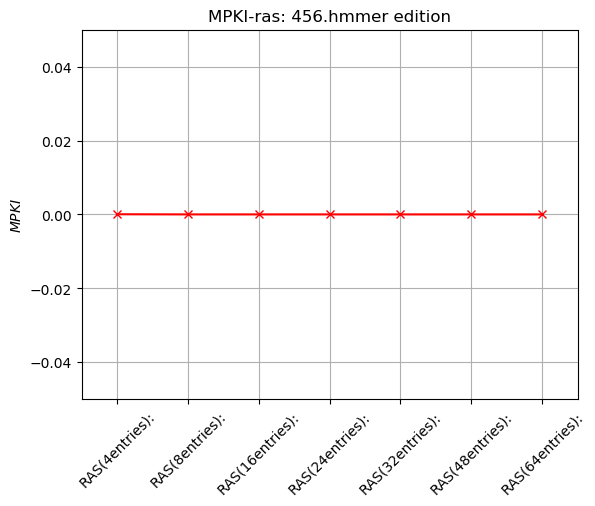
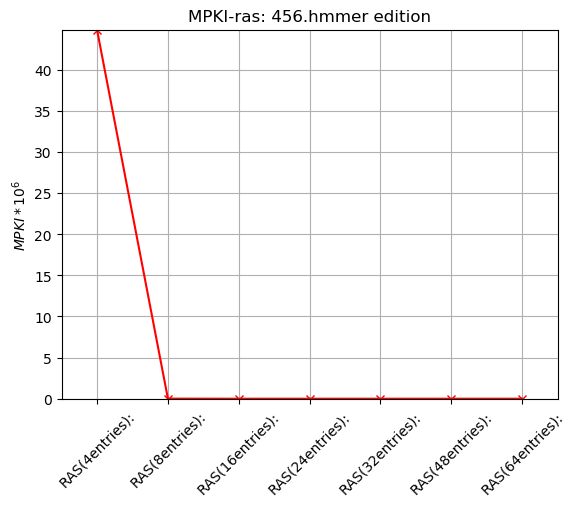
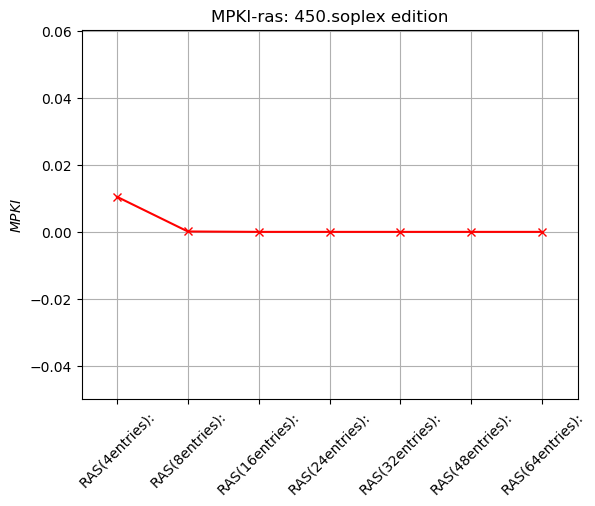
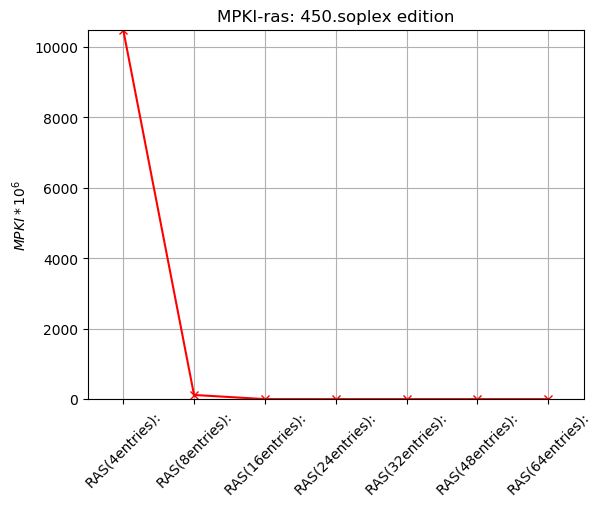
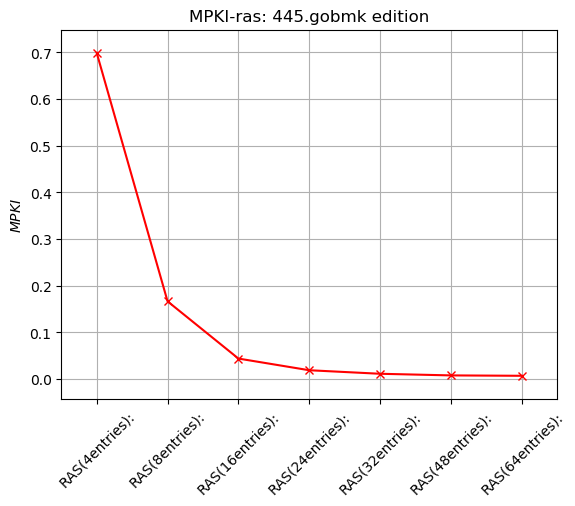
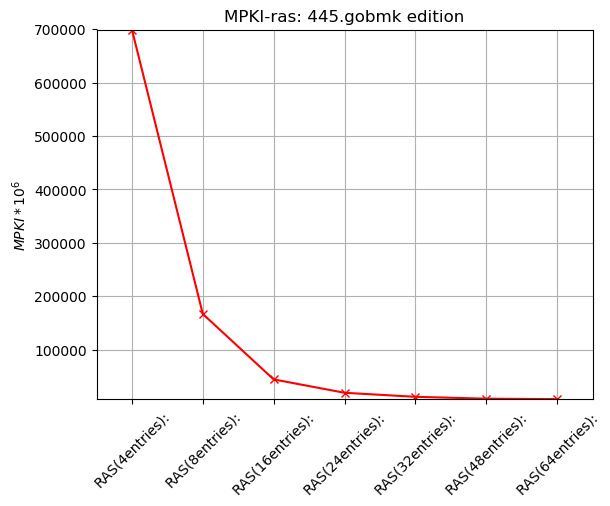
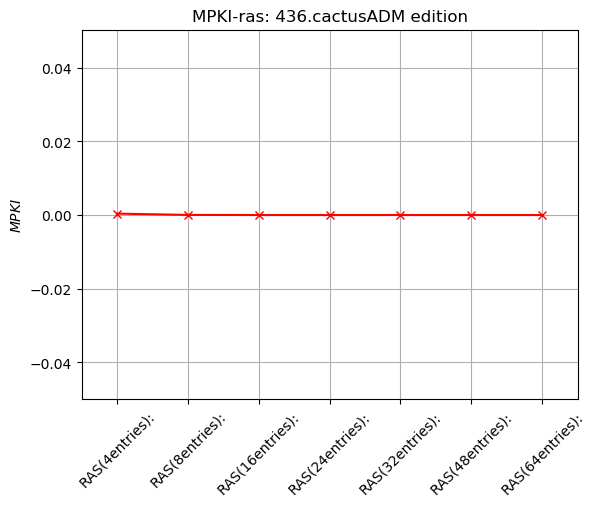
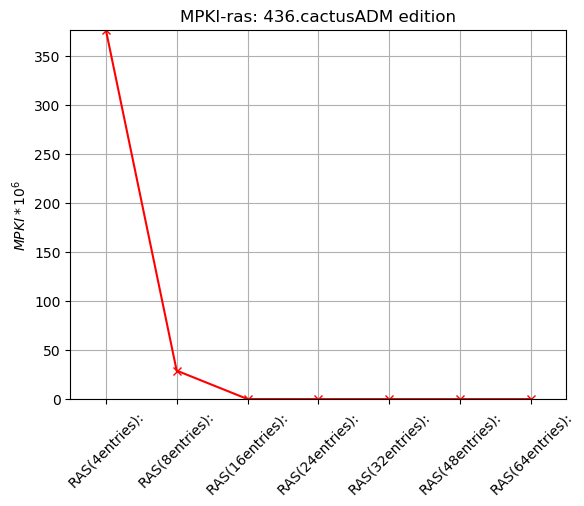
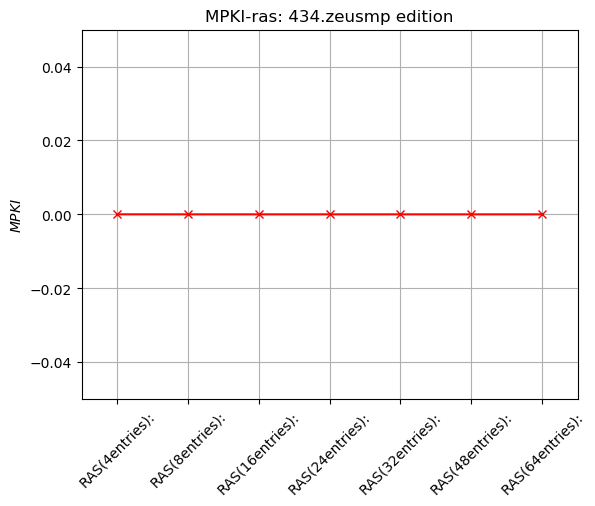
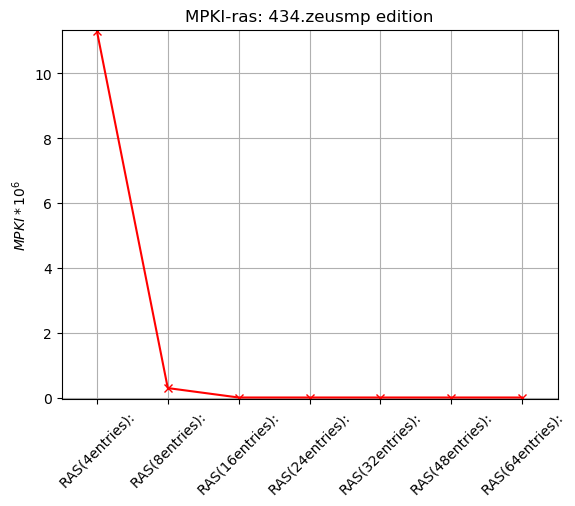
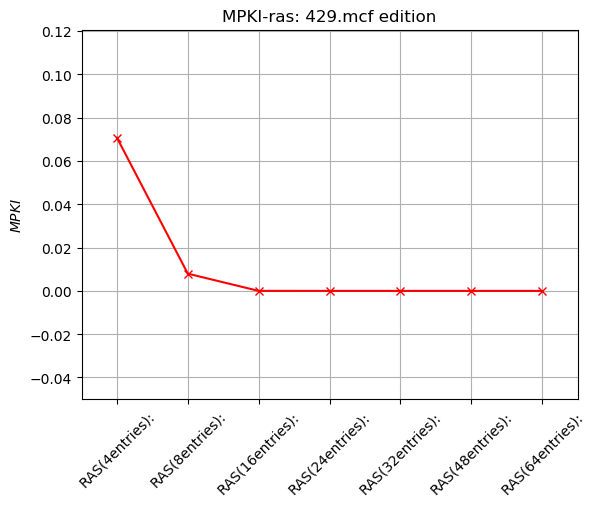
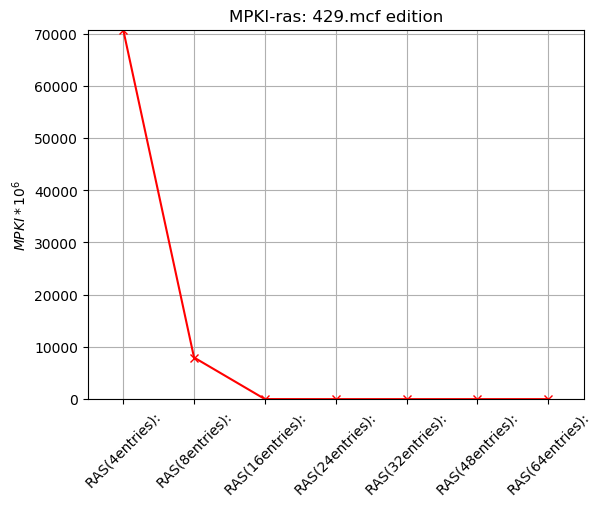
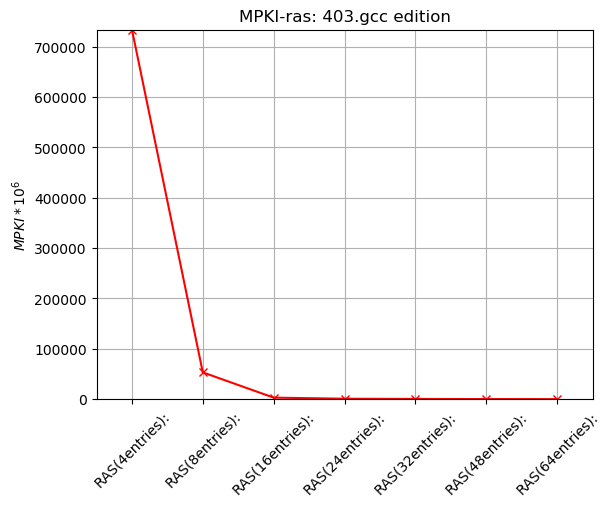
THE RESULTS WHERE ASTOUNDING AS IT SEEMS THAT AFTER ONLY 4 REGISTRIES ALMOST ALL BENCHMARCKS GAVE A MPKI APPROCING TO 0

FROM 1 TO 2 REGISTRIES WE OBSERVED THE GREATEST RATE OF CHANGE

AND AT ONE REGISTRY WE ENCOUNTERED MOST OF THE MISSES

RESULTS SIMILAR TO FIGURES IN SLIDES THAT REFERD TO RAS, THEORY PROVED BY OBSERVATION

MPKI\*100^6 IS PRESENTED FOR BETTER RESOLUTION OF THE PHENOMENONS



4.5\_PREDICTORS\_COMPARISON

COMMENTS ON CODES DESIGN:

MOST OF THE CODE BOTH FOR PLOTS AND FOR PREDICTORS IT IS BASE ON THE PREXISTING CODE FOR THE N-BIT PREDICTOR AND ITS .PY PLOT COUNTER PART.

I WILL GIVE AN EXPLANATION ONLY FOR A RATHER STRANGE BUT EFFICIENT PIECE OF CODE , THAT DOESN’T APPEAR IN THE PREXISTING PIESES BUT WAS NEED FOR HISTORY AND TOURNAMENT PREDICOTRS

THE PURPOSE OF THIS CODE IS TWO USE INDEXING FOR CONNECTING THE BHT MATRIX WITH THE PHT MATRIX.

ALL THIS WAS WRITTEN HAVING IN MIND THE PROCEDURE PRESENTED IN THE PICTURE Local-History Two-Level Predictor SLIDES PAGE 37 AND Global-History Two-Level Predictor PAGE 34

I PRESENT SEPERTLY THE FOLLOWING PIECE OF CODE AND THE ALPHA PREDICTOR. EVERYTHING ELSE CAN BE FOUND IN APPENDICIES

memset(PHT, 0, PHTentries \* sizeof(\*PHT));

this->BHR = 0;

/\* Need (PHTindex - BHRcntr) bits shifted BHRcntr bits left \*/

this->PHTmod = 1 << (this->PHTindex - this->BHRcntr);

inside predict

PHTinx = (ip % this->PHTmod) << this->BHRcntr;

/\*\*

\* 1 << BHRcntr = max number ( + 1) for a BHRentry.

\* So with mod we restrict the non zero bits we get

\* from the BHR, which are at most BHTcntr.

\*/

PHTinx = PHTinx | (BHR % (1 << this->BHRcntr));

class ALPHA : public BranchPredictor {

private:

unsigned int PHTindex, PHTcntr;

unsigned int BHRcntr;

unsigned int PHTentries;

unsigned long long int \*PHT;

unsigned long long int BHR;

unsigned int PHTmod;

unsigned int COUNTER\_MAX;

bool localPred, globalPred;

//NbitPredictor \*nbit;

LocalHistoryTwoLevel \*local;

GlobalHistoryTwoLevel \*global;

public:

ALPHA(unsigned int PHTindex,

unsigned int PHTcntr,

unsigned int BHRcntr,

unsigned int PHTLindex,

unsigned int PHTLcntr,

unsigned int BHTLindex,

unsigned int BHTLcntr,

unsigned int PHTGindex,

unsigned int PHTGcntr,

unsigned int BHRGcntr

) : BranchPredictor() {

this->PHTindex = PHTindex;

this->PHTcntr = PHTcntr;

this->BHRcntr = BHRcntr;

this->PHTentries = 1 << this->PHTindex;

this->PHT = new unsigned long long int [PHTentries];

memset(PHT, 0, PHTentries \* sizeof(\*PHT));

this->BHR = 0;

/\* Need (PHTindex - BHRcntr) bits shifted BHRcntr bits left \*/

this->PHTmod = 1 << (this->PHTindex - this->BHRcntr);

/\* For Nbit PHT predictor \*/

COUNTER\_MAX = (1 << this->PHTcntr) - 1;

//nbitPred = false;

localPred = false;

globalPred = false;

//nbit = new NbitPredictor(nbitIndex, nbitCntr);

global = new GlobalHistoryTwoLevel(PHTGindex, PHTGcntr, BHRGcntr);

local = new LocalHistoryTwoLevel(PHTLindex, PHTLcntr, BHTLindex, BHTLcntr);

}

~ALPHA() {

delete [] PHT;

}

virtual bool predict(ADDRINT ip, ADDRINT target) {

//nbitPred = nbit->predict(ip, target);

//I predict for P0 and P1

globalPred = global->predict(ip, target);

localPred = local->predict(ip, target);

//custumize my indexes

unsigned int PHTinx;

PHTinx = (ip % this->PHTmod) << this->BHRcntr;

PHTinx = PHTinx | (BHR % (1 << this->BHRcntr));

//use the truth table according to P0 and P1 values

if ((PHT[PHTinx] >> (this->PHTcntr - 1)) > 0) {

return globalPred;

} else {

return localPred;

}

return false;

}

virtual void update(bool predicted, bool actual, ADDRINT ip, ADDRINT target) {

//nbit->update(nbitPred, actual, ip, target);

local->update(localPred, actual, ip, target);

global->update(globalPred, actual, ip, target);

/\* For meta \*/

unsigned int PHTinx;

PHTinx = (ip % this->PHTmod) << this->BHRcntr;

PHTinx = PHTinx | (BHR % (1 << this->BHRcntr));

BHR = BHR << 1;

if(actual) {BHR++;}

if (localPred != globalPred) {

if (localPred == actual && PHT[PHTinx] > 0)

PHT[PHTinx]--;

else if (globalPred == actual && PHT[PHTinx] < COUNTER\_MAX)

//BHR++;

PHT[PHTinx]++;

}

updateCounters(predicted, actual);

}

virtual string getName() {

std::ostringstream stream;

stream << "ALPHA-" << "PHT-" << pow(2.0, double(PHTindex)) / 1024.0 << "K-" << PHTcntr << "-"

<< "BHR-" << BHRcntr << local->getName() << "-" << global->getName();

return stream.str();

}

};

#endif

CONCLUSIONS:

IN SYMPHONY WITH THE THEORY IN SLIDES AND IN THE “PATTERSON, HENNESY, COMPUTER ARCHITECTURE” BOOK TOURMENT PREDICTORS PROVED TO BE THE MOST EFFICIENT ONES.

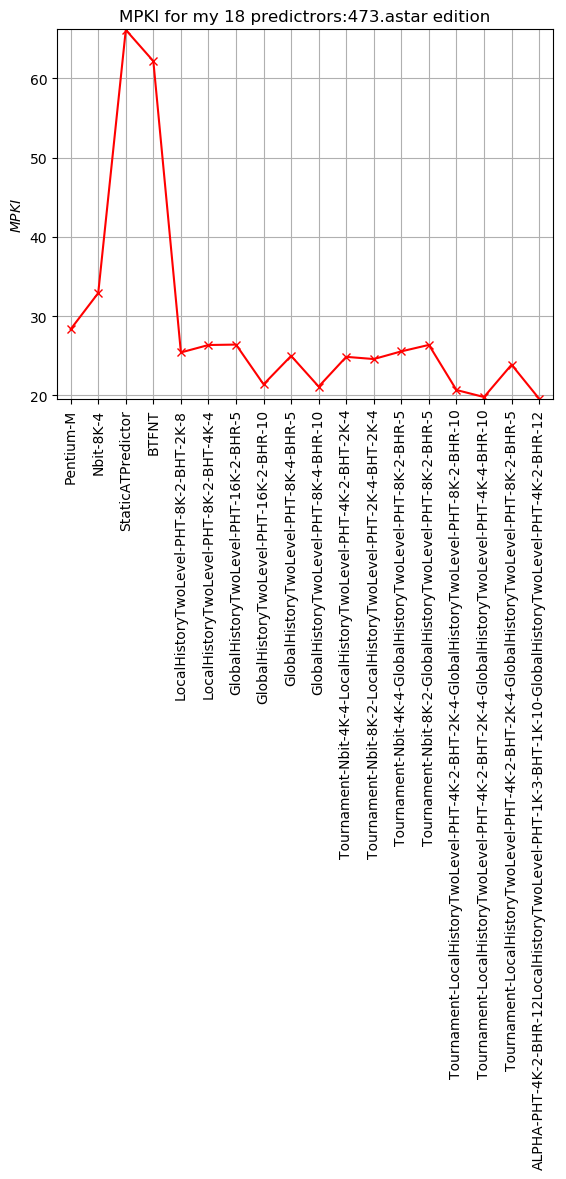
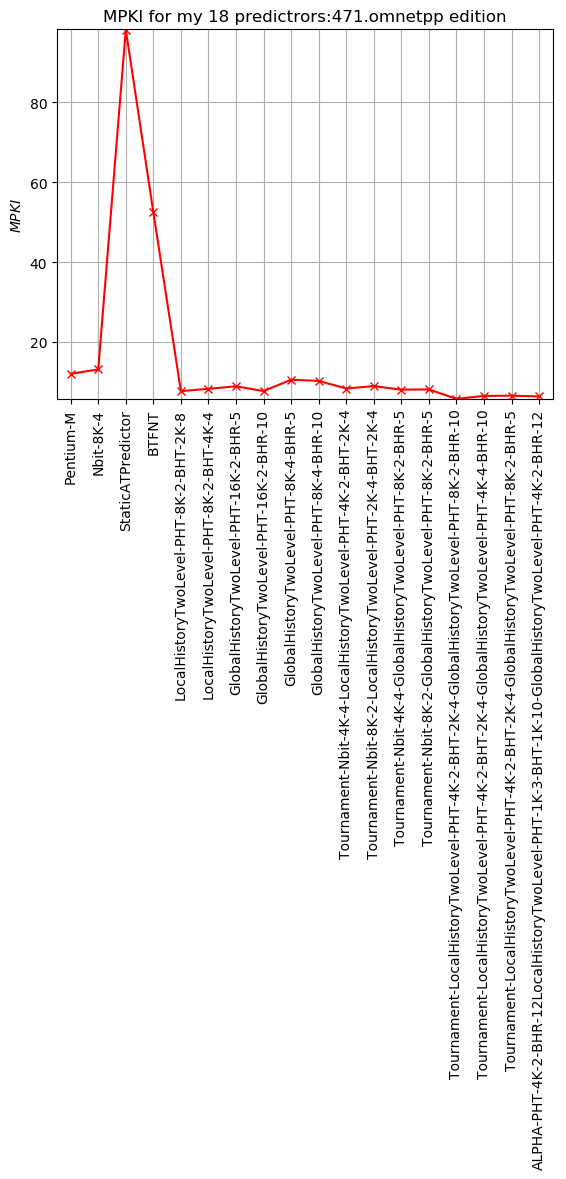
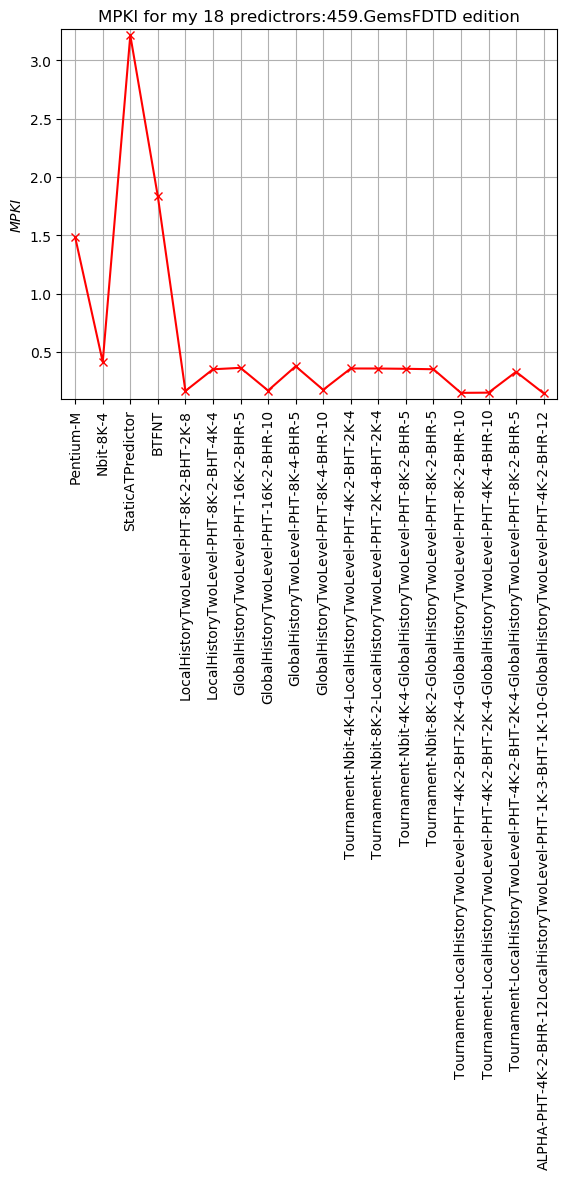
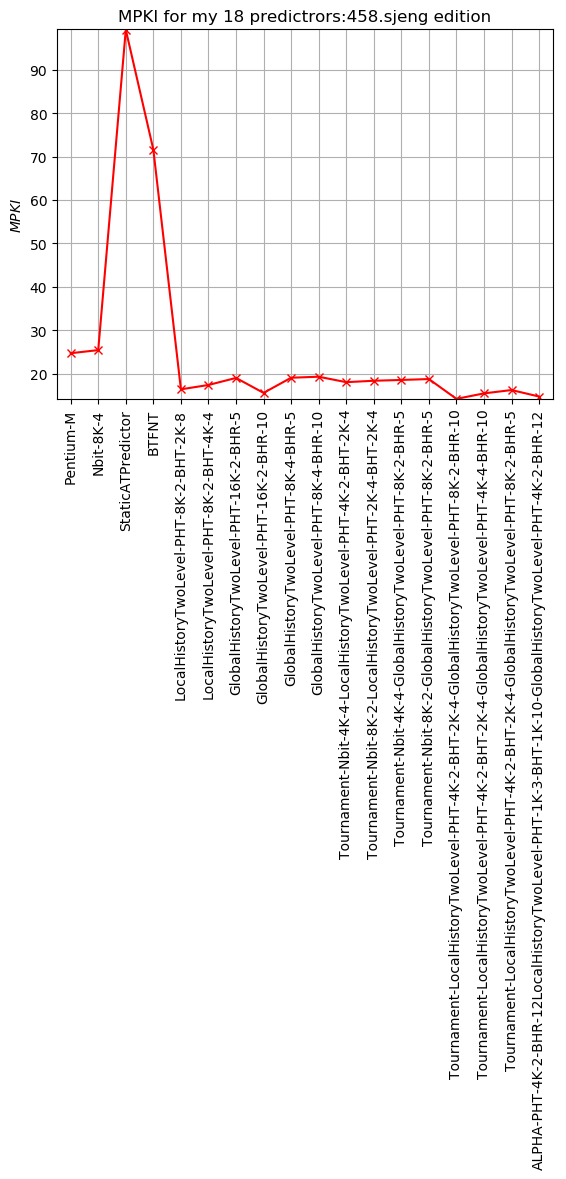
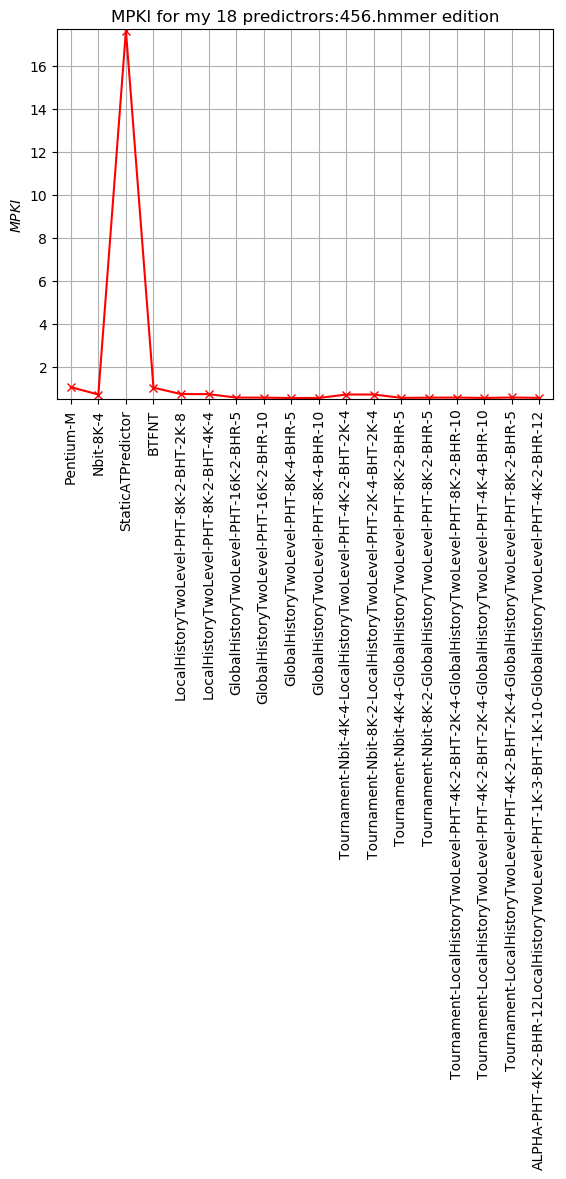
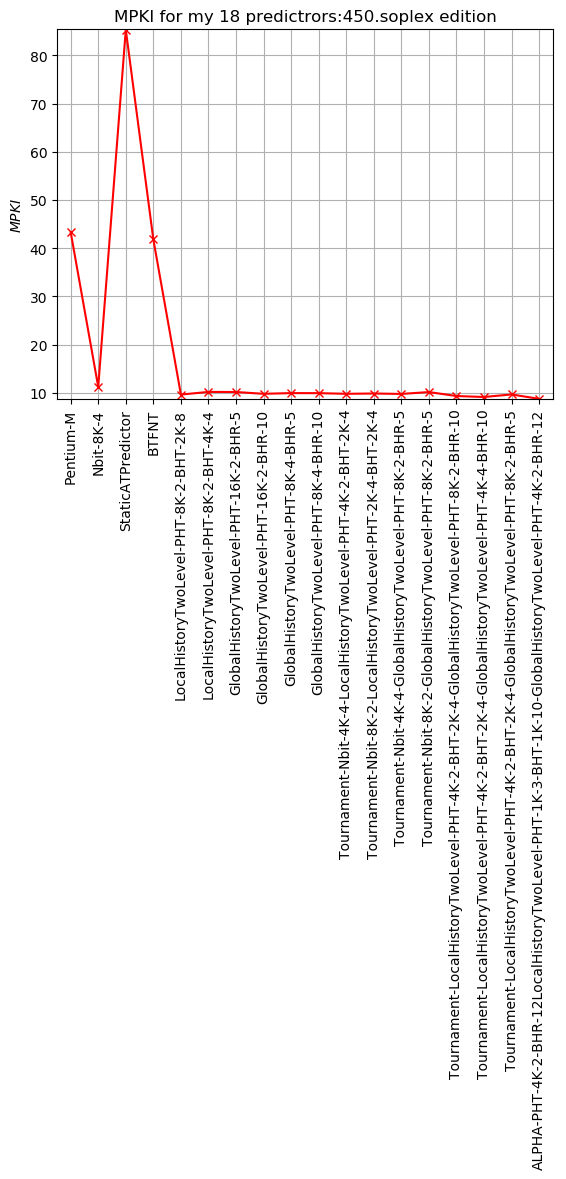
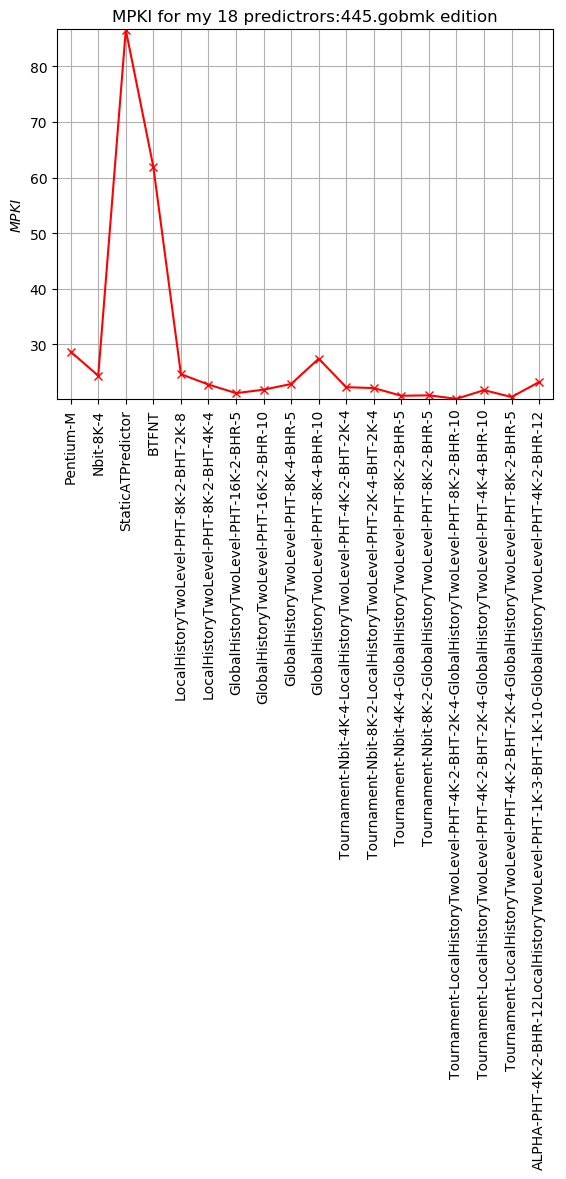
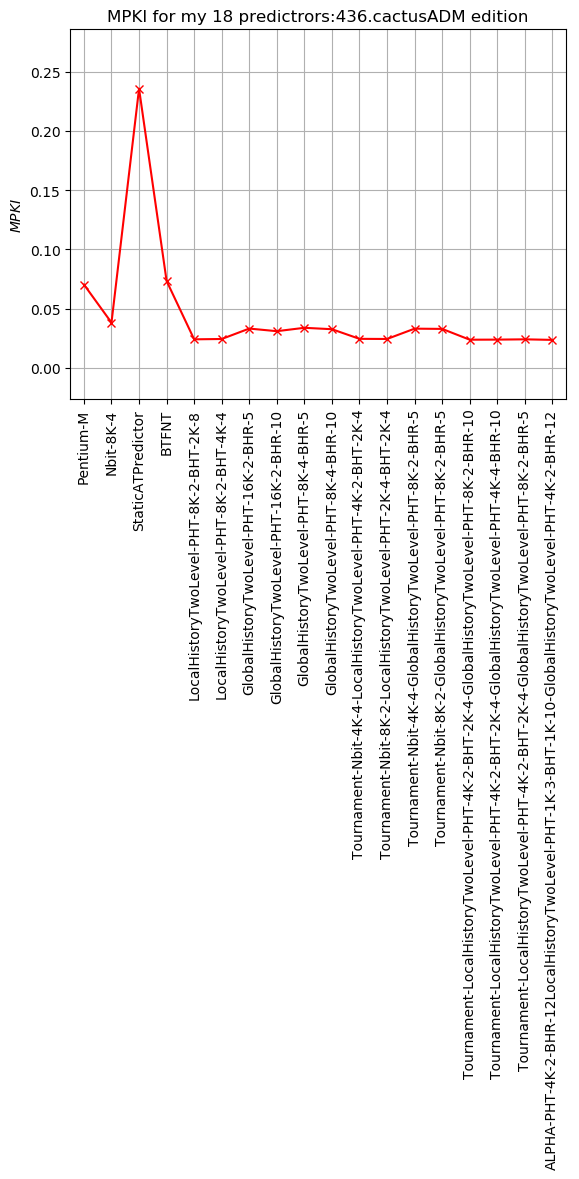
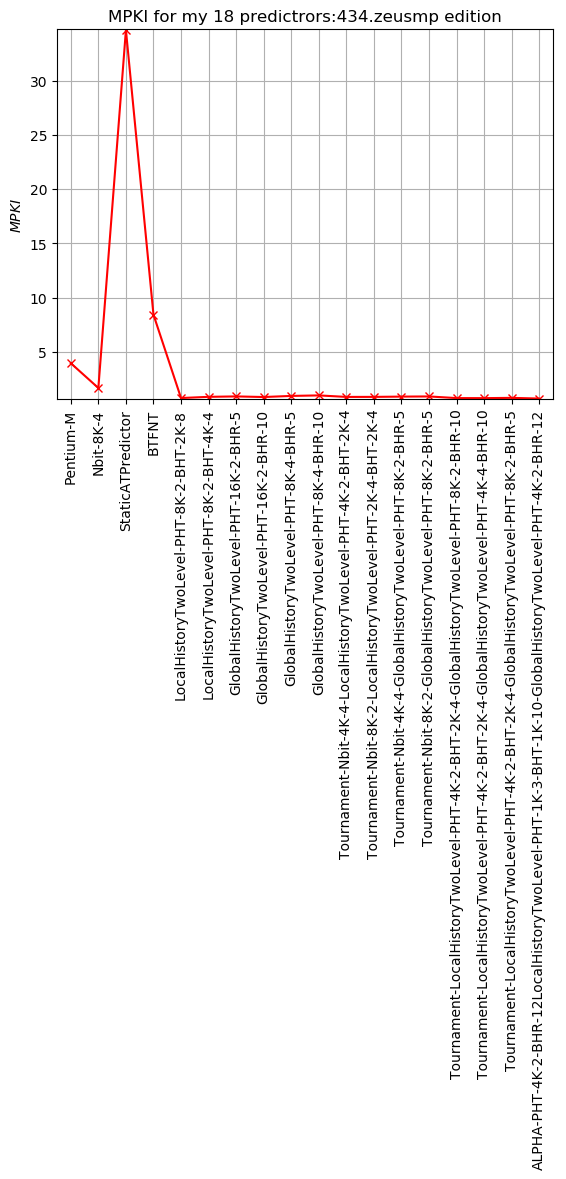
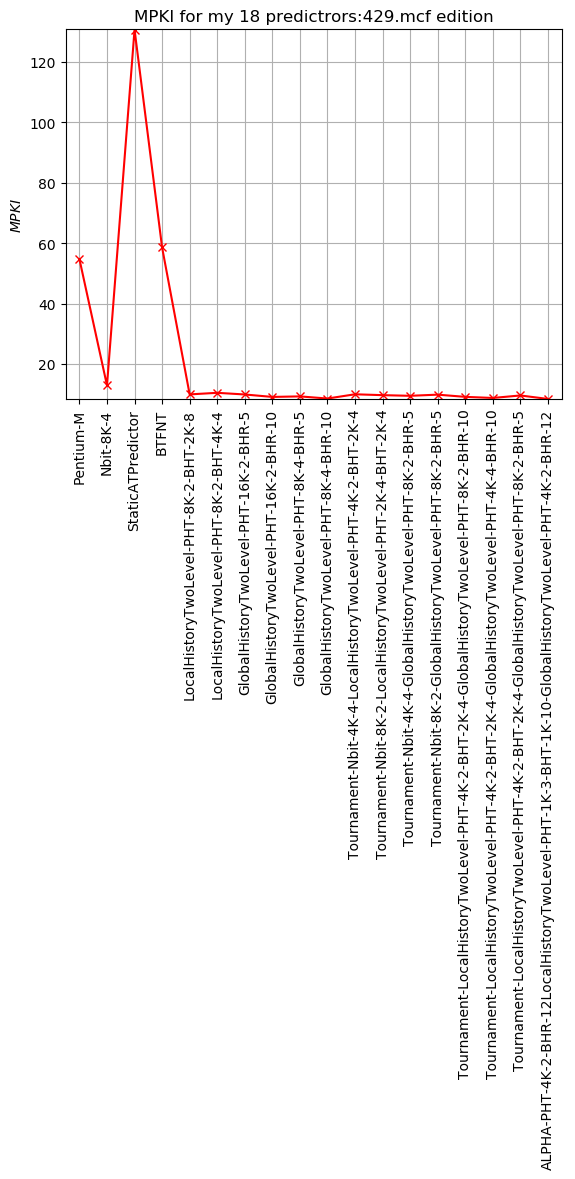
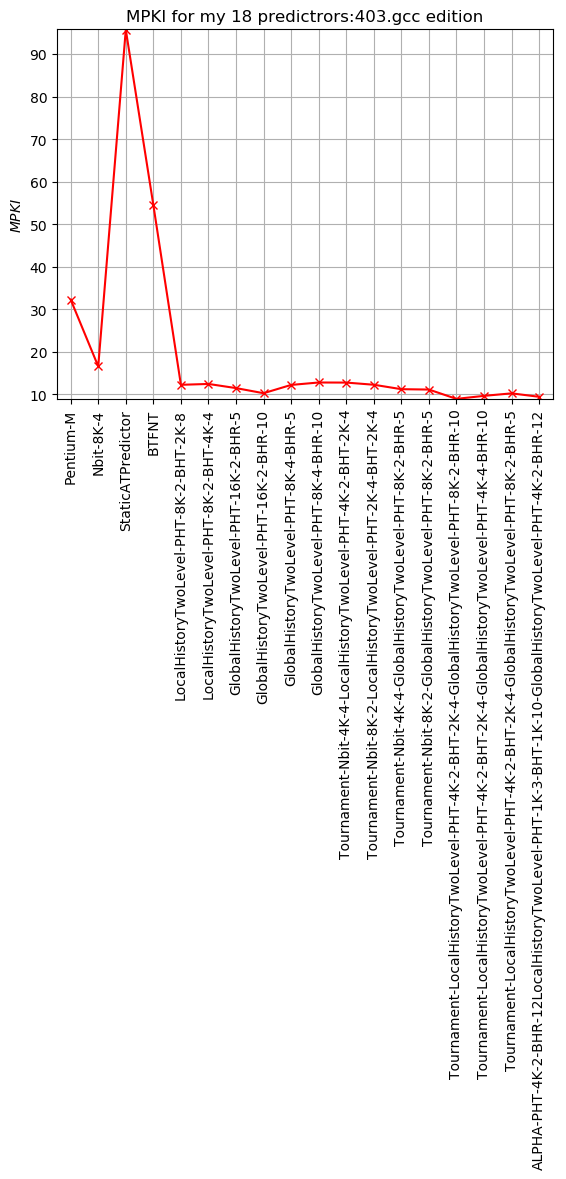
IN PARTICULAR THE MOST EFFICIENT OF ALL WAS ALPHA21264 PREDICTOR

FROM THE OTHER TOURNAMENT PREDICTORS THE GREATEST EFFICIENT WAS OBSERVED IN PREDISTORS THAT CONTAINED AT THE SAME TIME BOTH LOCAL AND GLOBAL INFORMATION, IN OUR CASE THE:”TOURNAMENT”.

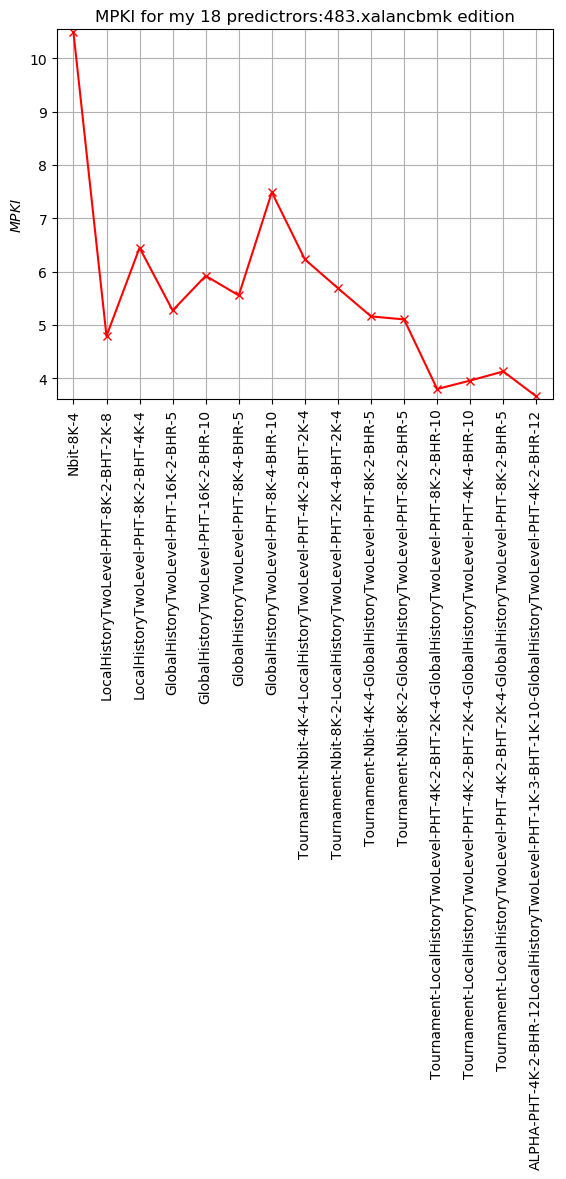
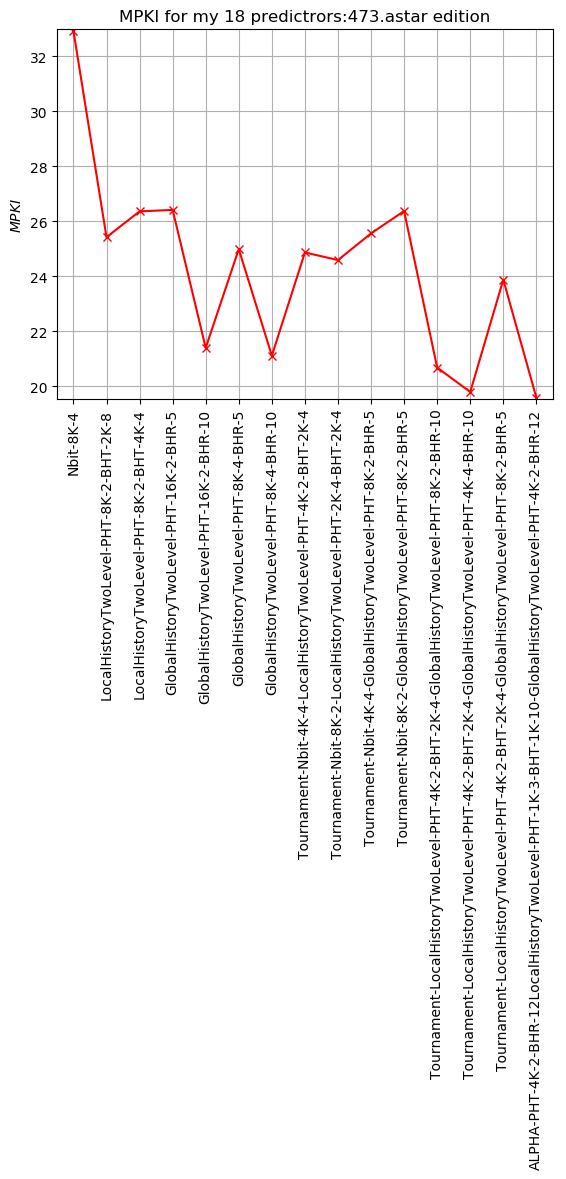
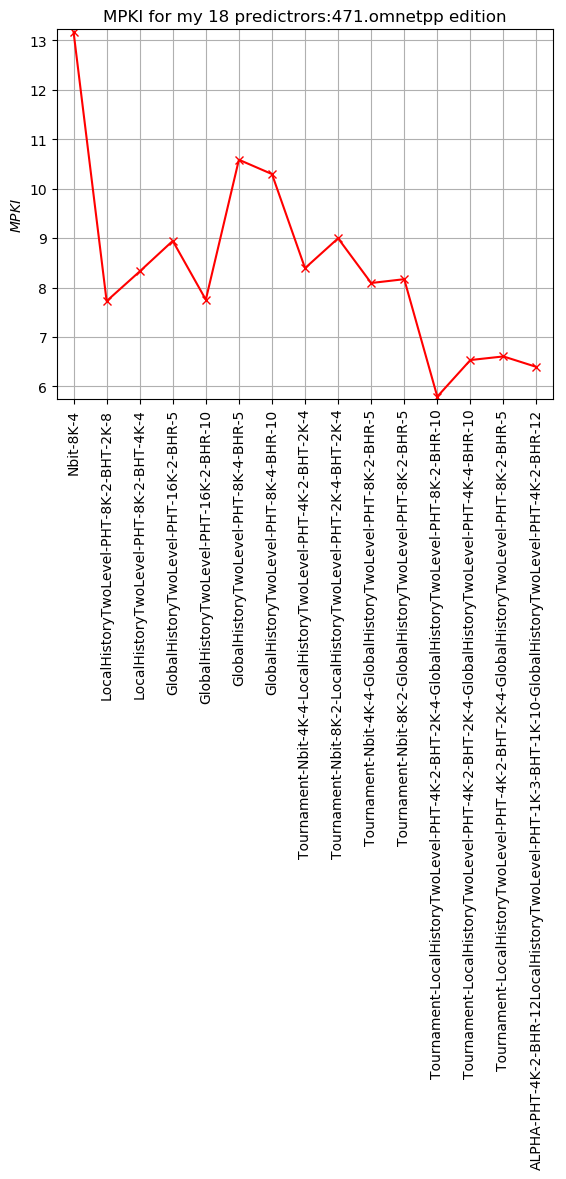
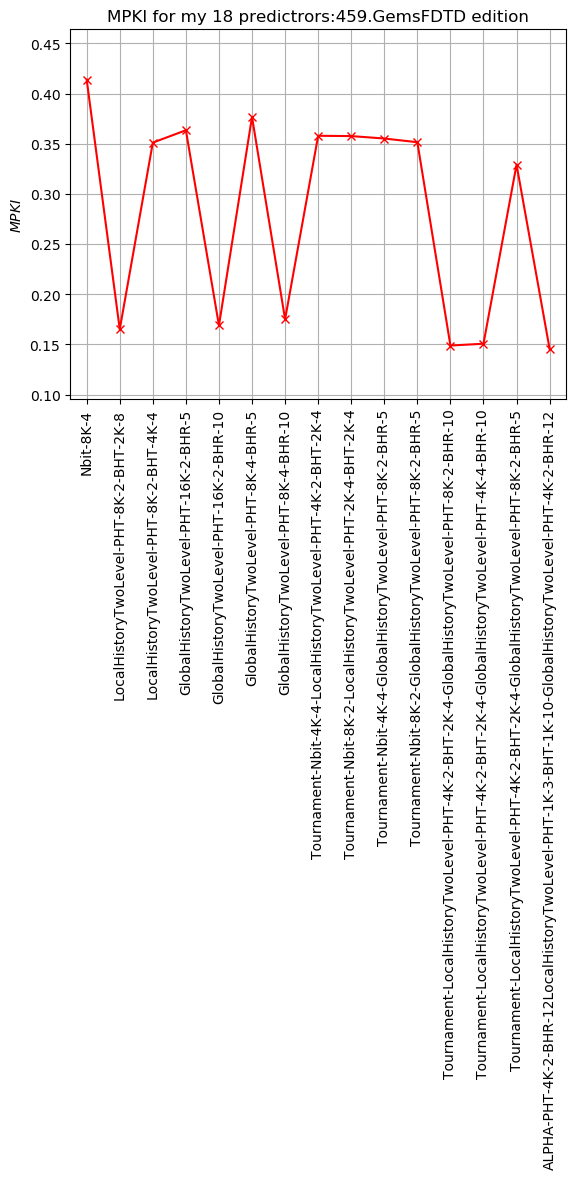
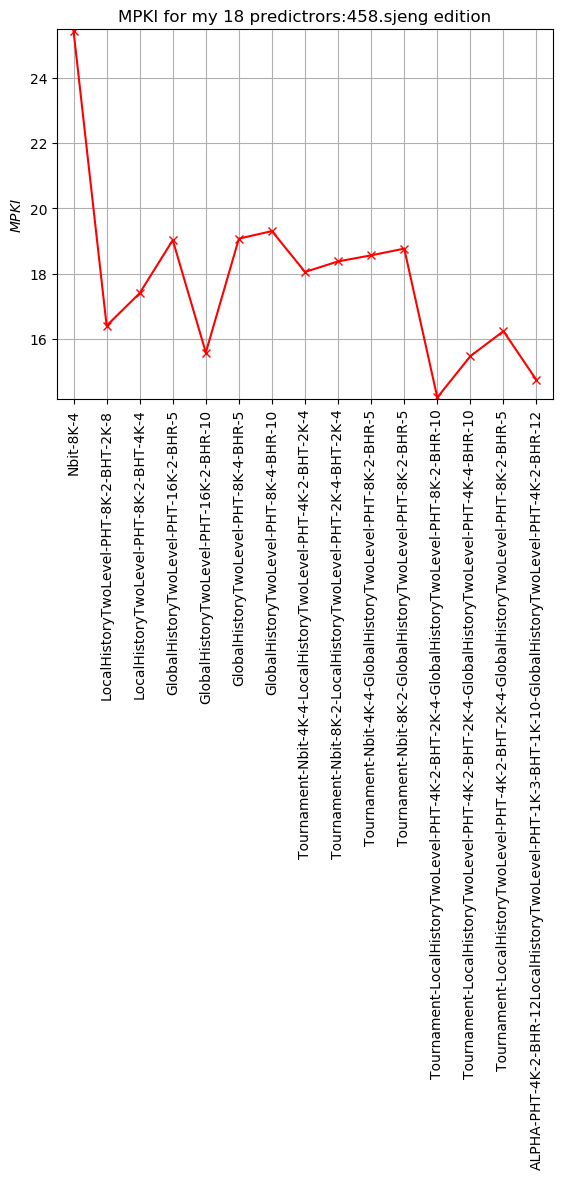
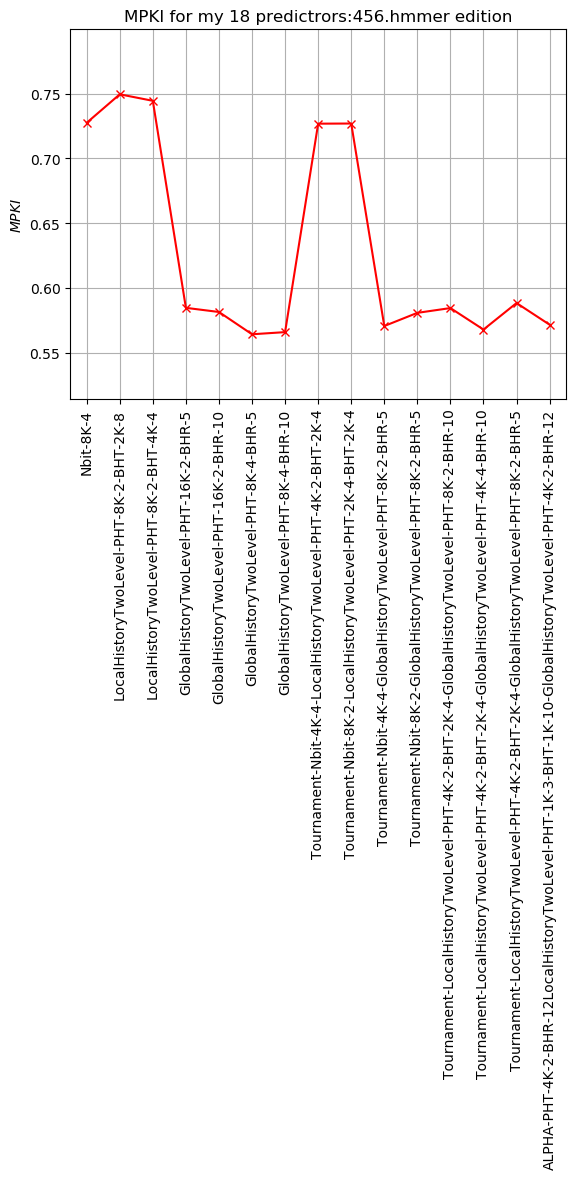
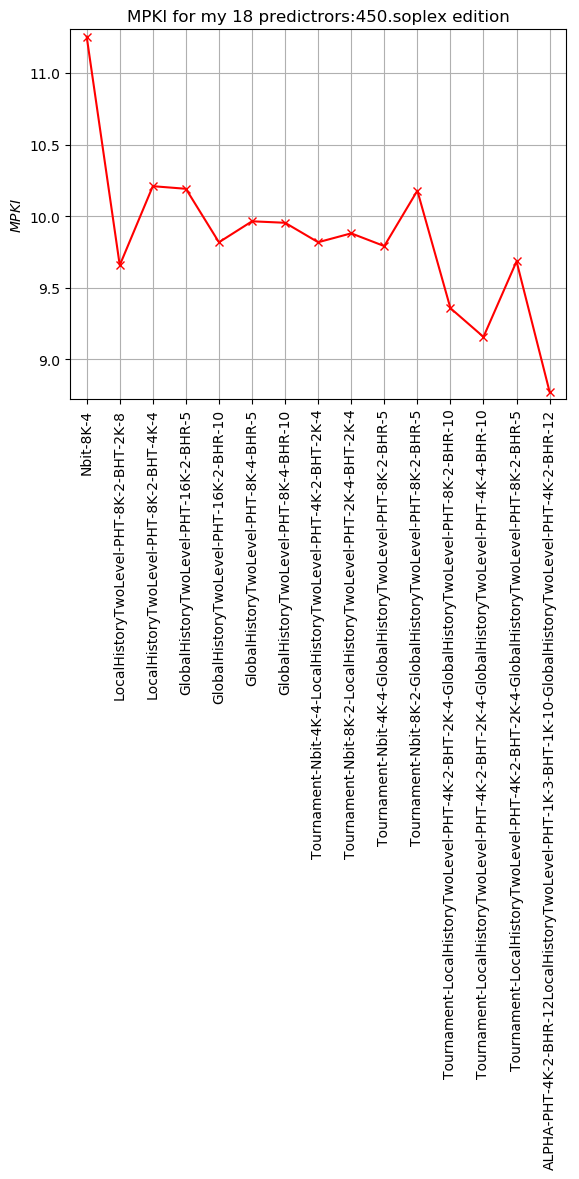
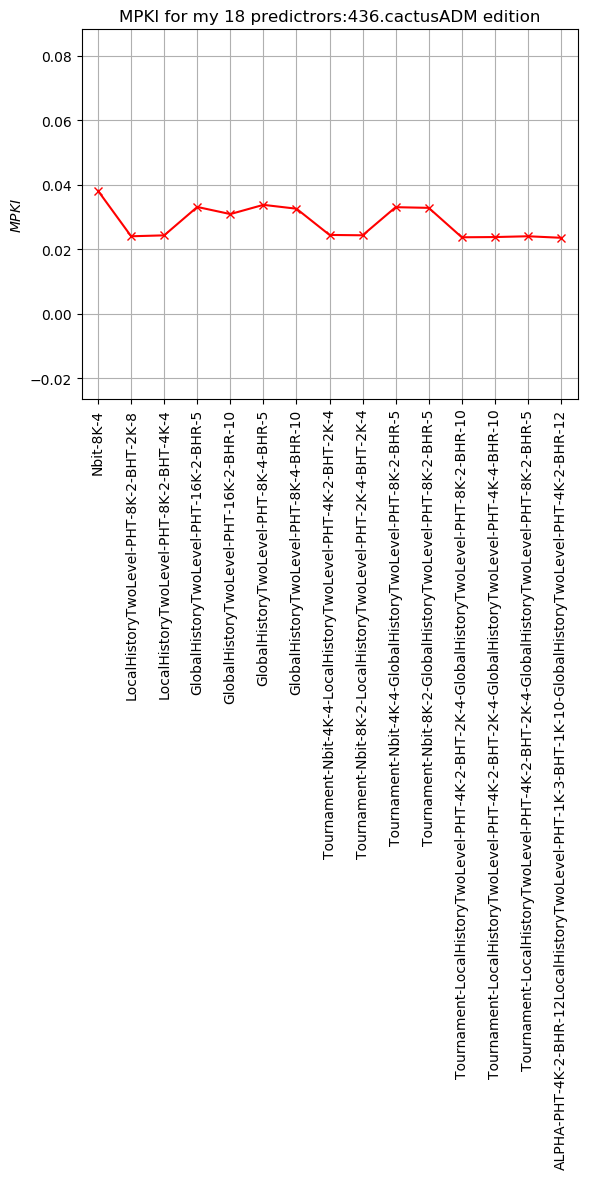
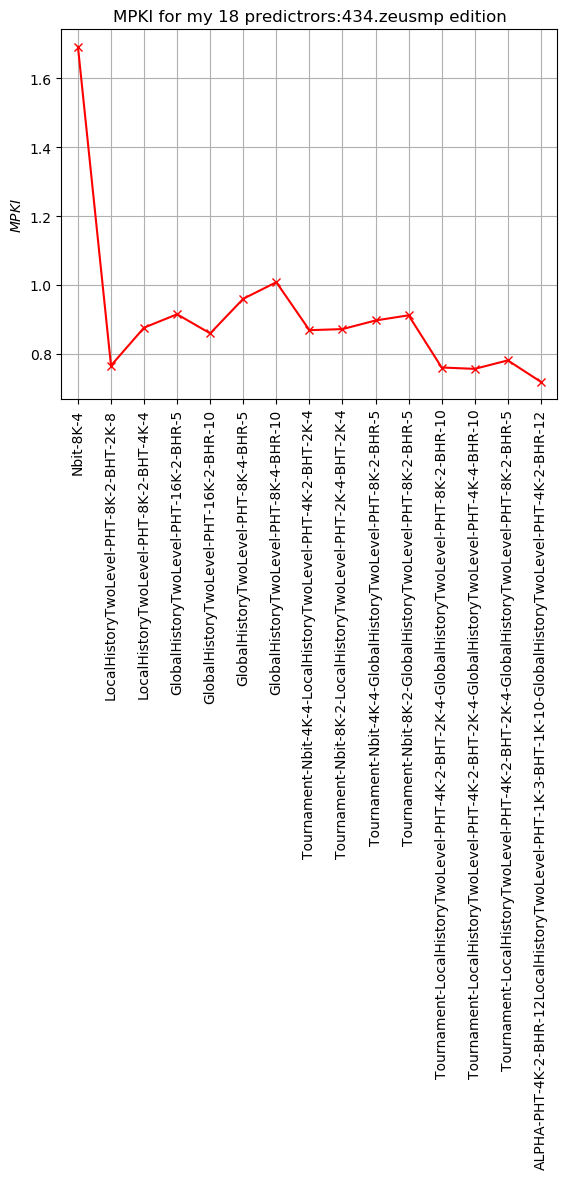
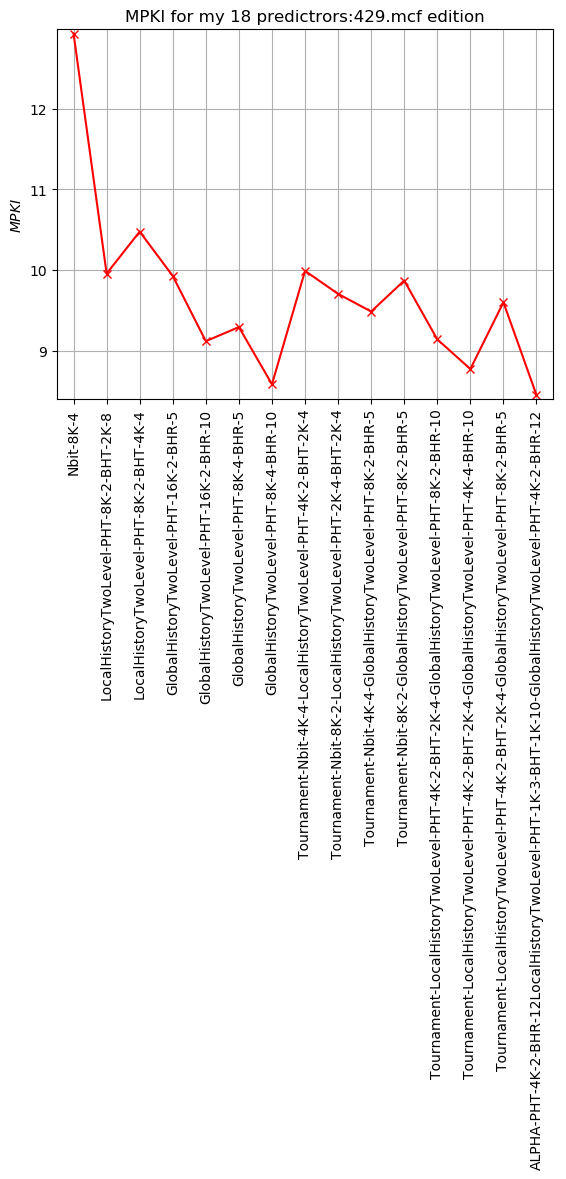
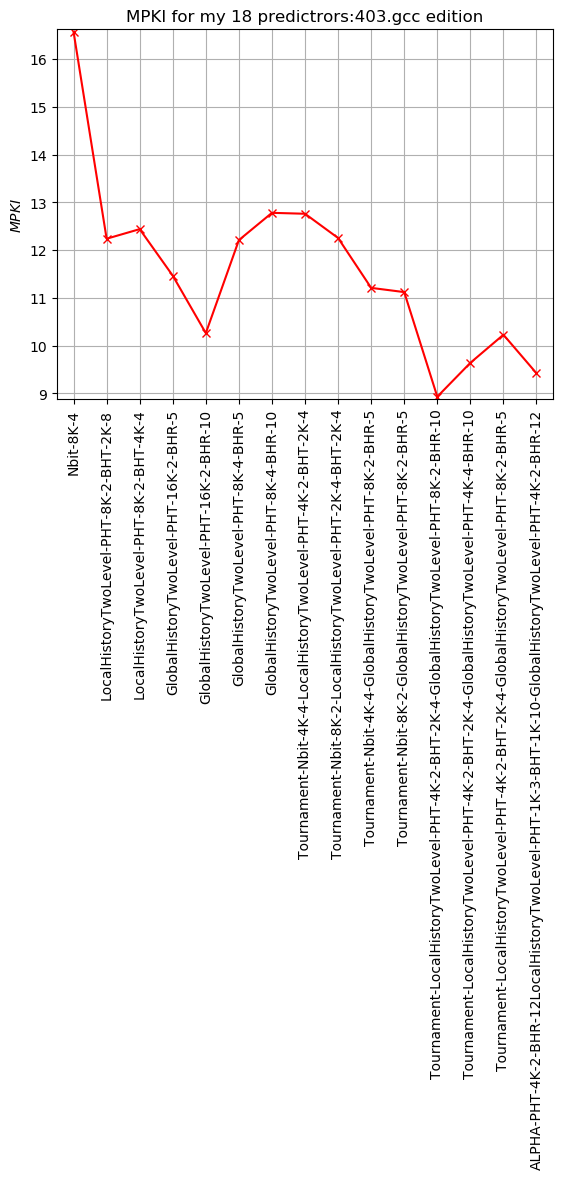
IF NOT THE BEST ALPHA WAS AT LEAST THE SECOND BEST IN EVERY BENCHMARK

ALPHA PREDICTOR WOULD BE MY CHOICE FOR FUTURE IMPLEMENTATIONS

ALL 18 PREDICTORS:



EXCLUDING THE UNEFFICIENT ONES:



APPENDIX A

CODE\_FOR\_PLOTS

Btb

#!/usr/bin/env python

import sys

import matplotlib

matplotlib.use('Agg')

import matplotlib.pyplot as plt

import numpy as np

## For nbit predictors

"""

fp1 = open("/home/thanasis-linux/advcomparch-2019-20-ex2-helpcode/ex2/4.5/names.txt")

lines = fp1.readline()

lines.split

print(lines)

names = []

for line in lines:

names.append(line)

print(len(line))

"""

outs = {

"429.mcf.cslab\_branch\_btb.out",

"483.xalancbmk.cslab\_branch\_btb.out",

"403.gcc.cslab\_branch\_btb.out",

"436.cactusADM.cslab\_branch\_btb.out",

"434.zeusmp.cslab\_branch\_btb.out",

"456.hmmer.cslab\_branch\_btb.out",

"458.sjeng.cslab\_branch\_btb.out",

"445.gobmk.cslab\_branch\_btb.out",

"459.GemsFDTD.cslab\_branch\_btb.out",

"450.soplex.cslab\_branch\_btb.out",

"473.astar.cslab\_branch\_btb.out",

"471.omnetpp.cslab\_branch\_btb.out",

}

i=0

for out in outs: #line:

#print("line: " + line)

#tokens = line.split()

#print ("tokens: ")

#print(tokens)

#lines=tokens[0].split('\n')[0]

#print("lines: " + lines)

predictors\_to\_plot = [ " BTB", ]

x\_Axis = []

mpki\_Axis = []

target\_Axis = []

fp = open("/home/thanasis-linux/advcomparch-2019-20-ex2-helpcode/ex2/4.3/" + out)

linek = fp.readline()

print("outerlinek :" + linek)

while linek:

#print("linek :" + linek)

tokens = linek.split()

if linek.startswith("Total Instructions:"):

total\_ins = long(tokens[2])

else:

for pred\_prefix in predictors\_to\_plot:

if linek.startswith(pred\_prefix):

predictor\_string = tokens[0].split(':')[0]

correct\_predictions = long(tokens[1])

incorrect\_predictions = long(tokens[2])

target\_predictions = long(tokens[3])

x\_Axis.append(predictor\_string)#+tokens[1]+tokens[2])

target\_Axis.append(target\_predictions / (total\_ins / 1000.0))

mpki\_Axis.append(incorrect\_predictions / (total\_ins / 1000.0))

linek = fp.readline()

fig, ax1 = plt.subplots()

ax1.grid(True)

xAx = np.arange(len(x\_Axis))

ax1.xaxis.set\_ticks(np.arange(0, len(x\_Axis), 1))

ax1.set\_xticklabels(x\_Axis, rotation=90)

ax1.set\_xlim(-0.5, len(x\_Axis) - 0.5)

ax1.set\_ylim(min(mpki\_Axis) - 0.05, max(mpki\_Axis) + 0.05)

ax1.set\_ylabel("$MPKI$")

line1 = ax1.plot(mpki\_Axis, label="mpki", color="red",marker='x')

ax2 = ax1.twinx()

ax2.xaxis.set\_ticks(np.arange(0, len(x\_Axis), 1))

ax2.set\_xticklabels(x\_Axis, rotation=90)

ax2.set\_xlim(-0.5, len(x\_Axis) - 0.5)

ax2.set\_ylim(min(target\_Axis) - 0.05 \* min(target\_Axis), max(target\_Axis) + 0.05 \* max(target\_Axis))

ax2.set\_ylabel("$TargetCorrect$")

line2 = ax2.plot(target\_Axis, label="TargetCorrect", color="b",marker='o')

tokens0 = out.split()

lines0=tokens0[0].split('.cslab\_branch\_btb.out')[0]

plt.title("MPKI-TARGET\_CORRECT-BTB: " +lines0 + " edition")

#raw\_input("Please provide a filename for the produced file (e.g. output.png): ")

tokens1 = out.split()

lines2=tokens1[0].split('.out')[0]

plt.savefig( "/home/thanasis-linux/advcomparch-2019-20-ex2-helpcode/plots/4.3/" +lines2 + ".png" , bbox\_inches="tight")

#\_zoomed

Stats

#!/usr/bin/env python

import sys

import matplotlib

matplotlib.use('Agg')

import matplotlib.pyplot as plt

import numpy as np

##problems with running all outs together never find time to understand and ##fix it

outs = {

# "429.mcf.cslab\_branch\_stats.out",

# "483.xalancbmk.cslab\_branch\_stats.out",

# "403.gcc.cslab\_branch\_stats.out",

# "436.cactusADM.cslab\_branch\_stats.out",

# "434.zeusmp.cslab\_branch\_stats.out",

"456.hmmer.cslab\_branch\_stats.out",

# "458.sjeng.cslab\_branch\_stats.out",

# "445.gobmk.cslab\_branch\_stats.out",

# "459.GemsFDTD.cslab\_branch\_stats.out",

# "450.soplex.cslab\_branch\_stats.out",

# "473.astar.cslab\_branch\_stats.out",

# "471.omnetpp.cslab\_branch\_stats.out",

}

rects1=0

rects2=0

rects3=0

rects4=0

rects5=0

rects6=0

i=0

for out in outs:

i+=1

print(rects1)

x\_Axis = []

mpki\_Axis = []

rects1=[]

fp = open("/home/thanasis-linux/advcomparch-2019-20-ex2-helpcode/ex2/4.1/" + out)

linek = fp.readline()

print("outerlinek :" + linek)

while linek:

tokens=linek.split()

if linek.startswith(" Total-Branches:"):

total\_bra = float(tokens[1])

elif linek.startswith(" Conditional-Taken-Branches:"):

total\_taken = float(tokens[1])

elif linek.startswith(" Conditional-NotTaken-Branches:"):

total\_notTaken = float(tokens[1])

elif linek.startswith(" Unconditional-Branches:"):

total\_unco = float(tokens[1])

elif linek.startswith(" Calls:"):

total\_calls = float(tokens[1])

elif linek.startswith(" Returns:"):

total\_returns = float(tokens[1])

linek = fp.readline()

x1=[1]#["Total-Branches"]

y1=[100]

x2=[2]#["Conditional-Taken-Branches"]

y2=[100\*total\_taken/total\_bra]

x3=[3]#["Conditional-NotTaken-Branches"]

y3=[100\*total\_notTaken/total\_bra]

x4=[4]

y4=[100\*total\_unco/total\_bra]

x5=[5]

y5=[100\*total\_calls/total\_bra]

x6=[6]

y6=[100\*total\_returns/total\_bra]

rects1=plt.bar(x1,y1,label='Total-Branches=%d' % total\_bra, color='b')

rects2=plt.bar(x2,y2,label='Taken-Branches',color='m')

rects3=plt.bar(x3,y3,label='NotTaken-Branches', color='r')

rects4=plt.bar(x4,y4,label='Unconditional-Branches', color='c')

rects5=plt.bar(x5,y5,label='Calls',color='y')

rects6=plt.bar(x6,y6,label='Returns', color='k')

plt.xlabel('x')

plt.ylabel('Percentage (%)')

tokens0 = out.split()

lines0=tokens0[0].split('.cslab\_branch\_stats.out')[0]

def per\_100\_(rects):

for rect in rects:

height = rect.get\_height()

plt.text(rect.get\_x() + rect.get\_width()/2., 1.05\*height, "%0.2f %%" % height, ha='center', va='bottom')

plt.title("Branches\_Statistics:" + lines0 + " edition")

plt.legend()

per\_100\_(rects1)

per\_100\_(rects2)

per\_100\_(rects3)

per\_100\_(rects4)

per\_100\_(rects5)

per\_100\_(rects6)

tokens1 = out.split()

lines2=tokens1[0].split('.out')[0]

plt.savefig( "/home/thanasis-linux/advcomparch-2019-20-ex2-helpcode/plots/4.1/" + lines2 + ".png" , bbox\_inches="tight")

#\_zoomed

print(rects1)

"""

rects1=plt.bar(x1,y1,label='Total-Branches=%d' % total\_bra, color='b')

rects2=plt.bar(x2,y2,label='Taken-Branches',color='m')

rects3=plt.bar(x3,y3,label='NotTaken-Branches', color='r')

rects4=plt.bar(x4,y4,label='Unconditional-Branches', color='c')

rects5=plt.bar(x5,y5,label='Calls',color='y')

rects6=plt.bar(x6,y6,label='Returns', color='k')

plt.xlabel('x')

plt.ylabel('Percentage (%)')

tokens1 = out.split()

lines2=tokens1[0].split('.out')[0]

plt.savefig( lines2 + ".png" , bbox\_inches="tight")

#\_zoomed

"""

Nbit

#!/usr/bin/env python

import sys

import matplotlib

matplotlib.use('Agg')

import matplotlib.pyplot as plt

import numpy as np

## For nbit predictors

"""

fp1 = open("/home/thanasis-linux/advcomparch-2019-20-ex2-helpcode/ex2/4.5/names.txt")

lines = fp1.readline()

lines.split

print(lines)

names = []

for line in lines:

names.append(line)

print(len(line))

"""

outs = {

"429.mcf.cslab\_branch\_predictors.out",

"483.xalancbmk.cslab\_branch\_predictors.out",

"403.gcc.cslab\_branch\_predictors.out",

"436.cactusADM.cslab\_branch\_predictors.out",

"434.zeusmp.cslab\_branch\_predictors.out",

"456.hmmer.cslab\_branch\_predictors.out",

"458.sjeng.cslab\_branch\_predictors.out",

"445.gobmk.cslab\_branch\_predictors.out",

"459.GemsFDTD.cslab\_branch\_predictors.out",

"450.soplex.cslab\_branch\_predictors.out",

"473.astar.cslab\_branch\_predictors.out",

"471.omnetpp.cslab\_branch\_predictors.out",

}

i=0

for out in outs:

predictors\_to\_plot = [ " Nbit", ]

x\_Axis = []

mpki\_Axis = []

fp = open("/home/thanasis-linux/advcomparch-2019-20-ex2-helpcode/ex2/4.2.1/" + out)

linek = fp.readline()

print("outerlinek :" + linek)

while linek:

#print("linek :" + linek)

tokens = linek.split()

if linek.startswith("Total Instructions:"):

total\_ins = long(tokens[2])

else:

for pred\_prefix in predictors\_to\_plot:

if linek.startswith(pred\_prefix):

predictor\_string = tokens[0].split(':')[0]

correct\_predictions = long(tokens[1])

incorrect\_predictions = long(tokens[2])

x\_Axis.append(predictor\_string)

mpki\_Axis.append(incorrect\_predictions / (total\_ins / 1000.0))

linek = fp.readline()

fig, ax1 = plt.subplots()

ax1.grid(True)

xAx = np.arange(len(x\_Axis))

ax1.xaxis.set\_ticks(np.arange(0, len(x\_Axis), 1))

ax1.set\_xticklabels(x\_Axis, rotation=90)

ax1.set\_xlim(-0.5, len(x\_Axis) - 0.5)

ax1.set\_ylim(min(mpki\_Axis) - 0.05, max(mpki\_Axis) + 0.05)

ax1.set\_ylabel("$MPKI$")

line1 = ax1.plot(mpki\_Axis, label="mpki", color="red",marker='x')

tokens0 = out.split()

lines0=tokens0[0].split('.cslab\_branch\_predictors.out')[0]

plt.title("MPKI Nbit:" + lines0 + " edition")

tokens1 = out.split()

lines2=tokens1[0].split('.out')[0]

plt.savefig( "/home/thanasis-linux/advcomparch-2019-20-ex2-helpcode/plots/4.2.1/" + lines2 + ".png" , bbox\_inches="tight")

#\_zoomed

All predictors 4.5

#!/usr/bin/env python

import sys

import matplotlib

matplotlib.use('Agg')

import matplotlib.pyplot as plt

import numpy as np

## For nbit predictors

"""

fp1 = open("/home/thanasis-linux/advcomparch-2019-20-ex2-helpcode/ex2/4.5/names.txt")

lines = fp1.readline()

lines.split

print(lines)

names = []

for line in lines:

names.append(line)

print(len(line))

"""

outs = {

"429.mcf.cslab\_branch\_predictormania.out",

"483.xalancbmk.cslab\_branch\_predictormania.out",

"403.gcc.cslab\_branch\_predictormania.out",

"436.cactusADM.cslab\_branch\_predictormania.out",

"434.zeusmp.cslab\_branch\_predictormania.out",

"456.hmmer.cslab\_branch\_predictormania.out",

"458.sjeng.cslab\_branch\_predictormania.out",

"445.gobmk.cslab\_branch\_predictormania.out",

"459.GemsFDTD.cslab\_branch\_predictormania.out",

"450.soplex.cslab\_branch\_predictormania.out",

"473.astar.cslab\_branch\_predictormania.out",

"471.omnetpp.cslab\_branch\_predictormania.out",

}

i=0

for out in outs: #line:

#print("line: " + line)

#tokens = line.split()

#print ("tokens: ")

#print(tokens)

#lines=tokens[0].split('\n')[0]

#print("lines: " + lines)

predictors\_to\_plot = [ " Pentium-M",

" Nbit-8K-4" ,

" StaticATPredictor" ,

" BTFNT",

" LocalHistoryTwoLevel-PHT-8K-2-BHT-2K-8" ,

" LocalHistoryTwoLevel-PHT-8K-2-BHT-4K-4",

" GlobalHistoryTwoLevel-PHT-16K-2-BHR-5",

" GlobalHistoryTwoLevel-PHT-16K-2-BHR-10",

" GlobalHistoryTwoLevel-PHT-8K-4-BHR-5",

" GlobalHistoryTwoLevel-PHT-8K-4-BHR-10",

" Tournament-Nbit-4K-4-LocalHistoryTwoLevel-PHT-4K-2-BHT-2K-4",

" Tournament-Nbit-8K-2-LocalHistoryTwoLevel-PHT-2K-4-BHT-2K-4",

" Tournament-Nbit-4K-4-GlobalHistoryTwoLevel-PHT-8K-2-BHR-5",

" Tournament-Nbit-8K-2-GlobalHistoryTwoLevel-PHT-8K-2-BHR-5",

" Tournament-LocalHistoryTwoLevel-PHT-4K-2-BHT-2K-4-GlobalHistoryTwoLevel-PHT-8K-2-BHR-10",

" Tournament-LocalHistoryTwoLevel-PHT-4K-2-BHT-2K-4-GlobalHistoryTwoLevel-PHT-4K-4-BHR-10",

" Tournament-LocalHistoryTwoLevel-PHT-4K-2-BHT-2K-4-GlobalHistoryTwoLevel-PHT-8K-2-BHR-5",

" ALPHA-PHT-4K-2-BHR-12LocalHistoryTwoLevel-PHT-1K-3-BHT-1K-10-GlobalHistoryTwoLevel-PHT-4K-2-BHR-12",]

x\_Axis = []

mpki\_Axis = []

fp = open("/home/thanasis-linux/advcomparch-2019-20-ex2-helpcode/ex2/4.5/" + out)

linek = fp.readline()

print("outerlinek :" + linek)

while linek:

#print("linek :" + linek)

tokens = linek.split()

if linek.startswith("Total Instructions:"):

total\_ins = long(tokens[2])

else:

for pred\_prefix in predictors\_to\_plot:

if linek.startswith(pred\_prefix):

predictor\_string = tokens[0].split(':')[0]

correct\_predictions = long(tokens[1])

incorrect\_predictions = long(tokens[2])

x\_Axis.append(predictor\_string)#+tokens[1]+tokens[2])

#if (incorrect\_predictions==0):

#incorrect\_predictions=1

mpki\_Axis.append(incorrect\_predictions / (total\_ins / 1000.0))

linek = fp.readline()

i+=i

x= [i]

fig, ax1 = plt.subplots()

ax1.grid(True)

xAx = np.arange(len(x\_Axis))

ax1.xaxis.set\_ticks(np.arange(0, len(x\_Axis), 1))

ax1.set\_xticklabels(x\_Axis, rotation=90)

ax1.set\_xlim(-0.5, len(x\_Axis) - 0.5)

ax1.set\_ylim(min(mpki\_Axis) - 0.05, max(mpki\_Axis) + 0.05)

ax1.set\_ylabel("$MPKI$")

line1 = ax1.plot(mpki\_Axis, label="mpki", color="red",marker='x')

#ax1.legend(bars, labels, bbox\_to\_anchor=(1.04, 0.5), loc="center left")

tokens0 = out.split()

lines0=tokens0[0].split('.cslab\_branch\_predictormania.out')[0]

plt.title("MPKI for my 18 predictrors:" + lines0 + " edition")

#raw\_input("Please provide a filename for the produced file (e.g. output.png): ")

tokens1 = out.split()

lines2=tokens1[0].split('.out')[0]

plt.savefig( "/home/thanasis-linux/advcomparch-2019-20-ex2-helpcode/plots/4.5/" + lines2 + ".png" , bbox\_inches="tight")

#\_zoomed#\_effecient\_ones

Ras

#!/usr/bin/env python

import sys

import matplotlib

matplotlib.use('Agg')

import matplotlib.pyplot as plt

import numpy as np

## For nbit predictors

outs = {

"429.mcf.cslab\_branch\_ras.out",

"483.xalancbmk.cslab\_branch\_ras.out",

"403.gcc.cslab\_branch\_ras.out",

"436.cactusADM.cslab\_branch\_ras.out",

"434.zeusmp.cslab\_branch\_ras.out",

"456.hmmer.cslab\_branch\_ras.out",

"458.sjeng.cslab\_branch\_ras.out",

"445.gobmk.cslab\_branch\_ras.out",

"459.GemsFDTD.cslab\_branch\_ras.out",

"450.soplex.cslab\_branch\_ras.out",

"473.astar.cslab\_branch\_ras.out",

"471.omnetpp.cslab\_branch\_ras.out",

}

for out in outs: #line:

#print("line: " + line)

#tokens = line.split()

#print ("tokens: ")

#print(tokens)

#lines=tokens[0].split('\n')[0]

#print("lines: " + lines)

predictors\_to\_plot = [ "RAS (" ]

x\_Axis = []

mpki\_Axis = []

mpki\_Axis1 = []

fp = open("/home/thanasis-linux/advcomparch-2019-20-ex2-helpcode/ex2/4.4/" + out)

linek = fp.readline()

print("outerlinek :" + linek)

while linek:

print("linek :" + linek)

tokens = linek.split()

if linek.startswith("Total Instructions:"):

total\_ins = long(tokens[2])

else:

for pred\_prefix in predictors\_to\_plot:

if linek.startswith(pred\_prefix):

predictor\_string = tokens[0].split(':')[0]

correct\_predictions = long(tokens[3])

incorrect\_predictions = long(tokens[4])

x\_Axis.append(predictor\_string+tokens[1]+tokens[2])

#if (incorrect\_predictions==0):

#incorrect\_predictions=1

mpki\_Axis.append(incorrect\_predictions / (total\_ins / 1000000000.0))

linek = fp.readline()

fig, ax1 = plt.subplots()

ax1.grid(True)

xAx = np.arange(len(x\_Axis))

ax1.xaxis.set\_ticks(np.arange(0, len(x\_Axis), 1))

ax1.set\_xticklabels(x\_Axis, rotation=45)

ax1.set\_xlim(-0.5, len(x\_Axis) - 0.5)

ax1.set\_ylim(min(mpki\_Axis) - 0.05, max(mpki\_Axis) + 0.05)

ax1.set\_ylabel("$MPKI\*10^6$")

line1 = ax1.plot(mpki\_Axis, label="mpki", color="red",marker='x')

#ax1.legend(bars, labels, bbox\_to\_anchor=(1.04, 0.5), loc="center left")

tokens0 = out.split()

lines0=tokens0[0].split('.cslab\_branch\_ras.out')[0]

plt.title("MPKI-ras: " +lines0 + " edition")

#raw\_input("Please provide a filename for the produced file (e.g. output.png): ")

tokens1 = out.split()

lines2=tokens1[0].split('.out')[0]

#lines=tokens[0].split('\n')[0]

#print("lines2: " + lines2)

plt.savefig( "/home/thanasis-linux/advcomparch-2019-20-ex2-helpcode/plots/4.4/" + lines2 + ".png" , bbox\_inches="tight")

APPENDIX B

CODE\_FOR\_BTB

CODE\_FOR\_

Everything except plots to appendix c due to time

APPENDIX C

CODE\_FOR\_COSTUM\_MADE\_PREDICTORS

Run\_brach\_predictors

#!/bin/bash

## Execute this script in the helpcode directory.

## Example of usage: ./run\_branch\_predictors.sh 403.gcc

## Modify the following paths appropriately

## CAUTION: use only absolute paths below!!!

PIN\_EXE=/home/thanasis-linux/pin-3.13-98189-g60a6ef199-gcc-linux/pin

PIN\_TOOL=/home/thanasis-linux/advcomparch-2019-20-ex2-helpcode/pintool/obj-intel64/cslab\_branch.so

outDir="/home/thanasis-linux/Desktop"

for BENCH in $@; do

cd spec\_execs\_train\_inputs/$BENCH

line=$(cat speccmds.cmd)

stdout\_file=$(echo $line | cut -d' ' -f2)

stderr\_file=$(echo $line | cut -d' ' -f4)

cmd=$(echo $line | cut -d' ' -f5-)

pinOutFile="$outDir/${BENCH}.cslab\_branch\_predictormania.out"

pin\_cmd="$PIN\_EXE -t $PIN\_TOOL -o $pinOutFile -- $cmd 1> $stdout\_file 2> $stderr\_file"

echo "PIN\_CMD: $pin\_cmd"

/bin/bash -c "time $pin\_cmd"

cd ../../

done

predictor.h

#ifndef BRANCH\_PREDICTOR\_H

#define BRANCH\_PREDICTOR\_H

#include <sstream> // std::ostringstream

#include <cmath> // pow()

#include <cstring> // memset()

/\*\*

\* A generic BranchPredictor base class.

\* All predictors can be subclasses with overloaded predict() and update()

\* methods.

\*\*/

class BranchPredictor

{

public:

BranchPredictor() : correct\_predictions(0), incorrect\_predictions(0) {};

~BranchPredictor() {};

virtual bool predict(ADDRINT ip, ADDRINT target) = 0;

virtual void update(bool predicted, bool actual, ADDRINT ip, ADDRINT target) = 0;

virtual string getName() = 0;

UINT64 getNumCorrectPredictions() { return correct\_predictions; }

UINT64 getNumIncorrectPredictions() { return incorrect\_predictions; }

void resetCounters() { correct\_predictions = incorrect\_predictions = 0; };

protected:

void updateCounters(bool predicted, bool actual) {

if (predicted == actual)

correct\_predictions++;

else

incorrect\_predictions++;

};

private:

UINT64 correct\_predictions;

UINT64 incorrect\_predictions;

};

class StaticATPredictor : public BranchPredictor

{

public:

StaticATPredictor() : BranchPredictor(){};

~StaticATPredictor() {};

virtual bool predict(ADDRINT ip, ADDRINT target)

{

return true;

}

virtual void update(bool predicted, bool actual, ADDRINT ip, ADDRINT target)

{

updateCounters(predicted, actual);

}

virtual string getName() {

std::ostringstream stream;

stream << "StaticATPredictor";

return stream.str();

}

};

class BTFNT : public BranchPredictor

{

public:

BTFNT() : BranchPredictor(){};

~BTFNT() {};

virtual bool predict(ADDRINT ip, ADDRINT target)

{

return (target<ip);

}

virtual void update(bool predicted, bool actual, ADDRINT ip, ADDRINT target)

{

updateCounters(predicted, actual);

}

virtual string getName() {

std::ostringstream stream;

stream << "BTFNT";

return stream.str();

}

};

class NbitPredictor : public BranchPredictor

{

public:

NbitPredictor(unsigned index\_bits\_, unsigned cntr\_bits\_)

: BranchPredictor(), index\_bits(index\_bits\_), cntr\_bits(cntr\_bits\_) {

table\_entries = 1 << index\_bits;

TABLE = new unsigned long long[table\_entries];

memset(TABLE, 0, table\_entries \* sizeof(\*TABLE));

COUNTER\_MAX = (1 << cntr\_bits) - 1;

};

~NbitPredictor() { delete TABLE; };

virtual bool predict(ADDRINT ip, ADDRINT target) {

unsigned int ip\_table\_index = ip % table\_entries;

unsigned long long ip\_table\_value = TABLE[ip\_table\_index];

unsigned long long prediction = ip\_table\_value >> (cntr\_bits - 1);

return (prediction != 0);

};

virtual void update(bool predicted, bool actual, ADDRINT ip, ADDRINT target) {

unsigned int ip\_table\_index = ip % table\_entries;

if (actual) {

if (TABLE[ip\_table\_index] < COUNTER\_MAX)

TABLE[ip\_table\_index]++;

} else {

if (TABLE[ip\_table\_index] > 0)

TABLE[ip\_table\_index]--;

}

updateCounters(predicted, actual);

};

virtual string getName() {

std::ostringstream stream;

stream << "Nbit-" << pow(2.0,double(index\_bits)) / 1024.0 << "K-" << cntr\_bits;

return stream.str();

}

private:

unsigned int index\_bits, cntr\_bits;

unsigned int COUNTER\_MAX;

/\* Make this unsigned long long so as to support big numbers of cntr\_bits. \*/

unsigned long long \*TABLE;

unsigned int table\_entries;

};

class NbitPredictorcustom : public BranchPredictor

{

public:

NbitPredictorcustom(unsigned index\_bits\_, unsigned cntr\_bits\_)

: BranchPredictor(), index\_bits(index\_bits\_), cntr\_bits(cntr\_bits\_) {

table\_entries = 1 << index\_bits;

TABLE = new unsigned long long[table\_entries];

memset(TABLE, 0, table\_entries \* sizeof(\*TABLE));

COUNTER\_MAX = (1 << cntr\_bits) - 1;

};

~NbitPredictorcustom() { delete TABLE; };

virtual bool predict(ADDRINT ip, ADDRINT target) {

unsigned int ip\_table\_index = ip % table\_entries;

unsigned long long ip\_table\_value = TABLE[ip\_table\_index];

unsigned long long prediction = ip\_table\_value >> (cntr\_bits - 1);

return (prediction != 0);

};

virtual void update(bool predicted, bool actual, ADDRINT ip, ADDRINT target) {

unsigned int ip\_table\_index = ip % table\_entries;

if (actual) {

if (TABLE[ip\_table\_index] < COUNTER\_MAX)

{

if(TABLE[ip\_table\_index]==1)

{

TABLE[ip\_table\_index]=3;

}

}

else

{

TABLE[ip\_table\_index]++;

}

} else {

if (TABLE[ip\_table\_index] > 0)

{

if(TABLE[ip\_table\_index]==2)

{

TABLE[ip\_table\_index]=0;

}

else

{

TABLE[ip\_table\_index]--;

}

}

}

updateCounters(predicted, actual);

};

virtual string getName() {

std::ostringstream stream;

stream << "NbitPredictorcustom-" << pow(2.0,double(index\_bits)) / 1024.0 << "K-" << cntr\_bits;

return stream.str();

}

private:

unsigned int index\_bits, cntr\_bits;

unsigned int COUNTER\_MAX;

/\* Make this unsigned long long so as to support big numbers of cntr\_bits. \*/

unsigned long long \*TABLE;

unsigned int table\_entries;

};

// Fill in the BTB implementation ...

class BTBPredictor : public BranchPredictor

{

public:

BTBPredictor(int btb\_lines, int btb\_assoc)

: table\_lines(btb\_lines/btb\_assoc), table\_assoc(btb\_assoc)

{

correctTarget\_predictions=0;

TABLE= new unsigned long long\*\*[table\_lines];

if (!TABLE) {printf("error"); exit(1);}

for(int i=0; i<table\_lines; i++)

{

TABLE[i]= new unsigned long long \*[table\_assoc];

if (!TABLE[i]) {printf("error"); exit(1);}

for (int j=0; j<table\_assoc; j++)

{

TABLE[i][j]= new unsigned long long[2];

if (!TABLE[i][j]) {printf("error"); exit(1);}

memset(TABLE[i][j],0, 2 \* sizeof(TABLE[0][0][0]));

}

}

/\* ... fill me ... \*/

}

~BTBPredictor()

{

for (int i=0; i<table\_lines; i++)

{

for (int j=0; j<table\_assoc; j++) {delete TABLE[i][j];}

delete TABLE[i];

}

delete TABLE;

/\* ... fill me ... \*/

}

virtual bool predict(ADDRINT ip, ADDRINT target)

{

unsigned int ip\_table\_index = ip % table\_lines;

for (int i=0; i<table\_assoc; i++)

{

if (TABLE[ip\_table\_index][i][0] == ip)

return true;

}

/\* ... fill me ... \*/

return false;

}

virtual void update(bool predicted, bool actual, ADDRINT ip, ADDRINT target)

{

updateCounters(predicted,actual);

unsigned int ip\_table\_index = ip % table\_lines;

if (actual &&!predicted)

{

//branch taken but not predicted,

//insert branch and it's destination into the btb cache

for (int i=0; i<table\_assoc; i++)

{

if (TABLE[ip\_table\_index][i][0]==0)

{

TABLE[ip\_table\_index][i][0]=ip;

TABLE[ip\_table\_index][i][1]=target;

return ;

}

}

int replace = std::rand() % table\_assoc;

TABLE[ip\_table\_index][replace][0]=ip;

TABLE[ip\_table\_index][replace][1]=target;

}

else if (!actual &&predicted)

{ //

for (int i=0; i<table\_assoc; i++)

{

if (TABLE[ip\_table\_index][i][0]==ip)

{

TABLE[ip\_table\_index][i][0]=0;

TABLE[ip\_table\_index][i][1]=0;

return ;

}

}

}

else if (actual &&predicted)

{

for (int i=0; i<table\_assoc; i++)

{

if (TABLE[ip\_table\_index][i][0]==ip)

{

if (TABLE[ip\_table\_index][i][1]==target) {correctTarget\_predictions++;}

else {TABLE[ip\_table\_index][i][1]=target;}

return;

}

}

}

/\* ... fill me ... \*/

}

virtual string getName() {

std::ostringstream stream;

stream << "BTB-" << table\_lines\*table\_assoc << "-" << table\_assoc;

return stream.str();

}

UINT64 getNumCorrectTargetPredictions() {

return correctTarget\_predictions;

/\* ... fill me ... \*/

//return 0;

}

private:

int table\_lines, table\_assoc;

unsigned long long \*\*\*TABLE;

UINT64 correctTarget\_predictions;

};

class LocalHistoryTwoLevel : public BranchPredictor {

private:

unsigned int PHTindex, PHTcntr;

unsigned int BHTindex, BHTcntr;

unsigned int PHTentries, BHTentries;

unsigned long long int \*PHT;

unsigned long long int \*BHT;

unsigned int PHTmod, BHTmod;

unsigned int COUNTER\_MAX;

public:

LocalHistoryTwoLevel(unsigned int PHTindex,

unsigned int PHTcntr,

unsigned int BHTindex,

unsigned int BHTcntr) : BranchPredictor() {

this->PHTindex = PHTindex;

this->PHTcntr = PHTcntr;

this->BHTindex = BHTindex;

this->BHTcntr = BHTcntr;

this->PHTentries = 1 << this->PHTindex;

this->BHTentries = 1 << this->BHTindex;

this->PHT = new unsigned long long int [PHTentries];

this->BHT = new unsigned long long int [BHTentries];

memset(PHT, 0, PHTentries \* sizeof(\*PHT));

memset(BHT, 0, BHTentries \* sizeof(\*BHT));

/\* Need (PHTindex - BHTcntr) bits shifted BHTcntr bits left \*/

this->PHTmod = 1 << (this->PHTindex - this->BHTcntr);

this->BHTmod = this->BHTentries;

/\* For 2bit PHT predictor \*/

COUNTER\_MAX = (1 << this->PHTcntr) - 1;

}

~LocalHistoryTwoLevel() {

delete [] PHT;

delete [] BHT;

}

virtual bool predict(ADDRINT ip, ADDRINT target) {

unsigned int PHTinx, BHTinx;

PHTinx = (ip % this->PHTmod) << this->BHTcntr;

BHTinx = ip % this->BHTmod;

/\*\*

\* 1 << BHTcntr = max number ( + 1) for a BHTentry.

\* So with mod we restrict the non zero bits we get

\* from the BHT, which are at most BHTcntr.

\*/

PHTinx = PHTinx | (BHT[BHTinx] % (1 << this->BHTcntr));

return ((PHT[PHTinx] >> (this->PHTcntr - 1)) != 0);

}

virtual void update(bool predicted, bool actual, ADDRINT ip, ADDRINT target) {

unsigned int PHTinx, BHTinx;

PHTinx = (ip % this->PHTmod) << this->BHTcntr;

BHTinx = ip % this->BHTmod;

/\*\*

\* 1 << BHTcntr = max number ( + 1) for a BHTentry.

\* So with mod we restrict the non zero bits we get

\* from the BHT, which are at most BHTcntr.

\*/

PHTinx = PHTinx | (BHT[BHTinx] % (1 << this->BHTcntr));

/\* Shift out oldest outcome \*/

BHT[BHTinx] = BHT[BHTinx] << 1;

/\*\*

\* To shift in new outcome we use this:

\* if actual is one then just increment BHTentry by 1

\* which is equivalent to place bit #0 to 1,

\* because previously we had shifted left so bit #0 is 0.

\* else do nothing for BHTentry

\*/

if (actual) {

/\* Shift in newest outcome \*/

BHT[BHTinx]++;

if (PHT[PHTinx] < COUNTER\_MAX)

PHT[PHTinx]++;

} else {

if (PHT[PHTinx] > 0)

PHT[PHTinx]--;

}

updateCounters(predicted, actual);

}

virtual string getName() {

std::ostringstream stream;

stream << "LocalHistoryTwoLevel-"

<< "PHT-" << pow(2.0, double(PHTindex)) / 1024.0 << "K-" << PHTcntr << "-"

<< "BHT-" << pow(2.0, double(BHTindex)) / 1024.0 << "K-" << BHTcntr;

return stream.str();

}

};

class GlobalHistoryTwoLevel : public BranchPredictor {

private:

unsigned int PHTindex, PHTcntr;

unsigned int BHRcntr;

unsigned int PHTentries;

unsigned long long int \*PHT;

unsigned long long int BHR;

unsigned int PHTmod;

unsigned int COUNTER\_MAX;

public:

GlobalHistoryTwoLevel(unsigned int PHTindex,

unsigned int PHTcntr,

unsigned int BHRcntr

) : BranchPredictor() {

this->PHTindex = PHTindex;

this->PHTcntr = PHTcntr;

this->BHRcntr = BHRcntr;

this->PHTentries = 1 << this->PHTindex;

this->PHT = new unsigned long long int [PHTentries];

memset(PHT, 0, PHTentries \* sizeof(\*PHT));

this->BHR = 0;

/\* Need (PHTindex - BHRcntr) bits shifted BHRcntr bits left \*/

this->PHTmod = 1 << (this->PHTindex - this->BHRcntr);

/\* For Nbit PHT predictor \*/

COUNTER\_MAX = (1 << this->PHTcntr) - 1;

}

~GlobalHistoryTwoLevel() {

delete [] PHT;

}

virtual bool predict(ADDRINT ip, ADDRINT target) {

unsigned int PHTinx;

PHTinx = (ip % this->PHTmod) << this->BHRcntr;

/\*\*

\* 1 << BHRcntr = max number ( + 1) for a BHRentry.

\* So with mod we restrict the non zero bits we get

\* from the BHR, which are at most BHTcntr.

\*/

PHTinx = PHTinx | (BHR % (1 << this->BHRcntr));

return ((PHT[PHTinx] >> (this->PHTcntr - 1)) != 0);

}

virtual void update(bool predicted, bool actual, ADDRINT ip, ADDRINT target) {

unsigned int PHTinx;

PHTinx = (ip % this->PHTmod) << this->BHRcntr;

/\*\*

\* 1 << BHRcntr = max number ( + 1) for BHR value.

\* So with mod we restrict the non zero bits we get

\* from the BHR, which are at most BHRcntr.

\*/

PHTinx = PHTinx | (BHR % (1 << this->BHRcntr));

/\* Shift out oldest outcome \*/

BHR = BHR << 1;

/\*\*

\* To shift in new outcome we use this:

\* if actual is one then just increment BHR by 1

\* which is equivalent to place bit #0 to 1,

\* because previously we had shifted left so bit #0 is 0.

\* else do nothing for BHR

\*/

if (actual) {

/\* Shift in newest outcome \*/

BHR++;

if (PHT[PHTinx] < COUNTER\_MAX)

PHT[PHTinx]++;

} else {

if (PHT[PHTinx] > 0)

PHT[PHTinx]--;

}

updateCounters(predicted, actual);

}

virtual string getName() {

std::ostringstream stream;

stream << "GlobalHistoryTwoLevel-"

<< "PHT-" << pow(2.0, double(PHTindex)) / 1024.0 << "K-" << PHTcntr << "-"

<< "BHR-" << BHRcntr;

return stream.str();

}

};

class TournamentLocalNbit : public BranchPredictor {

private:

unsigned int index, cntr;

unsigned int entries;

unsigned long long int \*table;

unsigned int COUNTER\_MAX;

bool nbitPred, localPred;

NbitPredictor \*nbit;

LocalHistoryTwoLevel \*local;

public:

TournamentLocalNbit(unsigned int index,

unsigned int cntr,

unsigned int nbitIndex,

unsigned int nbitCntr,

unsigned int PHTindex,

unsigned int PHTcntr,

unsigned int BHTindex,

unsigned int BHTcntr

) : BranchPredictor() {

this->index = index;

this->cntr = cntr;

this->entries = 1 << this->index;

table = new unsigned long long int [this->entries];

memset(table, 0, sizeof(\*table) \* this->entries);

COUNTER\_MAX = (1 << this->cntr) - 1;

nbitPred = false;

localPred = false;

nbit = new NbitPredictor(nbitIndex, nbitCntr);

local = new LocalHistoryTwoLevel(PHTindex, PHTcntr, BHTindex, BHTcntr);

}

~TournamentLocalNbit() {

delete [] table;

}

virtual bool predict(ADDRINT ip, ADDRINT target) {

nbitPred = nbit->predict(ip, target);

localPred = local->predict(ip, target);

unsigned int index = ip % this->entries;

if ((table[index] >> (this->cntr - 1)) > 0) {

return localPred;

} else {

return nbitPred;

}

return false;

}

virtual void update(bool predicted, bool actual, ADDRINT ip, ADDRINT target) {

nbit->update(nbitPred, actual, ip, target);

local->update(localPred, actual, ip, target);

/\* For meta \*/

unsigned int index = ip % this->entries;

if (nbitPred != localPred) {

if (nbitPred == actual && table[index] > 0)

table[index]--;

else if (localPred == actual && table[index] < COUNTER\_MAX)

table[index]++;

}

updateCounters(predicted, actual);

}

virtual string getName() {

std::ostringstream stream;

stream << "Tournament-" << nbit->getName() << "-" << local->getName();

return stream.str();

}

};

class TournamentGlobalNbit : public BranchPredictor {

private:

unsigned int index, cntr;

unsigned int entries;

unsigned long long int \*table;

unsigned int COUNTER\_MAX;

bool nbitPred, globalPred;

NbitPredictor \*nbit;

GlobalHistoryTwoLevel \*global;

public:

TournamentGlobalNbit(unsigned int index,

unsigned int cntr,

unsigned int nbitIndex,

unsigned int nbitCntr,

unsigned int PHTindex,

unsigned int PHTcntr,

unsigned int BHRcntr

) : BranchPredictor() {

this->index = index;

this->cntr = cntr;

this->entries = 1 << this->index;

table = new unsigned long long int [this->entries];

memset(table, 0, sizeof(\*table) \* this->entries);

COUNTER\_MAX = (1 << this->cntr) - 1;

nbitPred = false;

globalPred = false;

nbit = new NbitPredictor(nbitIndex, nbitCntr);

global = new GlobalHistoryTwoLevel(PHTindex, PHTcntr, BHRcntr);

}

~TournamentGlobalNbit() {

delete [] table;

}

virtual bool predict(ADDRINT ip, ADDRINT target) {

nbitPred = nbit->predict(ip, target);

globalPred = global->predict(ip, target);

unsigned int index = ip % this->entries;

if ((table[index] >> (this->cntr - 1)) > 0) {

return globalPred;

} else {

return nbitPred;

}

return false;

}

virtual void update(bool predicted, bool actual, ADDRINT ip, ADDRINT target) {

nbit->update(nbitPred, actual, ip, target);

global->update(globalPred, actual, ip, target);

/\* For meta \*/

unsigned int index = ip % this->entries;

if (nbitPred != globalPred) {

if (nbitPred == actual && table[index] > 0)

table[index]--;

else if (globalPred == actual && table[index] < COUNTER\_MAX)

table[index]++;

}

updateCounters(predicted, actual);

}

virtual string getName() {

std::ostringstream stream;

stream << "Tournament-" << nbit->getName() << "-" << global->getName();

return stream.str();

}

};

class TournamentLocalGlobal : public BranchPredictor {

private:

unsigned int index, cntr;

unsigned int entries;

unsigned long long int \*table;

unsigned int COUNTER\_MAX;

bool localPred, globalPred;

//NbitPredictor \*nbit;

LocalHistoryTwoLevel \*local;

GlobalHistoryTwoLevel \*global;

public:

TournamentLocalGlobal(unsigned int index,

unsigned int cntr,

unsigned int PHTindex,

unsigned int PHTcntr,

unsigned int BHTindex,

unsigned int BHTcntr,

unsigned int PHTGindex,

unsigned int PHTGcntr,

unsigned int BHRcntr

) : BranchPredictor() {

this->index = index;

this->cntr = cntr;

this->entries = 1 << this->index;

table = new unsigned long long int [this->entries];

memset(table, 0, sizeof(\*table) \* this->entries);

COUNTER\_MAX = (1 << this->cntr) - 1;

//nbitPred = false;

localPred = false;

globalPred = false;

//nbit = new NbitPredictor(nbitIndex, nbitCntr);

global = new GlobalHistoryTwoLevel(PHTGindex, PHTGcntr, BHRcntr);

local = new LocalHistoryTwoLevel(PHTindex, PHTcntr, BHTindex, BHTcntr);

}

~TournamentLocalGlobal() {

delete [] table;

}

virtual bool predict(ADDRINT ip, ADDRINT target) {

//nbitPred = nbit->predict(ip, target);

globalPred = global->predict(ip, target);

localPred = local->predict(ip, target);

unsigned int index = ip % this->entries;

if ((table[index] >> (this->cntr - 1)) > 0) {

return localPred;

} else {

return globalPred;

}

return false;

}

virtual void update(bool predicted, bool actual, ADDRINT ip, ADDRINT target) {

//nbit->update(nbitPred, actual, ip, target);

local->update(localPred, actual, ip, target);

global->update(globalPred, actual, ip, target);

/\* For meta \*/

unsigned int index = ip % this->entries;

if (localPred != globalPred) {

if (globalPred == actual && table[index] > 0)

table[index]--;

else if (localPred == actual && table[index] < COUNTER\_MAX)

table[index]++;

}

updateCounters(predicted, actual);

}

virtual string getName() {

std::ostringstream stream;

stream << "Tournament-" << local->getName() << "-" << global->getName();

return stream.str();

}

};

class ALPHA : public BranchPredictor {

private:

unsigned int PHTindex, PHTcntr;

unsigned int BHRcntr;

unsigned int PHTentries;

unsigned long long int \*PHT;

unsigned long long int BHR;

unsigned int PHTmod;

unsigned int COUNTER\_MAX;

bool localPred, globalPred;

//NbitPredictor \*nbit;

LocalHistoryTwoLevel \*local;

GlobalHistoryTwoLevel \*global;

public:

ALPHA(unsigned int PHTindex,

unsigned int PHTcntr,

unsigned int BHRcntr,

unsigned int PHTLindex,

unsigned int PHTLcntr,

unsigned int BHTLindex,

unsigned int BHTLcntr,

unsigned int PHTGindex,

unsigned int PHTGcntr,

unsigned int BHRGcntr

) : BranchPredictor() {

this->PHTindex = PHTindex;

this->PHTcntr = PHTcntr;

this->BHRcntr = BHRcntr;

this->PHTentries = 1 << this->PHTindex;

this->PHT = new unsigned long long int [PHTentries];

memset(PHT, 0, PHTentries \* sizeof(\*PHT));

this->BHR = 0;

/\* Need (PHTindex - BHRcntr) bits shifted BHRcntr bits left \*/

this->PHTmod = 1 << (this->PHTindex - this->BHRcntr);

/\* For Nbit PHT predictor \*/

COUNTER\_MAX = (1 << this->PHTcntr) - 1;

//nbitPred = false;

localPred = false;

globalPred = false;

//nbit = new NbitPredictor(nbitIndex, nbitCntr);

global = new GlobalHistoryTwoLevel(PHTGindex, PHTGcntr, BHRGcntr);

local = new LocalHistoryTwoLevel(PHTLindex, PHTLcntr, BHTLindex, BHTLcntr);

}

~ALPHA() {

delete [] PHT;

}

virtual bool predict(ADDRINT ip, ADDRINT target) {

//nbitPred = nbit->predict(ip, target);

//I predict for P0 and P1

globalPred = global->predict(ip, target);

localPred = local->predict(ip, target);

//custumize my indexes

unsigned int PHTinx;

PHTinx = (ip % this->PHTmod) << this->BHRcntr;

PHTinx = PHTinx | (BHR % (1 << this->BHRcntr));

//use the truth table according to P0 and P1 values

if ((PHT[PHTinx] >> (this->PHTcntr - 1)) > 0) {

return globalPred;

} else {

return localPred;

}

return false;

}

virtual void update(bool predicted, bool actual, ADDRINT ip, ADDRINT target) {

//nbit->update(nbitPred, actual, ip, target);

local->update(localPred, actual, ip, target);

global->update(globalPred, actual, ip, target);

/\* For meta \*/

unsigned int PHTinx;

PHTinx = (ip % this->PHTmod) << this->BHRcntr;

PHTinx = PHTinx | (BHR % (1 << this->BHRcntr));

BHR = BHR << 1;

if(actual) {BHR++;}

if (localPred != globalPred) {

if (localPred == actual && PHT[PHTinx] > 0)

PHT[PHTinx]--;

else if (globalPred == actual && PHT[PHTinx] < COUNTER\_MAX)

//BHR++;

PHT[PHTinx]++;

}

updateCounters(predicted, actual);

}

virtual string getName() {

std::ostringstream stream;

stream << "ALPHA-" << "PHT-" << pow(2.0, double(PHTindex)) / 1024.0 << "K-" << PHTcntr << "-"

<< "BHR-" << BHRcntr << local->getName() << "-" << global->getName();

return stream.str();

}

};

#endif

From \_branch.cpp

VOID InitPredictors()

{

// N-bit predictors

//for (int i=1; i <= 1; i++) {

//NbitPredictor \*nbitPred = new NbitPredictor(14, i);

//branch\_predictors.push\_back(nbitPred);

//NbitPredictorcustom \*nbitPred = new NbitPredictorcustom(14, 2);

//branch\_predictors.push\_back(nbitPred);

//}

//NbitPredictor \*nbitPred = new NbitPredictor(15, 1);

//branch\_predictors.push\_back(nbitPred);

//nbitPred = new NbitPredictor(14, 2);

//branch\_predictors.push\_back(nbitPred);

//NbitPredictorcustom \*nbitPred = new NbitPredictorcustom(14, 2);

//branch\_predictors.push\_back(nbitPred);

//nbitPred = new NbitPredictor(13, 4);

//branch\_predictors.push\_back(nbitPred);

/\* BTBPredictor \*btbPred = new BTBPredictor(512, 1);

btb\_predictors.push\_back(btbPred);

btbPred = new BTBPredictor(512, 2);

btb\_predictors.push\_back(btbPred);

btbPred = new BTBPredictor(256, 2);

btb\_predictors.push\_back(btbPred);

btbPred = new BTBPredictor(256, 4);

btb\_predictors.push\_back(btbPred);

btbPred = new BTBPredictor(128, 4);

btb\_predictors.push\_back(btbPred);

btbPred = new BTBPredictor(64, 8);

btb\_predictors.push\_back(btbPred);\*/

//Pentium-M predictor

PentiumMBranchPredictor \*pentiumPredictor = new PentiumMBranchPredictor();

branch\_predictors.push\_back(pentiumPredictor);

StaticATPredictor \*staticT = new StaticATPredictor();

BTFNT \*btfnt = new BTFNT();

NbitPredictor \*nbitPred = new NbitPredictor(13, 4);

LocalHistoryTwoLevel \*local1 = new LocalHistoryTwoLevel(13, 2, 11, 8);

LocalHistoryTwoLevel \*local2 = new LocalHistoryTwoLevel(13, 2, 12, 4);

GlobalHistoryTwoLevel \*global1 = new GlobalHistoryTwoLevel(14, 2, 5);

GlobalHistoryTwoLevel \*global2 = new GlobalHistoryTwoLevel(14, 2, 10);

GlobalHistoryTwoLevel \*global3 = new GlobalHistoryTwoLevel(13, 4, 5);

GlobalHistoryTwoLevel \*global4 = new GlobalHistoryTwoLevel(13, 4, 10);

TournamentLocalNbit \*tour1 =

new TournamentLocalNbit(10, 2, 12, 4, 12, 2, 11, 4);

TournamentLocalNbit \*tour2 =

new TournamentLocalNbit(11, 2, 13, 2, 11, 4, 11, 4);

TournamentGlobalNbit \*tour3 =

new TournamentGlobalNbit(10, 2, 12, 4, 13, 2, 5);

TournamentGlobalNbit \*tour4 =

new TournamentGlobalNbit(11, 2, 13, 2, 13, 2, 5);

branch\_predictors.push\_back(nbitPred);

branch\_predictors.push\_back(staticT);

branch\_predictors.push\_back(btfnt);

branch\_predictors.push\_back(local1);

branch\_predictors.push\_back(local2);

branch\_predictors.push\_back(global1);

branch\_predictors.push\_back(global2);

branch\_predictors.push\_back(global3);

branch\_predictors.push\_back(global4);

branch\_predictors.push\_back(tour1);

branch\_predictors.push\_back(tour2);

branch\_predictors.push\_back(tour3);

branch\_predictors.push\_back(tour4);

//TournamentLocalGlobal \*tour111 =

//new TournamentLocalGlobal(11, 2, 12, 2, 12, 2, 13, 2, 10);

TournamentLocalGlobal \*tour21 =

new TournamentLocalGlobal(11, 2, 12, 2, 11, 4, 13, 2, 10);

//TournamentLocalGlobal \*tour31 =

//new TournamentLocalGlobal(11, 2, 11, 4, 11, 4, 13, 2, 10);

TournamentLocalGlobal \*tour41 =

new TournamentLocalGlobal(11, 2, 12, 2, 11, 4, 12, 4, 10);

//TournamentLocalGlobal \*tour5 =

//new TournamentLocalGlobal(11, 2, 11, 2, 11, 4, 12, 4, 10);

//TournamentLocalGlobal \*tour6 =

//new TournamentLocalGlobal(11, 2, 12, 2, 11, 4, 13, 2, 5);

//TournamentLocalGlobal \*tour7 =

//new TournamentLocalGlobal(10, 2, 12, 2, 12, 2, 13, 2, 10);

TournamentLocalGlobal \*tour8 =

new TournamentLocalGlobal(10, 2, 12, 2, 11, 4, 13, 2, 5);

//TournamentLocalGlobal \*tour9 =

//new TournamentLocalGlobal(10, 2, 12, 2, 11, 4, 13, 2, 10);

//TournamentLocalGlobal \*tour10 =

//new TournamentLocalGlobal(10, 2, 12, 2, 12, 4, 12, 4, 10);

//TournamentLocalGlobal \*tour11 =

//new TournamentLocalGlobal(10, 2, 11, 4, 11, 4, 12, 4, 10);

//TournamentLocalGlobal \*tour12 =

//new TournamentLocalGlobal(10, 2, 12, 2, 11, 4, 13, 2, 5);

// branch\_predictors.push\_back(tour111);

branch\_predictors.push\_back(tour21);

// branch\_predictors.push\_back(tour31);

branch\_predictors.push\_back(tour41);

//branch\_predictors.push\_back(tour5);

//branch\_predictors.push\_back(tour6);

//branch\_predictors.push\_back(tour7);

branch\_predictors.push\_back(tour8);

//branch\_predictors.push\_back(tour9);

//branch\_predictors.push\_back(tour10);

//branch\_predictors.push\_back(tour11);

//branch\_predictors.push\_back(tour12);

ALPHA \*touralpha = new ALPHA(12, 2, 12, 10, 3, 10, 10, 12, 2, 12);

branch\_predictors.push\_back(touralpha);

//4.4

/\*RAS \*ras = new RAS(4);

ras\_vec.push\_back(ras);

//btbPred = new RAS(4);

//ras\_vec.push\_back(btbPred);

ras = new RAS(8);

ras\_vec.push\_back(ras);

ras = new RAS(16);

ras\_vec.push\_back(ras);

ras = new RAS(24);

ras\_vec.push\_back(ras);

ras = new RAS(32);

ras\_vec.push\_back(ras);

ras = new RAS(48);

ras\_vec.push\_back(ras);

ras = new RAS(64);

ras\_vec.push\_back(ras);\*/

}