**目录**

[1. BranchPredictor 2](#_Toc532063039)

[1) 类定义 2](#_Toc532063040)

[2) 构造函数 2](#_Toc532063041)

[3) computerEnergy函数 5](#_Toc532063042)

[4) displayEnergy函数 6](#_Toc532063043)

# BranchPredictor

## 类定义

|  |
| --- |
| //分支预测器  class BranchPredictor **:**public Component **{**  public**:**  //获取参数  ParseXML **\***XML**;**  //核的标号，只有通过核的标号才能够进一步在XML中找到具体的值  int ithCore**;**  //cacti的接口  InputParameter interface\_ip**;**  //core的动态参数  CoreDynParam coredynp**;**  //与core设置的一样  double clockRate**,**executionTime**;**  double scktRatio**,** chip\_PR\_overhead**,** macro\_PR\_overhead**;**  //锦标赛的分支预测器，alpha 21264，比较常见  //全局的BPT表  ArrayST **\*** globalBPT**;**  //局部的BPT表？，没有被使用和实例化  ArrayST **\*** localBPT**;**  //局部的第一层  ArrayST **\*** L1\_localBPT**;**  //局部的第二层  ArrayST **\*** L2\_localBPT**;**  //选择器  ArrayST **\*** chooser**;**  //返回栈  ArrayST **\*** RAS**;**  //用于判断对象是否存在  bool exist**;**  BranchPredictor**(**ParseXML **\***XML\_interface**,** int ithCore\_**,** InputParameter**\*** interface\_ip\_**,**const CoreDynParam **&** dyn\_p\_**,** bool exsit**=true);**  void computeEnergy**(**bool is\_tdp**=true);**  void displayEnergy**(**uint32\_t indent **=** 0**,**int plevel **=** 100**,** bool is\_tdp**=true);**  **~**BranchPredictor**();**  **};** |

## 构造函数

* 主要功能：初始化五个arrayST类型对象的参数(interface\_ip)，分别实例化这些对象。在arrayST初始化的过程中将会计算功率。然后调用其中的函数计算面积
* 初始化一些每个arrayST都会使用并且相同的参数

|  |  |
| --- | --- |
| 参数 | 解释 |
| interface\_ip.assoc  nterface\_ip.pure\_cam | 需要设置array的相连度和类型  Assoc=1，因为分支预测器中的预测表都是直接映射  Pure\_cam=false，因为预测表不是CAM类型的存储单元 |
| interface\_ip.is\_cache  interface\_ip.pure\_ram | 这两个参数需要根据core的类型进行设置  当core是多线程时（不是多线程的时候结果相反）  Is\_cache=true，因为此时需要为表项增加tag域，指代不同的线程，这种结构是类cache的结构  Pure\_ram=false，此时不是单纯的锁存器类型的结构  **(需要注意，RAS在多线程的情况下仍旧是锁存器类型)** |
| interface\_ip.specific\_tag  interface\_ip.tag\_w | 当core的类型是多线程时，表项需要增加tag域，因此需要给出tag域的信息  Specific\_tag=1，代表使用tag，并且将会给出tag的宽度  Tag\_w= int(log2(coredynp.num\_hthreads)+ EXTRA\_TAG\_BITS)  //EXTRA\_TAG\_BITS为cacti中的参数，const int = 5 |

代码的设置情况

|  |
| --- |
| //设置相连度和类型，不是cam，同时相连度为1，直接映射  interface\_ip**.**assoc **=** 1**;**  interface\_ip**.**pure\_cam **=** **false;**  //是否为多线程  **if** **(**coredynp**.**multithreaded**)**  **{**//如果是多线程  //需要为每一个表项增加线程编号，因此会增加tag位域  //EXTRA\_TAG\_BITS为cacti中的参数，const int = 5  tag **=** int**(**log2**(**coredynp**.**num\_hthreads**)+** EXTRA\_TAG\_BITS**);**  //bool类型,是否有tag域  interface\_ip**.**specific\_tag **=** 1**;**  //tag的宽度  interface\_ip**.**tag\_w **=** tag**;**    //bool类型，是否是一个cache  interface\_ip**.**is\_cache **=** **true;**  //是否是cam类型的array  interface\_ip**.**pure\_ram **=** **false;**  **}**  **else**//如果不是多线程  **{**  //不是cache，不再是使用tag索引  interface\_ip**.**is\_cache **=** **false;**  //是一个RAM  interface\_ip**.**pure\_ram **=** **true;**  **}** |

* 设置五种ArrayST类型各自的参数（RAS需要重新设置is\_cache和pure\_ram）

每个结构的参数都是一样的，只是设置时具体值可能不同

|  |  |
| --- | --- |
| 参数 | 解释 |
| Data(临时使用) | int(ceil(XML->sys.core[ithCore].predictor.global\_predictor\_bits/8.0))  向上取整，以字节为单位，计算每个表项的宽度 |
| interface\_ip.line\_sz | Data  单位字节，每个cache line的大小 |
| interface\_ip.cache\_sz | data\*XML->sys.core[ithCore].predictor.global\_predictor\_entries  单位字节，整个cache的大小，表项数\*每个表项的大小 |
| interface\_ip.nbanks | 1  Bank的个数，预测表都没有分bank |
| interface\_ip.out\_w | interface\_ip.line\_sz\*8  单位bit，输出宽度，每个表项的宽度 |
| interface\_ip.  access\_mode | 2  访问模式，0 normal, 1 seq, 2 fast  fast(2): 数据和tag同时访问（具体见interface\_ip解析） |
| interface\_ip.  throughput | 1.0/clockRate  吞吐量 |
| interface\_ip.latency | 1.0/clockRate，延迟 |
| interface\_ip.  obj\_func\_dyn\_energy  obj\_func\_dyn\_power  obj\_func\_leak\_power  obj\_func\_cycle\_t | 0, 0, 0, 1  这些参数在cacti中未使用到 |
| interface\_ip.  num\_rw\_ports | 0  读写端口数 |
| interface\_ip.  num\_rd\_ports | 单独的读端口数，=coredynp.predictionW  prediction\_width，core的参数，一般设置为1 |
| interface\_ip.  num\_wr\_ports | 单独的写端口数，=coredynp.predictionW |
| interface\_ip.  num\_se\_rd\_ports | 0  单端读端口 |

* 实例化ArrayST对象，五种结构都基本一样，除了名称不同（以globalBPT为例）

|  |
| --- |
| //Core\_device: 可能代表着是核内设备  //enum Device\_ty的第一个，Core\_device uncore\_device，LLC\_device  //coredynp.opt\_local，一般设置为了0  //coredynp.core\_ty，machine\_type：ino/order  //在构造函数中，已经调用了计算功率的函数  //ArrayST(const InputParameter \*configure\_interface, string \_name,  // enum Device\_ty device\_ty\_, bool opt\_local\_=true,  // enum Core\_type core\_ty\_=Inorder, bool \_is\_default=true)  globalBPT **=** **new** ArrayST**(&**interface\_ip**,** "Global Predictor"**,** Core\_device**,**  coredynp**.**opt\_local**,** coredynp**.**core\_ty**);**  //计算面积  globalBPT**->**area**.**set\_area**(**globalBPT**->**area**.**get\_area**()+** globalBPT**->**local\_result**.**area**);**  //统计面积参数  area**.**set\_area**(**area**.**get\_area**()+** globalBPT**->**local\_result**.**area**);** |

## computerEnergy函数

* 函数参数bool is\_tdp。当is\_tdp为true时，计算分支预测器的静态功耗/TDP；否则计算分支预测器的动态功耗
* 分支预测器中有五个ArrayST类的对象，基本计算方式相同。以globalBPT为例，说明计算过程
  + ArrayST中包括statsDef的三个对象：tdp\_stats,rtp\_stats, stats\_t。其中stats\_t可以看作为临时变量
  + statsDef包括了三个statsComponents对象：readAc, writeAc, searchAc。其中statsCompents包括三个属性：access, hit, miss
  + 第一步：根据is\_tdp设置globalBPT中的stats\_t和tdp\_stats或者rtp\_stats中的readAc.access和writeAc.access参数
  + 当is\_tdp=true时，设置stats\_t和tdp\_stats的参数

readAc.access=coredynp.predictionW\*coredynp.BR\_duty\_cycle

writeAc.access=0\*coredynp.BR\_duty\_cycle

（BR\_duty\_cycle，core的stats，代表BR的工作时间比例，初始化为1）

* + 当is\_tdp=false时，设置stats\_t和rtp\_stats的参数

readAc.access=XML->sys.core[ithCore].branch\_instructions

（所有的分支指令都会读一遍预测器）

writeAc.access=XML->sys.core[ithCore].branch\_mispredictions + 0.1\*XML->sys.core[ithCore].branch\_instructions

（当分支预测错误或者状态迁移(10%的分支指令)的时候需要写一次预测器）

对于RAS对象，设置有所不同，此时读写次数均为函数调用次数

readAc.access = XML->sys.core[ithCore].function\_calls

writeAc.access = XML->sys.core[ithCore].function\_calls

* + 第二步：将globalBPT的参数power\_t重新初始化，初始化为0

power\_t：powerDef (cacti/cacti\_interface.h:87)

powerDef有三个powerComponents类的对象：

readOp,writeOp, searchOp（CAM才有）

powerComponents的属性值：dynamic, leakage, gate\_leakage, short\_circuit, longer\_channel\_leakage, power\_gated\_leakage, power\_gated\_with\_long\_channel\_leakage

* + 第三步：计算动态功耗，并保存在stats\_t的readOp.dynamic中

dynamic=local\_result.power.readOp.dynamic\* stats\_t.readAc.access +local\_result.power.writeOp.dynamic\* stats\_t.writeAc.access

* + 第四步：计算得到所有的功耗参数。当is\_tdp=true时，结果放入globalBPT->power中，否则放入globalBPT->rt\_power

globalBPT->power = globalBPT->power\_t + globalBPT->local\_result.power\*pppm\_lkg

解释：（其中，pppm\_lkg= {0,1,1,0}, cacti/const.h:262）

除了RAS和多线程相关之外，其它都一样

dynamic=power\_t.readop.dynamic

leakage=local\_result.power.readop.leakage

gate\_leakage=local\_result.power.readop.gate\_leakage

* + 当对象为RAS时：

RAS->power=RAS->power\_t + RAS->local\_result.power\*coredynp.pppm\_lkg\_multhread

(其中pppm\_lkg\_multhread={0, coredynp.num\_hthreads, coredynp.num\_hthreads, 0})

dynamic=power\_t.readop.dynamic

leakage=local\_result.power.readop.leakage\*num\_hthreads

gate\_leakage=local\_result.power.readop.gate\_leakage\*num\_hthreads

* + 将所有的对象的功耗参数放在一起：

power / rt\_power = power + globalBPT->power + L1\_localBPT->power + L2\_localBPT->power + chooser->power + RAS->power;

## displayEnergy函数

* 参数为is\_tdp，但是该参数只被设置为true，因此可以不考虑
* 显示的结果包括：

**Area =** globalBPT->area.get\_area()\*1e-6

**Peak Dynamic =** globalBPT->power.readOp.dynamic\*clockRate

**Subthreshold Leakage =** (long\_channel? globalBPT->power.readOp.longer\_channel\_leakage:globalBPT->power.readOp.leakage)

**Subthreshold Leakage with power gating =** (long\_channel? globalBPT->power.readOp.power\_gated\_with\_long\_channel\_leakage : globalBPT->power.readOp.power\_gated\_leakage)

**Gate Leakage =** globalBPT->power.readOp.gate\_leakage

**Runtime Dynamic =** globalBPT->rt\_power.readOp.dynamic/executionTime

其中：long\_channel = XML->sys.longer\_channel\_device

power\_gating = XML->sys.power\_gating

|  |
| --- |
|  |