

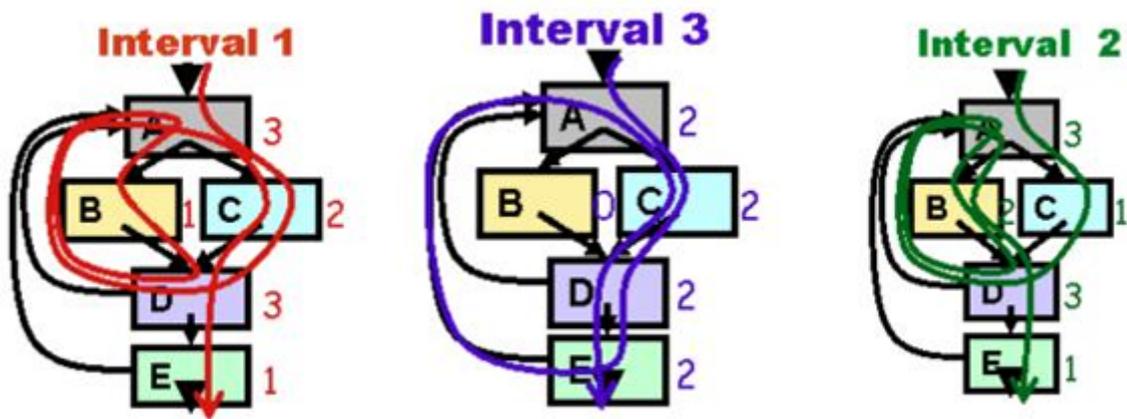
1 SimPoint基础知识

1.1 背景

SimPoint是基于离线聚类分析的一项技术，其动机是出于执行高效、准确的程序分析和架构仿真的需要

SimPoint基于这样一个观察：**程序在执行过程中往往会重复相似的行为模式。因此，可以通过识别这些重复的模式，选择代表性的程序片段进行详细模拟，而不是模拟整个程序执行过程。**

如果能统计出那几个主要的操作以及它们所占的比例，是不是只对那几个操作做仿真再加权就可以了呢？



1.2 基本块向量 (BBV)

基本块

基本块：从头到尾执行的一段代码，只有一个出口和一个入口。SimPoint使用基本块的执行频率作为指标，比较应用程序执行的不同部分。因为程序在给定时间的行为与它在该间隔 (interval) 内执行的代码直接相关，而基本块的分布为我们提供了这些信息。

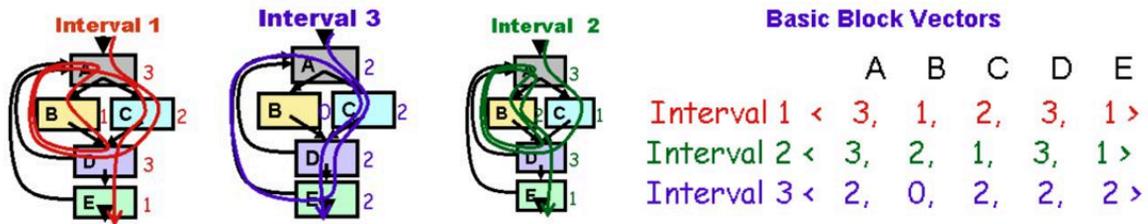
问题：基本块的类型是如何区分的？每个基本块都有一个唯一编号

程序在任意的时间间隔内将执行每个基本块一定次数，基于此可以得到每个执行间隔内的基本块向量，并告诉我们程序在哪个部位花费了时间。多个执行间隔会有多个基本块向量，通过对比基本块向量可以得到这些执行间隔之间的相似程度，如果两个基本块向量相似，那么这两个间隔在相同代码应该花费大致相同的时间，并且性能应该相似。

基本块向量

基本块向量 (BBV)：一维数组 (也可以视为多维向量)，其中程序的每个静态基本块都对应数组中的一个元素。以间隔 (interval) 为单位收集基本块向量。在每个间隔结束时，记录在本间隔内进入 (执行) 每个基本块的次数。**数组中的每个元素是指在执行间隔内进入 (执行) 相应的基本块的次数乘以该基本块的指令数。**通过乘以每个 basic block 中的指令数量，可以确保我们对 instructions 的权重相同，无论它们位于大 basic block 还是 small basic block 中。

BBV：元素个数等于基本块个数，每个元素位置对应相同的基本块。



我们关注的不是某个时间间隔内基本块的实际执行次数，而是**各个基本块之间执行时间的比例关系**。因此，对BBV进行**归一化处理**，具体方法是该向量中的每个元素除以该向量所有元素的总和。

一个interval对应多个基本块和一个BBV。BBV归一化处理后记录该interval中各个基本块执行时间比例（次数 * 指令数 / 总指令数）。

基本块差异对比

得到归一化后的BBV如何对比计算，输出结果表示BBV之间的相似程度？

欧几里得距离：将每个向量视为D维空间中的单个点，计算两点之间的距离

$$EuclideanDist(a, b) = \sqrt{\sum_{i=1}^D (a_i - b_i)^2}$$

曼哈顿距离：计算两个D维向量各个维度之间的差异。

$$ManhattanDist(a, b) = \sum_{i=1}^D |a_i - b_i|$$

归一化处理后，两个间隔对应的BBV曼哈顿距离介于0—2之间。

使用曼哈顿表示差异，曼哈顿在表示高维数据中的差异更准确。

在聚类算法使用欧几里得，在低纬度数据中更准确。

1.3 聚类

许多执行间隔彼此相似，一种有效的表示方法是将具有相似行为的间隔分组在一起。这个问题类似于聚类问题。

聚类的目标是将一组点分成组，使得每个组内的点彼此相似（根据某种度量，通常是距离）。

使用随机线性投影降维，然后是k-means聚类算法，为了选择k的值，使用贝叶斯信息准则（BIC）分数。

以下步骤是聚类算法步骤示例：

1. 分析每个程序执行的基本块，为每1亿条指令（interval）的执行生成基本块向量。
2. 使用随机线性投影将BBV数据的维度降至15维。
3. 对降维后的数据尝试k-means聚类算法，k值从1到10。k-means的每次运行都会产生一个聚类，**即将intervals分成k个不同的聚类。**
4. 对每个聚类（k = 1...10），使用BIC计算其拟合度。选择得分至少达到最佳得分90%的聚类中k最小的那个。

降维

对于聚类问题，必须解决维度问题。所有聚类算法都受到所谓的“维度诅咒”的影响，这指的是当维度数量增加时，聚类数据变得极其困难：

- 对于基本块向量，维度的数量是程序中执行的基本块的数量，在我们的实验数据中范围从2,756到102,038，对于非常大的程序，这个数字可能增长到数百万。
- 另一个实际问题是聚类算法的运行时间取决于数据的维度，如果维度增长太大，就会变得很慢。

由于BBV可能有很高的维度，SimPoint使用随机投影将其降到较低的维度（通常是15维），以减少计算复杂度。

k-means

SimPoint使用k-means算法对降维后的BBV进行聚类。每个聚类代表程序执行的一个阶段或行为模式。**输出是一组最终的聚类中心和每个点所属的聚类的映射。**由于我们已将数据投影到15维，我们可以快速为k-means生成聚类。

k-means算法还有个缺点就是需要提前确定k的值，k表示最后需要聚合成几个类，simpoint中用的方法是按某种规则（非随机，实验多次选区的都是同一组k值，可能也是有某种算法）选择多个k值（介于1—MAXk之间，选取的个数等于进行k-Means算法的次数），分别计算得到聚类的结果后使用 BIC算法进行打分，打分最高的k值作为最终的选择并作为最优的聚类数量

贝叶斯信息准则

SimPoint使用贝叶斯信息准则（BIC）来自动选择最佳的聚类数量（k值）。BIC分数越高，聚类被评分为对数据的“拟合”越好。

1.4 采样点选择

对于最后“拟合”最好的每个聚类，SimPoint选择一个代表性的采样点（通常是离聚类中心最近的点）。

只有这些采样点才会用于详细的架构模拟。每个采样点被分配一个权重，基于其所代表的聚类在整个程序执行中所占的比例。

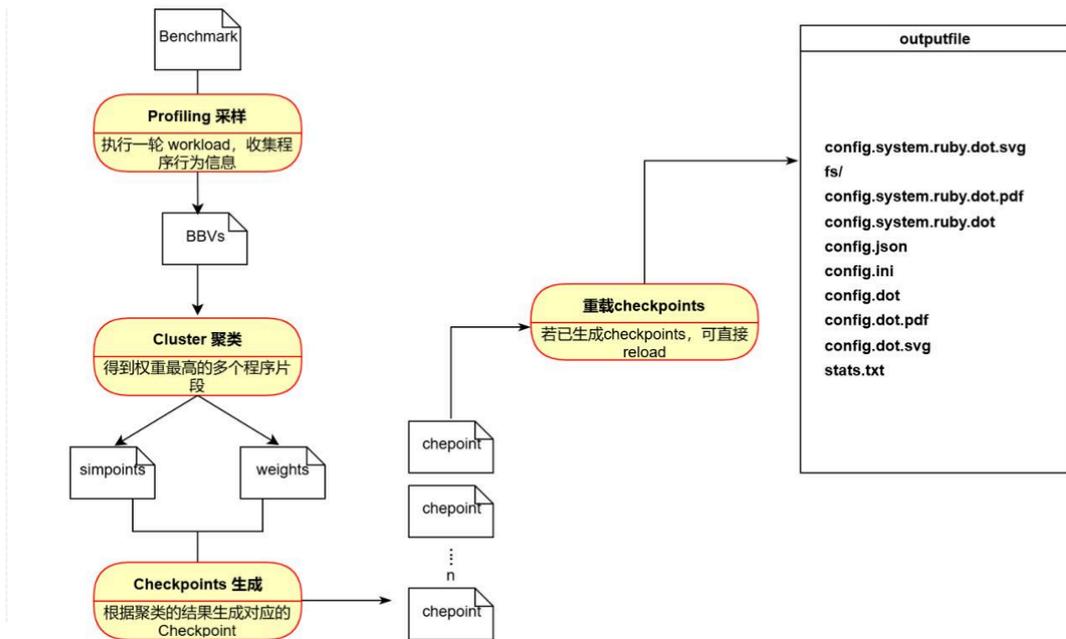
采样点：实际就是BBV，对应一个具体的interval

1.5 模拟

只对选定的采样点进行详细的架构模拟，而不是模拟整个程序。

使用采样点的模拟结果和它们的权重，重建整个程序执行的性能估计。

2 SimPoint在Gem5中的应用



2.1 采样

运行一遍应用程序，生成BBVs

- interval大小经验法则：一个常见的做法是选择一个使得程序总共有 1000-3000 个采样点的 interval 大小。
- 快速模拟，收集BBV信息，注意CPU type: `--cpu-type=NonCachingSimpleCPU`
- 输出: `simpoint.bb.gz`文件

```
1 # profiling
2 build/RISCV/gem5.opt \
3     -d path/to/output/dir \
4     configs/deprecated/example/se.py \
5     # 为了快速生成BBVs, 这里的CPU type使用NonCachingSimpleCPU
6     --cpu-type=NonCachingSimpleCPU \
7     -c path/to/benchmark.exe \
8     --simpoint-profile --simpoint-interval 10000
```

`simpoint.bb.gz`文件格式: `T[:BB_id:count]`

- 每个T对应一个interval, BB_id为basic block的编号;
- count为对应的basic block的值, 值的计算方式为在此interval中对应的block出现的次数和block中指令的个数的乘积。

```
cat /usr/bin/benchmarks/consensus-pro/output/simpoint/consensus-pro/radix2/profiling/mesho/radix2.ccat simpoint.bb.gz | head -n 100
7:11:1 2:6 3:12 4:14 5:12 6:11 7:7 8:16 9:20 10:6 11:12 12:5 13:5 14:2 15:5 16:9 17:7 18:1 19:1 20:3 21:1 22:1 23:2 24:2 25:2 26:2 27:2 28:2 29:2 30:6 31:4 32:14 33:3 34:1 35:3 36:1
37:2 38:10 39:5 40:4 41:15 42:1 43:10 44:7 45:2 46:8 47:6 48:9 49:4 50:2 51:13 52:24 53:2 54:16 55:8 56:1 57:2 58:3 59:3 60:9 61:6 62:3 63:21 64:17 65:77 66:16 67:8 68:20 69:4 70:28
71:5 72:16 73:8 74:36 75:56 76:47 77:20 78:1 79:6 80:3 81:1 82:2 83:9 84:18 85:18 86:2 87:8 88:2 89:6 90:2 91:4 92:5 93:3 94:1 95:27 96:18 97:45 98:9 99:28 100:12 101:2 102:8 103:8
104:88 105:14 106:10 107:10 108:4 109:6 110:16 111:10 112:14 113:6 114:4 115:11 116:9 117:2 118:2 119:3 120:2 121:2 122:7 123:10 124:2 125:2 126:13 127:13 128:10 129:15 130:2 131:2 132:4
133:13 134:12 135:10 136:5 137:6 138:11 139:168 140:14 141:28 142:14 143:13 144:4 145:13 146:6 147:22 148:4 149:10 150:4 151:6 152:15 153:17 154:3 155:11 156:12 157:77 158:21 159:35 160:7
161:24 162:119 163:34 164:32 165:30 166:16 167:8 168:8 169:6 170:6 171:9 172:56 173:77 174:2 175:5 176:3 177:7 178:4 179:16 180:56 181:1 182:19 183:25 184:20 185:60 186:5 187:2 188:3 189
7 190:1 191:6 192:3 193:5 194:14 195:2 196:3 197:4 198:13 199:9 200:504 201:3 202:6 203:5 204:5 205:5 206:5 207:5 208:5 209:5 210:5 211:5 212:5 213:5 214:5 215:5 216:5 217:4 218:42 219
24 220:20 221:12 222:2 223:4 224:6 225:4 226:2 227:2 228:3 229:150 230:50 231:16 232:32 233:10 234:6 235:18 236:32 237:160 238:11 239:20 240:180 241:102 242:168 243:50 244:20 245:12 246:6
247:34 248:10 249:15 250:1 251:1 252:1 253:9 254:3 255:4 256:6 257:4 258:4 259:6 260:3 261:3 262:2 263:2 264:6 265:3 266:1 267:2 268:9 269:3 270:2 271:2 272:4 273:7 274:4 275:4 276:4 7
277:20 278:4 279:4 280:160 281:2 282:6 283:9 284:18 285:54 286:8 287:12 288:78 289:190 290:30 291:4 292:16 293:13 294:2 295:16 296:12 297:24 298:18 299:15 300:35 301:144 302:56 303:16 304:64
305:12 306:18 307:10 308:42 309:3 310:8 311:5 312:15 313:13 314:5 315:8 316:17 317:4 318:15 319:2 320:10 321:3 322:21 323:3 324:4 325:42 326:6 327:9 328:5 329:13 330:7 331:12 332:2 33
3:5 334:2 335:5 336:2 337:5 338:3 339:3 340:3 341:3 342:2 343:3 344:2 345:2 346:3 347:3 348:6 349:9 350:12 351:5 352:8 353:4 354:5 355:1 356:2 357:4 358:2 359:14 360:3 361:3 362:1 363:1
364:4 365:1 366:14 367:4 368:1 369:7 370:3 371:9 372:3 373:45 374:48 375:2 376:11 377:50 378:66 379:22 380:259 381:26 382:13 383:18 384:9 385:18 386:9 387:18 388:8 389:7 390:5 391:10 39
2:1 393:3 394:6 395:3 396:6 397:3 398:4 399:4 400:20 401:100 402:10 403:7 404:1 405:9 406:6 407:12 408:1 409:4 410:2 411:7 412:8 413:2 414:2 415:2 416:2 417:7 418:6 419:4 420:4 421:11 4
22:1 423:13 424:8 425:3 426:1 427:4 428:18 429:8 430:2 431:7 432:8 433:3 434:6 435:1 436:2 437:3 438:15 439:5 440:3 441:7 442:3 443:2 444:2 445:2 446:11 447:7 448:4 449:9 450:2 451:3 45
2:4 453:2 454:4 455:10 456:2 457:3 458:10 459:1 460:2 461:2 462:3 463:2 464:4 465:3 466:3 467:13 468:2 469:2 470:27 471:5 472:6 473:1 474:11 475:21 476:11 477:19 478:2 479:11 480:3 481:27
482:6 483:6 484:6 485:16 486:27 487:16 488:9 489:16 490:9 491:3 492:6 493:9 494:16 495:3 496:9 497:6 498:6 499:30 500:57 501:30 502:4 503:24 504:7 505:2 506:4 507:18 508:5 509:10 510:6 511:1
24 512:6 513:21 514:6 515:3 516:36 517:3 518:16 519:6 520:6 521:24 522:4 523:24 524:16 525:4 526:4 527:2 528:2 529:2 530:2 531:2 532:1 533:9 534:6 535:12 536:12 537:30 538:30 539:13 540:98
541:56 542:21 543:35 544:35 545:20 546:27 547:18 548:45 549:9 550:27 551:7 552:14 553:28 554:70 555:6 556:6 557:6 558:6 559:7 560:7 561:17 562:14 563:8 564:3 565:5 566:5 567:5 568:1 569:3
570:1 571:2 572:4 573:10 574:1 575:6 576:2 577:32 578:10 579:10 580:4 581:2 582:4 583:18 584:2 585:9 586:1 587:3 588:6 589:2 590:1 591:1 592:5 593:1 594:6 595:7 596:1 597:4 598:21 599
28 600:1 601:27 602:1 603:27 604:1 605:27 606:1 607:27 608:1 609:28 610:6 611:2 612:26 613:22 614:6 615:76 616:16 617:8 618:16 619:60 620:6 621:6 622:14 623:4 624:3 625:31 626:22 627:6 7
628:6 629:6 630:14 631:5 632:5 633:3 634:1 635:24 636:2 637:28 638:3 639:3 640:8 641:2 642:6 643:10 644:22 645:3 646:7 647:7 648:2 649:3 650:2 651:3 652:3 653:17 654:15 655:5 656:2 657
18 658:10 659:53 660:117 661:5 662:94
T:194:2 218:7 219:4 220:4 511:8 657:18 662:1363 663:2 664:15 665:768 666:576 667:256 668:6144 669:256 670:256 671:256 672:6 673:9 674:15
T:221:2 229:15 230:5 231:2 232:4 233:2 239:2 240:18 241:6 242:7 243:5 244:2 248:16 249:6 256:2 257:4 301:16 302:7 303:2 304:8 305:2 306:3 307:2 308:7 318:3 512:2 513:7 636:11 637:14 642:3 643
5 644:11 658:19 659:53 660:117 661:51 662:1457 663:2 664:15 665:732 666:549 667:244 668:5856 669:244 670:244 671:244
T:194:2 218:7 219:4 220:4 221:2 229:15 230:5 231:2 232:4 233:2 239:2 240:18 241:6 242:7 243:5 244:2 248:16 249:6 256:2 257:4 301:16 302:7 303:2 304:8 305:2 306:3 307:2 308:7 318:3 511:8 512:2
513:7 636:11 637:14 642:3 643:5 644:11 657:18 658:10 659:53 660:117 661:51 662:1457 663:2 664:15 665:726 666:543 667:242 668:5808 669:244 670:244 671:240 672:6 673:9 676:12
T:194:2 218:7 219:4 220:4 221:2 229:15 230:5 231:2 232:4 233:2 239:2 240:18 241:6 242:7 243:5 244:2 248:16 249:6 256:2 257:4 301:16 302:7 303:2 304:8 305:2 306:3 307:2 308:7 318:3 511:8 512:2
513:7 636:11 637:14 642:3 643:5 644:11 657:18 658:10 659:53 660:117 661:51 662:1457 663:2 664:15 665:726 666:543 667:242 668:5808 669:244 670:244 671:240 672:6 673:9 676:12
T:665:123 666:93 667:40 668:984 669:40 670:40 671:44 672:16 673:9 677:8 678:1 679:4 680:1 681:630 682:42 683:207 684:2484 685:1449 686:294 687:84 688:126 689:62 690:93 691:164 692:164 693:861
694:164 695:164 696:123 697:265 698:243 699:164 700:246 701:164 702:246 703:40 704:168
T:681:720 682:48 683:242 684:2904 685:1694 686:336 687:96 688:144 689:68 690:102 691:196 692:196 693:1029 694:192 695:192 696:144 697:240 698:360 699:192 700:288 701:192 702:288 703:56 704:96
T:681:720 682:48 683:248 684:2880 685:1680 686:336 687:96 688:144 689:76 690:114 691:192 692:192 693:1008 694:196 695:196 696:147 697:245 698:333 699:192 700:288 701:192 702:288 703:40 704:144
```

2.2 聚类

生成最佳k个聚类。

- 使用Simpoint工具分析Gem5生成的BBV文件。
- 使用k-means算法何BIC算法确定最佳的聚类数量和采样点。
- maxK = 30是经验值
- 输出: xx.simpoints 与 xx.weights 两个文件

```
1 # cluster
2 simpoint.3.2/bin/simpoint \
3     -maxK 30 \
4     -numInitSeeds 1 \
5     -loadFVFile path/to/simpoint.bb.gz \
6     -inputVectorsGzipped \
7     -saveSimpoints path/to/save/radix2.simpoints \
8     -saveSimpointWeights path/to/save/radix2.weights
```

k-means算法还有个缺点就是需要提前确定k的值，k表示最后需要聚合成几个类，simpoint中用的方法是按某种规则（非随机，实验多次选区的都是同一组k值，可能也是有某种算法）选择多个k值（介于1—maxK之间，选取的个数等于进行k-Means算法的次数，默认是7次，即选7个k值），分别计算得到聚类的结果后使用 BIC算法进行打分，打分最高的k值作为最终的选择并作为最优的聚类数量

```

-----
Run number 6 of at most 7, k = 6
-----
Initialization seed trial #1 of 1; initialization seed = 493575231
-----
Initialized k-means centers using random sampling: 6 centers
Number of k-means iterations performed: 13
BIC score: 69256.6
Distortion: 1.3911
Distortions/cluster: 0.0251696 0.264212 0.0184463 0.59405 0.0179946 0.471232
Variances: 0.000997208
Variances/cluster: 5.92226e-05 0.0440353 5.18154e-05 0.0048643 0.000719785 0.000818112
The best initialization seed trial was #1
-----

Run number 7 of at most 7, k = 7
-----
Initialization seed trial #1 of 1; initialization seed = 493575232
-----
Initialized k-means centers using random sampling: 7 centers
Number of k-means iterations performed: 33
BIC score: 68998.7
Distortion: 1.37506
Distortions/cluster: 0.0168676 0.453661 0.0199836 0.00829146 0.0179946 0.59405 0.264212
Variances: 0.000986413
Variances/cluster: 5.00059e-05 0.00180024 3.58772e-05 3.85649e-05 0.000719785 0.0048643 0.0440353
The best initialization seed trial was #1
-----

Post-processing runs
-----
For the BIC threshold, the best clustering was run 4 (k = 8)
Post-processing run 4 (k = 8)
Saving simpoints of all non-empty clusters to file '/home/cwh/work/benchmarks/coremark-pro/output/simpoint/coremark-pro/radix/cluster/radix2.simpoints'
Saving weights of all non-empty clusters to file '/home/cwh/work/benchmarks/coremark-pro/output/simpoint/coremark-pro/radix/cluster/radix2.weights'

```

xx.simpoints文件内容:

每行包含两个数字，格式为 "采样点 聚类ID"，每个聚类都由一个特定的 interval 来代表，这个 interval 被认为最能代表该聚类的行为特征。性能模拟时，只需要模拟这些被选中的 intervals，每个被选中的 interval 代表了程序执行中的一个不同阶段或行为模式。

```

1 465 0
2 532 1
3 741 2
4 1398 3
5 1 4
6 475 5
7 1088 6
8 1372 7
9

```

- 例如，"465 0" 表示第465个采样点属于第0个聚类，也意味着第0个聚类的中心点是第465次采样

采样点 == BBV
 第465次采样: 第465个interval, 地址是465 * interval大小

- 这个文件列出了8个代表性的采样点，每个对应一个聚类

xx.weights 文件内容:

```

1 0.249108 0
2 0.388294 1
3 0.0963597 2
4 0.00499643 3
5 0.00499643 4
6 0.234832 5
7 0.0028551 6
8 0.0185582 7

```

- 每行包含两个数字，格式为 "权重 聚类ID"
- 例如，"0.0378378 0" 表示第0个聚类的权重为0.0378378

- 较高的权重表示该聚类代表了程序执行中更大的部分

2.3 生成CheckoutPoints

拿到simpoints和weights文件后就可以知道各段interval的起始位置和权重了。可以使用GEM5快速forward到对应的interval，然后切换到GEM5的detailCPU模式开始运行。

但为了在后续仿真时跳过forward过程，加速仿真流程，GEM5提供了checkpoint功能：第一次运行时，快速forward到interval的起始点 后使用checkpoint功能把快照保存起来，以后运行时直接从快照恢复速度快很多。

省去Gem5 Forward到对应interval的过程，所以这里使用的CPU type是：--cpu-type AtomicSimpleCPU

- 使用SimPoint生成的采样点信息。
- 设置Gem5以在指定的指令数后生成检查点。
- CPU type是：--cpu-type AtomicSimpleCPU

```
1 # checkpoints
2 build/RISCV/gem5.opt \
3     -d path/to/output/dir \
4     configs/deprecated/example/se.py \
5     --take-simpoint-checkpoint=path/to/radix2.simpoints,
6     path/to/radix2.weights,10000,1000 \
7     -n 16 \
8     # 注意这里的CPU type
9     --cpu-type AtomicSimpleCPU \
10    --l1d_size=64kB \
11    --l1i_size=64kB \
12    --num-l2caches=16 \
13    --l2_size=128kB \
14    --num-dirs=16 \
15    --ruby \
16    --network=garnet \
17    --topology=Mesh_XY \
18    --mesh-rows=4 \
19    -c path/to/benchmark.exe
```

checkpoint类似于上下文切换时保存的上下文，也类似于微架构中遇到分支指令时，保存的checkpoint 可以理解成是在一个时刻CPU运行的快照，会保存架构寄存器和缓存memory的状态。

输入为前面生成的 simpoints 和 weights 文件， 输出生成了与simpoints数目相同个checkpoint 文件 文件夹里面存储了给定时刻（执行到各个聚类代表interval位置时）硬件的全部信息，比如cache配置，cpu类型等。

```
✓ checkpoints
  ✓ cpt.simpoint_00_inst_9000_weight_0.004996_interval_10000_warmup_1000
    📄 m5.cpt
    📄 system.physmem.store0.pmem
    📄 system.ruby.cache.gz
  > cpt.simpoint_01_inst_4649000_weight_0.249108_interval_10000_warmup_1000
  > cpt.simpoint_02_inst_4749000_weight_0.234832_interval_10000_warmup_1000
  > cpt.simpoint_03_inst_5319000_weight_0.388294_interval_10000_warmup_1000
  > cpt.simpoint_04_inst_7409000_weight_0.096360_interval_10000_warmup_1000
  > cpt.simpoint_05_inst_10879000_weight_0.002855_interval_10000_warmup_1000
  > cpt.simpoint_06_inst_13719000_weight_0.018558_interval_10000_warmup_1000
  > cpt.simpoint_07_inst_13979000_weight_0.004996_interval_10000_warmup_1000
```

m5.cpt即为当前cpu运行状态的快照，比如寄存器值，页表映射关系之类的system.physmem.store0.pmem则是当时的内存的状态。

因此有了cpt和pmem，gem5就可以从任意一个checkpoint进行restore重载

2.4 重载

由于Checkpoint保存的快照，Gem5可以从任意一个 checkpoint进行restore重载，下图中的4表示从第4个CheckPoint恢复。

- 重载时是详细模拟采样点对应的intervals
- **重载时的架构参数除了CPU type，其它参数应该与生成checkpoint时相同，因为checkpoint保存的是上一步的状态**

所以说，如果更改了底层架构，需要重新生成一遍checkpoint

- 注意--restore-simpoint-checkpoint -r 4 对应CheckPoint的编号是 3 (checkpoint文件从0开始编号)

```
1 # reload
2 build/RISCV/gem5.opt \
3   -d path/to/output/dir \
4   ./configs/deprecated/example/se.py \
5   # 这里以下配置应该与上一步生成checkpoints的配置相同
6   -n 16 \
7   # 这里的CPU type使用详细模拟的type
8   --cpu-type TimingSimpleCPU \
9   --l1d_size=64kB \
10  --l1i_size=64kB \
11  --num-l2caches=16 \
12  --l2_size=128kB \
13  --num-dirs=16 \
14  --ruby \
15  --network=garnet \
16  --topology=Mesh_XY \
17  --mesh-rows=4 \
18  --restore-simpoint-checkpoint -r 4 \
```

```
19 --checkpoint-dir path/to/checkpoints \  
20 -c path/to/benchmark.exe
```

2.5 合并仿真结果

合并算法

使用 SimPoint 的最后一步是合并加权仿真点，以得出程序执行的总体性能估计。

- 收集每个SimPoint模拟的性能数据。
- 使用SimPoint权重文件 (.weights) 来加权平均这些结果，得到整个程序的性能估计。

每个权重代表属于其阶段的总执行的比例，总体性能估计值是一组模拟点估计值的加权平均值。

例如，如果有 3 个模拟点，它们的权重为 [0.22, 0.33, 0.45]，它们的 CPI 为 (CPI1, CPI2, CPI3) 则这些点的加权平均值为： $CPI = 0.22 * CPI1 + 0.33 * CPI2 + 0.45 * CPI3$

加权平均计算出的 CPI 是程序完整执行的估计值，此计算方法可用于任何指标

预热问题

处理器中的 Cache、MMU、分支预测器的冷启动会影响性能评估的准确性，因此需要进行Warm-Up，对 Cache、MMU、分支预测器进行数据预热。具体实现方式为提前多执行W(Warmuplength)条指令，例如：一个预期的 Checkpoint，时间节点为N，采样区间长度(cpt-interval的参数)为1，预热长度为 W。真正生成的 Checkpoint 节点为 N-W，处理器执行时，需要执行(N-W, N+I)，即 W+1条指令。收集性能数据时需要舍去(N-M,N)部分，只收集(N, N+I)部分的性能数据。

在Gem5重载检查点的m5out/stats.txt文件中会有两部分内容：

```
----- Begin Simulation Statistics -----  
...  
----- End Simulation Statistics -----
```

第一部分就是预热，预热数据不考虑，只分析第二部分数据。

3 问题

SimPoint在生成检查点 (BBVs) 时 (对应Gem5的采样阶段) 是完整的执行一遍程序吗?

SimPoint在生成采样点时并不需要完整执行一遍程序。

SimPoint的优势在于它能在不完整执行程序的情况下识别出代表性的采样点。这大大减少了分析时间，同时仍能捕获程序的关键行为特征。

1. BBV 分析阶段：
 - 这是第一次"执行"程序，但不是完整或详细的执行。
 - 使用功能模拟器或二进制插桩工具。
 - 目的是收集基本块执行频率信息。

- 这个过程比真正的执行快得多，因为它忽略了许多微架构细节（在Gem5中使用`--cpu-type=NonCachingSimpleCPU`）。

2. 聚类分析阶段：

- 完全离线，不涉及程序执行。
- 使用收集到的 BBV 数据进行数学分析。
- 识别相似的程序阶段并选择代表性采样点。

3. 检查点生成阶段：

- 这是第一次“真正”执行程序，但只执行到选定的采样点。
- 使用快速的功能模拟器（Gem5中使用：`--cpu-type AtomicSimpleCPU`）。
- 目的是在每个采样点创建系统状态的快照（检查点）。

4. 详细模拟阶段：

- 这是最终的详细执行阶段。
- 从每个检查点开始，使用详细的微架构模拟器。
- 只模拟每个采样点后的一小段指令（通常是一个完整的间隔）。

可以在社交网站上请求别人发SimPoint选好的点给他，使用别人的simpoints吗？

这肯定是不行的，因为两边的Binary很可能不一样，受到程序版本、编译器版本、优化选项、库的版本的影响。A这边的第N个片段和B的第N个片段很可能不是同一个片段。