МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №3

по курсу «Архитектура современных микропроцессоров и мультипроцессоров»

ОПРЕДЕЛЕНИЕ ПАРАМЕТРОВ ДИНАМИЧЕСКОГО ПРЕДСКАЗАТЕЛЯ ПРОЦЕССОРА

Выполнил: студент 3-го курса гр. 17208

Гафиятуллин А.Р

Новосибирск, 2020

1. **ЦЕЛИ РАБОТЫ:**
   1.1. научиться определять параметры динамического предсказателя переходов микропроцессора.
2. **ЗАДАЧИ РАБОТЫ:**
   2.1. Спроектировать и написать программу, определяющую параметры предсказателя переходов (наличие локального и глобального предсказателей, размеры их истории переходов).
   2.2. Определить параметры предсказателя переходов, а также величину задержки при неправильном предсказании, для двух микропроцессоров с разными микроархитектурами. Полученные результаты сравнить с реальными характеристиками. По результатам работы сделать вывод.
3. **ХОД РАБОТЫ:**
   3.1. Для достижения цели написано три программы, реализующие 1, 2, 3 и 5 шаги микробенчмарка, описанного в работе [1] (в списке литературы):
   - программа на Си, замеряющая для различных шагов из работы [1] величину неправильных предсказаний переходов посредством библиотеки PAPI;
   - программа на Си, замеряющая среднее количество тактов на итерацию цикла посредством библиотеки PAPI для определения величины задержки при неправильном предсказании;
   - программа на Python, генерирующая и компилирующая программы на Си, а также получающая и обрабатывающая результаты, в том числе определяющая задержку при неправильном предсказании.

Листинг первой программы на Си:

```
#include <stdio.h>
#include <vector>
#include <papi.h>

constexpr unsigned long long N = 10000000;

int main() {
  #include "declarations"

  std::vector<int> event_codes = { PAPI_BR_CN, PAPI_BR_MSP };
  std::vector<long long> values(event_codes.size());
  int event_set = PAPI_NULL;

  PAPI_library_init(PAPI_VER_CURRENT);
  PAPI_create_eventset(&event_set);
```

```
  for (auto code : event_codes) {
    PAPI_add_event(event_set,code);
  }

  PAPI_start(event_set);
  for (unsigned long long i = 0; i < N; i++) {
    #include "branches"
  }
  PAPI_stop(event_set, values.data());

  printf("%.8f", static_cast<float>(values[1]) / values[0] * 100);

  PAPI_cleanup_eventset(event_set);
  PAPI_destroy_eventset(&event_set);

  return 0;
}
```

Листинг второй программы на Си:

```
#include <stdio.h>
#include <papi.h>
#include <vector>

constexpr unsigned long long N = 10000000;

int main() {
  int a = 0;
  int event_set = PAPI_NULL;
  long long int results = 0;

  PAPI_library_init(PAPI_VER_CURRENT);
  PAPI_create_eventset(&event_set);
  PAPI_add_event(event_set, PAPI_TOT_CYC);

  PAPI_start(event_set);
  for(long long i = 0; i < N; i++) {
      #include "branches"
  }
  PAPI_stop(event_set, &results);

  printf("%lf", (double)results / N);

  PAPI_cleanup_eventset(event_set);
  PAPI_destroy_eventset(&event_set);

  return 0;
}
```

Листинг программы на Python:

```
import subprocess
from subprocess import Popen, PIPE
import signal
import numpy
import time
```

```
L = 1
max_L = 128
threshold = 1000
prime_numbers = [2,   3,   5,   7,  11,  13,  17,  19,  23,  29,  31,  37,
                 41,  43,  47,  53,  59,  61,  67,  71,  73,  79,  83,  89,  97,
                 101, 103, 107, 109, 113, 127, 131, 137]
results = []
stop = False


def stop_benchmark(signal, frame):
    global stop
    print("Benchmark is stopped!")
    stop = True


signal.signal(signal.SIGINT, stop_benchmark)

log = open("1st_step.log", "w")
branches = open("branches", "w")
branches.write("if(i % L == 0) a = 0;\n")
branches.close()

#first step
print("1st step start!\n")
while not stop and L <= max_L:

    L_file = open("declarations", "w")
    L_file.write("long long int a = 1;\n")
    L_file.write("int L = " + str(L) + ";")
    L_file.close()

    subprocess.call(["g++", "benchmark.cpp", "-lpapi"])

    with Popen(["./a.out"], stdout=PIPE) as test:
        try:
            mpr = float(test.stdout.read().decode("ascii"))
        except:
            continue
        log.write(str(L) + " " + str(mpr) + "\n")
        results.append(mpr)
    L = L + 1

log.close()
```

```
L_r = numpy.argmax(results)
while results[L_r] / results[L_r - 1] < threshold:
    L_r = L_r - 1

L = L_r
print("MAXIMUM LENGTH OF HISTORY: " + str(L))
print("MPR(MAX_LENGTH) = " + str(results[L - 1]) + " %")
print("MPR(MAX_LENGTH + 1) = " + str(results[L]) + " %")

# second step
print("\n2nd step start!\n")

L_file = open("declarations", "w")
L_file.write("long long int a = 1;\n")
L_file.write("int L = " + str(L_r) + ";")
L_file.close()

L = 2 * (L_r - 1)

branches = open("branches", "w")
for i in range(1, L + 1):
    branches.write("if(i < 0) a = 1;\n")
branches.write("if(i % L == 0) a = 0;\n")
branches.close()

subprocess.call(["g++", "benchmark.cpp", "-lpapi"])

mpr = 0.0
with Popen(["./a.out"], stdout=PIPE) as test:
    try:
        mpr = float(test.stdout.read().decode("ascii"))
    except:
        print("Step 2: something went wrong!")

print("MPR = " + str(mpr) + " %")

# third step
print("\n3rd step start!\n")
L = L_r
i = 0
while prime_numbers[i] < L:
    i = i + 1
L1 = prime_numbers[i - 2]
L2 = prime_numbers[i - 1]
```

```
L_file = open("declarations", "w")
L_file.write("int L1 = " + str(L1) + ";\n")
L_file.write("int L2 = " + str(L2) + ";\n")
L_file.write("int a, b, c;\n")
L_file.close()

branches = open("branches", "w")
branches.write("if(i % L1 == 0) a = 1; else a = 0;\n")
branches.write("if(i % L2 == 0) b = 1; else b = 0;\n")
branches.write("if(a * b == 1) c = 1;\n")
branches.close()

subprocess.call(["g++", "benchmark.cpp", "-lpapi"])

with Popen(["./a.out"], stdout=PIPE) as test:
    try:
        mpr = float(test.stdout.read().decode("ascii"))
    except:
        print("Step 3: something went wrong!")

print("MPR = " + str(mpr) + " %")

# fifth step
print("\n5th step start!\n")
L3 = L_r + 1

L_file = open("declarations", "w")
L_file.write("int L3 = " + str(L3) + ";\n")
L_file.write("int a;\n")
L_file.close()

branches = open("branches", "w")
branches.write("if(i % L3 == 0) a = 1;\n")
branches.close()

subprocess.call(["g++", "benchmark.cpp", "-lpapi"])

with Popen(["./a.out"], stdout=PIPE) as test:
    try:
        mpr = float(test.stdout.read().decode("ascii"))
    except:
        print("Step 5: something went wrong!")

print("MPR for 1 branch = " + str(mpr) + " %")
```

```
branches = open("branches", "w")
branches.write("if(i % L3 == 0) a = 1;\n")
branches.write("if(i % L3 == 0) a = 1;\n")
branches.close()

subprocess.call(["g++", "benchmark.cpp", "-lpapi"])

with Popen(["./a.out"], stdout=PIPE) as test:
    try:
        mpr = float(test.stdout.read().decode("ascii"))
    except:
        print("Step 5: something went wrong!")

print("MPR for 2 branches = " + str(mpr) + " %")

# delay
print("\nDefine a delay!\n")

L_file = open("declarations", "w")
L_file.write("int L = " + str(L_r) + ";")
L_file.close()

branches = open("branches", "w")
branches.write("if(i % " + str(L_r) + ") a = 1;\n")
branches.close()

subprocess.call(["g++", "delay.cpp", "-lpapi"])

mp_cycles = 0
with Popen(["./a.out"], stdout=PIPE) as test:
    try:
        mp_cycles = float(test.stdout.read().decode("ascii"))
    except:
        print("Delay: something went wrong!")

branches = open("branches", "w")
branches.write("if(i % " + str(L_r - 1) + ") a = 1;\n")
branches.close()

subprocess.call(["g++", "delay.cpp", "-lpapi"])

p_cycles = 0
with Popen(["./a.out"], stdout=PIPE) as test:
    try:
        p_cycles = float(test.stdout.read().decode("ascii"))
```

```
    except:
        print("Delay: something went wrong!")

print("Delay = " + str(int((mp_cycles - p_cycles) * (L_r - 1))) + " cycles")
```

3.2. **Моменты, требующие пояснения**:

- Результаты работы шагов 1, 2, 3 и 5 программы интерпретировать нужно по описанию, которое дано в работе [1];
- Так как ошибки предсказания случаются не каждую итерацию цикла для длины истории, которая была выяснена на предыдущих шагах, то задержка усредняется по всем итерациям. Поэтому, чтобы все-таки получить задержку именно для неправильных предсказаний, **усредненная разница тактов умножается на длину истории**. Как будет показано далее, это неплохо приближает результаты к тем, которые были получены Агнером Фогом.

3.3. Описание тестируемых архитектур:

- **Intel Coffee Lake**
  - **3.3..1. Представитель:** Intel® Core™ i7-9700F Processor, 12M Cache, up to 4.70 GHz
- **Intel Kaby Lake**
  - **3.3..1. Представитель:** Intel® Core™ i5-7200U Processor, 3M Cache, up to 3.10 GHz

К сожалению, никакой информации о работе предсказателей переходов в этих архитектурах найти не удалось (Intel их не публикует), кроме величины задержки.

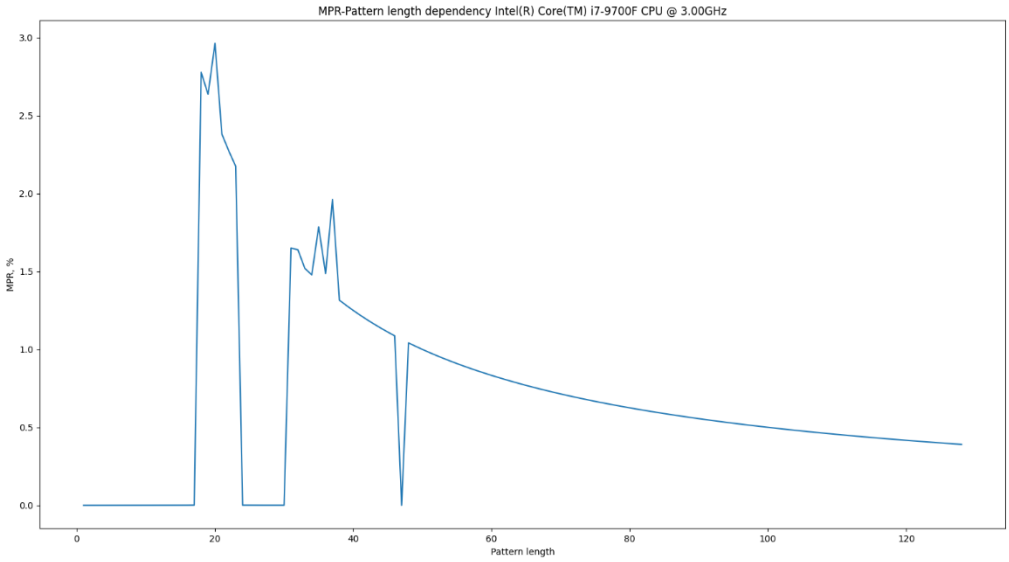
Величина задержки при неправильном предсказании перехода: 15-20 тактов (Агнер Фог).

3.4. Результаты:

- **Intel Coffee Lake:**

Шаг 1.

График зависимости величины неправильных переходов от длины шаблона:



Результаты выдачи программы в характерной точке:

1st step start!

MAXIMUM LENGTH OF HISTORY: 17

MPR(MAX_LENGTH) = 0.001195 %

MPR(MAX_LENGTH + 1) = 2.77811098 %

Зафиксировано резкое увеличение процента неправильных предсказаний переходов при длине шаблона равной 18, значит длина истории переходов равна примерно 17.

Шаг 2.

Результаты выдачи программы:

2nd step start!

MPR = 0.000325 %

На прошлом шаге была вычислена длина истории переходов, однако не ясно, локальный или глобальный предсказатель, ведь в истории переходов так же может участвовать и сам цикл (то есть условие в шапке цикла, которое организует его).

Чтобы это выяснить, нужно «забить» историю предсказателя ненужной информацией. Это было сделано так же, как и в работе [1].

Как можно заметить, процент неправильных предсказаний переходов остался очень маленьким (выдача программы). **Это говорит нам о том, что сейчас сработал локальный предсказатель, но не говорит о том, что глобального предсказателя нет вообще.**

Шаг 3.

Результаты выдачи программы:

3rd step start!

MPR = 3.0548203 %

Этот шаг должен показать количество бит глобального предсказателя. Работа этого шага основана на корреляции трех разных условных переходов, причем без возможности точно предсказать последний переход только с помощью локального предсказателя.

Так как процент неправильных предсказаний переходов оказался большим(относительно), то можно сделать вывод, что **глобального предсказателя либо нет, либо он имеет однобитную длину истории.**

Шаг 5.

Результаты выдачи программы:

5th step start!

MPR for 1 branch = 2.77806616 %

MPR for 2 branches = 0.0006 %

Этот шаг должен показать наличие хотя бы одного бита глобального предсказателя. Для этого одно и то же условие дублируется и на втором предсказатель ошибаться не должен (или хотя бы намного меньше) в случае наличия глобального предсказателя.

Как видно из выдачи программы, действительно, при наличии двух одинаковых подряд идущих условий, предсказатель не ошибается на втором (увеличилось количество условных переходов, неправильных предсказаний не увеличилось, что привело к уменьшению процента).

Таким образом, можно сделать вывод, что **есть глобальный предсказатель с однобитной историей переходов**.

Задержка:

Результаты выдачи программы:

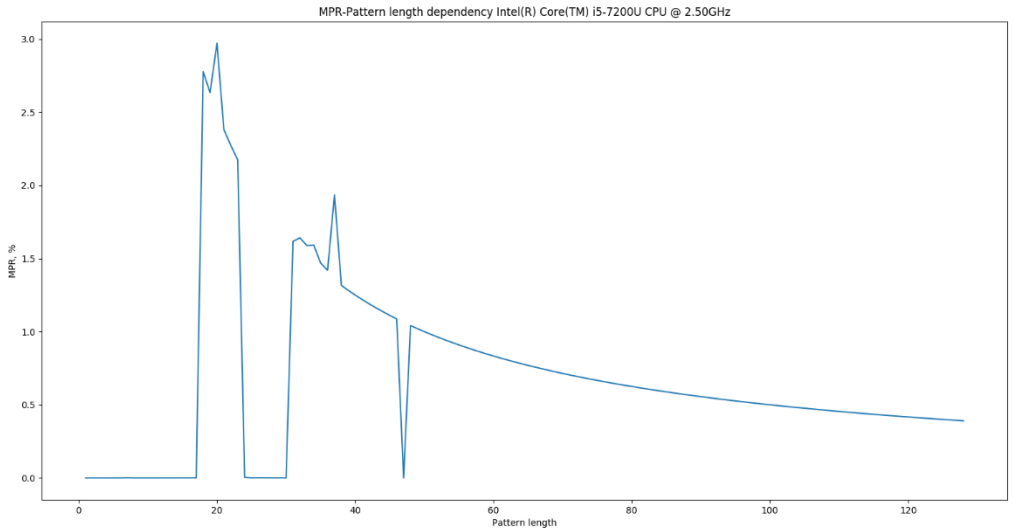
Define a delay!

Delay = 25 cycles

Величина задержки вычислена так, как было указано ранее и примерно равна той, которую указывал Агнер Фог.

- **Intel Kaby Lake**

Результаты абсолютно не отличаются, вплоть до практически точного совпадения графиков, поэтому детально их описывать нет смысла:



1st step start!

MAXIMUM LENGTH OF HISTORY: 17

MPR(MAX_LENGTH) = 0.00076 %

MPR(MAX_LENGTH + 1) = 2.77812099 %

2nd step start!

MPR = 0.000825 %

3rd step start!

MPR = 3.07853532 %

5th step start!

MPR for 1 branch = 2.77805114 %

MPR for 2 branches = 0.00398333 %

Define a delay!

Delay = 26 cycles

3. ВЫВОДЫ:

3.2. Научились определять параметры динамического предсказателя переходов микропроцессора;

3.3. Похоже, что в Lake-архитектурах Intel стоит один и тот же предсказатель переходов;

3.4. Тестируемые предсказатели оказались очень точными.

4. ЛИТЕРАТУРА:

1. Milena Milenkovic, Aleksandar Milenkovic and Jeffrey Kulick, Microbenchmarks for determining branch predictor organization.