

实验报告

开课学期： 2022秋季

课程名称： 计算机体系结构(实验)

实验名称： 实验2：分支预测器设计

学生班级： 6班

学生学号： 200110625

学生姓名： 柯炽炜

实验与创新实践教育中心印制

2022年11月

# 1. 题目分析

1. 基于BHT的分支预测

原理：使用2bit饱和计数器记录最近一次跳转的分支历史。发生跳转，计数器增1，未发生跳转，计数器减1。

工作过程分为预测和更新。

预测：利用分支指令的地址查BHT，获得相应的饱和计数器值，最高位为1预测跳转，最高位为0预测不跳转。

更新：当跳转，则计数器增1；不跳转，计数器减1

1. 基于全局历史的分支预测

原理：使用全局历史寄存器将所有的分支指令关联起来

工作过程也分为预测和更新。

预测：利用分支指令地址和ghr的value值进行哈希运算，得到的值去查PHT，根据查到的结果进行预测跳转还是不跳转

更新：通过移位进行更新。

跳转：GHR = （GHR << 1）| 1

不跳转：GHR = （GHR << 1）| 0

1. 锦标赛分支预测

原理：将多个分支预测方法结合，多个预测方法结合，取最精确的结果

工作过程：

预测：用数组将不同的分支预测器结合，根据计数器结果输出计数器的值

更新：根据实际跳转结果和预测跳转结果对每一个子预测器进行更新

1. Tage分支预测

原理：将若干个GHR位宽呈几何倍数增长的全局历史预测器结合起来做复合预测器

预测机制：将指令地址同时送往各个子预测器。除了第一个基础预测器外，其余预测器判断是否发生tag匹配。将所有发生tag匹配的子预测器选择出一个位宽最大的预测器输出其预测结果，并找到位宽次大的预测器作为备选。

更新机制：

分别更新输出结果的子预测器，usefulness和entry即可

# 2. 设计与实现

## 2.1 方案设计

BHT：

BOOL predict(ADDRINT addr)

        {

            // ������ addr �� BHT �ж�Ӧ��Ԥ����

            return m\_scnt[truncate(addr, m\_entries\_log)].isTaken();

        }

        void update(BOOL takenActually, BOOL takenPredicted, ADDRINT addr)

        {

            // TODO: Update BHT according to branch results and prediction

            if (takenActually) {

                m\_scnt[truncate(addr, m\_entries\_log)].increase();

            } else {

                m\_scnt[truncate(addr, m\_entries\_log)].decrease();

            }

        }

GHR：

// Only for TAGE: return a tag according to the specificed address

        UINT128 get\_tag(ADDRINT addr)

        {

            // TODO

            ADDRINT table\_addr = hash(addr, m\_ghr->getVal());

            return truncate(table\_addr, m\_entries\_log);

        }

        // Only for TAGE: return GHR's value

        UINT128 get\_ghr()

        {

            return m\_ghr->getVal();

        }

        UINT128 get\_ghr\_wid()

        {

            return m\_ghr->getWid();

        }

        // Only for TAGE: reset a saturating counter to default value (which is weak taken)

        void reset\_ctr(ADDRINT addr)

        {

            // TODO

            ADDRINT table\_addr = hash(addr, m\_ghr->getVal());

            m\_scnt[truncate(table\_addr, m\_entries\_log)].reset();

        }

        bool predict(ADDRINT addr)

        {

            // TODO: Produce prediction according to GHR and PHT

            ADDRINT table\_addr = hash(addr, m\_ghr->getVal());

            return m\_scnt[truncate(table\_addr, m\_entries\_log)].isTaken();

        }

        void update(bool takenActually, bool takenPredicted, ADDRINT addr)

        {

            // TODO: Update GHR and PHT according to branch results and prediction

            ADDRINT table\_addr = hash(addr, m\_ghr->getVal());

            if (takenActually) {

                m\_ghr->shiftIn(1);

                m\_scnt[truncate(table\_addr, m\_entries\_log)].increase();

            } else {

                m\_ghr->shiftIn(0);

                m\_scnt[truncate(table\_addr, m\_entries\_log)].decrease();

            }

        }

Tournament：

bool predict(ADDRINT addr) {

            if (m\_gshr->isTaken()) {

                return m\_BPs[1]->predict(addr);

            } else {

                return m\_BPs[0]->predict(addr);

            }

        }

        void update(bool takenActually, bool takenPredicted, ADDRINT addr) {

            bool subPredictResult1 = m\_BPs[0]->predict(addr);

            bool subPredictResult2 = m\_BPs[1]->predict(addr);

            // ֻ����Ԥ����1��ȷ

            if (takenActually == subPredictResult1 && takenActually != subPredictResult2) {

                m\_gshr->decrease();

            }

            // ���ֻ����Ԥ����2��ȷ

            else if (takenActually != subPredictResult1 && takenActually == subPredictResult2) {

                m\_gshr->increase();

            }

            // ������Ԥ����

            m\_BPs[0]->update(takenActually, takenPredicted, addr);

            m\_BPs[1]->update(takenActually, takenPredicted, addr);

        }

Tage：

bool predict(ADDRINT addr)

        {

            // 初始化provider\_index为0

            altpred\_indx = 0;

            provider\_indx = 0;

            // 遍历其余GHR

            for(size\_t i = 1; i < m\_tnum; i++) {

                auto curGHR = (GlobalHistoryPredictor <hash1>\*) m\_T[i];

                auto h = curGHR->get\_ghr();

                UINT128 h1 = hash1(addr, h);

                UINT128 h2 = hash2(addr, h);

                UINT128 tag = m\_tag[i][truncate(h1, m\_entries\_log)];

                // 默认参数alpha大于1

                if (tag == h2) {

                    altpred\_indx = provider\_indx;

                    provider\_indx = i;

                }

            }

            return m\_T[provider\_indx]->predict(addr);

        }

        void update(bool takenActually, bool takenPredicted, ADDRINT addr)

        {

            if (provider\_indx == 0) {

                // TODO: Update provider itself

                m\_T[provider\_indx]->update(takenActually, takenPredicted, addr);

            } else {

                auto curGHR = (GlobalHistoryPredictor <hash1>\*) m\_T[provider\_indx];

                auto h = curGHR->get\_ghr();

                auto h1 = hash1(addr, h);

                // TODO: Update provider itself

                m\_T[provider\_indx]->update(takenActually, takenPredicted, addr);

                // TODO: Update usefulness

                bool altPred = m\_T[altpred\_indx]->predict(addr);

                if (altPred != takenPredicted) {

                    if (takenPredicted == takenActually) {

                        // provider预测正确

                        if (m\_useful[provider\_indx][truncate(h1, m\_entries\_log)] < 3)

                            m\_useful[provider\_indx][truncate(h1, m\_entries\_log)]++;

                    } else {

                        if (m\_useful[provider\_indx][truncate(h1, m\_entries\_log)] > 0)

                            m\_useful[provider\_indx][truncate(h1, m\_entries\_log)]--;

                }

            }

            }

            // TODO: Reset usefulness periodically

            m\_rst\_cnt++;

            if (m\_rst\_cnt == m\_rst\_period) {

                for (size\_t i = 1; i < m\_tnum; i++) {

                    memset(m\_useful[i], 0, sizeof(UINT8)\*(1 << m\_entries\_log));

                }

                m\_rst\_cnt = 0;

            }

            // TODO: Entry replacement

            if (takenActually != takenPredicted) {

                for (size\_t i = provider\_indx + 1; i < m\_tnum; i++) {

                    if ((int)i == provider\_indx) continue;

                    auto curGHR = (GlobalHistoryPredictor <hash1>\*) m\_T[i];

                    auto h = curGHR->get\_ghr();

                    UINT128 h1 = hash1(addr, h);

                    // GHR位宽大于provider且对应entry的usefulness等于0

                    if (m\_useful[i][truncate(h1, m\_entries\_log)] == 0) {

                        curGHR->reset\_ctr(addr);

                    }

                    else if (m\_useful[i][truncate(h1, m\_entries\_log)] != 0) {

                        if (m\_useful[i][truncate(h1, m\_entries\_log)] > 0)

                            m\_useful[i][truncate(h1, m\_entries\_log)]--;

                    }

                }

            }

        }

## 2.2 实验结果及分析

*描述实验过程及各预测器的关键参数，阐述参数选择过程。计算各预测器实际大小。自行绘制并填写测试表，并对实验结果进行分析。*

