|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 00:00:00,375 --> 00:00:01,818 | 大家好 |  |
| 2 | 00:00:02,275 --> 00:00:06,871 | 我是赛昉的CPU团队的负责人余红斌 |  |
| 3 | 00:00:07,037 --> 00:00:09,715 | 我这次来我希望能够 |  |
| 4 | 00:00:09,715 --> 00:00:11,562 | 从技术层面上来讲一讲 |  |
| 5 | 00:00:11,562 --> 00:00:14,320 | 我们高性能的天枢项目 |  |
| 6 | 00:00:14,360 --> 00:00:16,400 | 它从技术上是一个什么样的项目 |  |
| 7 | 00:00:16,840 --> 00:00:18,420 | 因为其实在半年之前 |  |
| 8 | 00:00:18,421 --> 00:00:21,600 | 在去年十二月份的【】的上 |  |
| 9 | 00:00:21,601 --> 00:00:24,255 | 我们其实已经对外宣布过 |  |
| 10 | 00:00:24,256 --> 00:00:26,855 | 说我们有这么一个高性能的CPU |  |
| 11 | 00:00:27,381 --> 00:00:30,681 | 我们当时也披露了一些 performance的性能 |  |
| 12 | 00:00:30,835 --> 00:00:32,815 | 一个非常粗的一个框图 |  |
| 13 | 00:00:33,025 --> 00:00:35,840 | 但是我想我是工程师 |  |
| 14 | 00:00:35,940 --> 00:00:39,400 | 所以我会从技术层面上来讲我们的项目 |  |
| 15 | 00:00:39,440 --> 00:00:40,400 | 是一个什么样子 |  |
| 16 | 00:00:40,734 --> 00:00:44,040 | 它里面的一些主要的技术上的决定是什么 |  |
| 17 | 00:00:44,900 --> 00:00:48,820 | 我们现在距离上一次已经有半年了 |  |
| 18 | 00:00:49,360 --> 00:00:50,840 | 我们现在的状态是什么 |  |
| 19 | 00:01:12,406 --> 00:01:17,180 | 在开始讲之前 |  |
| 20 | 00:01:17,181 --> 00:01:21,317 | 我想我们从一个非常粗的一个层面上 |  |
| 21 | 00:01:21,317 --> 00:01:25,320 | 来对我们的这个项目来做一个综述 |  |
| 22 | 00:01:25,681 --> 00:01:27,365 | 它大概是一个什么样的项目 |  |
| 23 | 00:01:28,975 --> 00:01:30,265 | 首先说明 |  |
| 24 | 00:01:30,265 --> 00:01:33,800 | 这上面Arm的这些A76 A77 A78 |  |
| 25 | 00:01:33,800 --> 00:01:36,525 | 所有的东西都是我从网上找的 |  |
| 26 | 00:01:37,000 --> 00:01:40,370 | 下面的这些比较是我个人的理解 |  |
| 27 | 00:01:40,370 --> 00:01:42,290 | 我想每个人 |  |
| 28 | 00:01:42,290 --> 00:01:45,009 | 可能会有不一样的理解 |  |
| 29 | 00:01:45,310 --> 00:01:47,350 | 所以我没有办法对Arm的数据负责 |  |
| 30 | 00:01:47,410 --> 00:01:48,030 | 但是 |  |
| 31 | 00:01:48,895 --> 00:01:51,015 | 我对我们天枢这个项目的每一项 |  |
| 32 | 00:01:51,455 --> 00:01:53,875 | 每一项数据我们都会负责 |  |
| 33 | 00:01:53,876 --> 00:01:54,965 | 我保证它是对的 |  |
| 34 | 00:01:55,212 --> 00:01:56,155 | 因为我是工程师 |  |
| 35 | 00:01:56,156 --> 00:01:59,035 | 所以我不会有任何marketing的一些 |  |
| 36 | 00:01:59,555 --> 00:02:00,595 | 这种成分在里面 |  |
| 37 | 00:02:01,315 --> 00:02:04,095 | 我们的项目在开始讲之前 |  |
| 38 | 00:02:04,096 --> 00:02:05,735 | 我说一下我们项目的定位 |  |
| 39 | 00:02:05,895 --> 00:02:07,535 | 我们的项目其实 |  |
| 40 | 00:02:08,660 --> 00:02:13,080 | 主要的定位实际上就是做一个通用的CPU |  |
| 41 | 00:02:13,740 --> 00:02:15,775 | 通用的CPU它要求的是什么 |  |
| 42 | 00:02:15,950 --> 00:02:20,440 | 我想前端除了取指令以外 |  |
| 43 | 00:02:20,441 --> 00:02:22,460 | 最重要的一个部分就是分支预测 |  |
| 44 | 00:02:22,853 --> 00:02:27,160 | 分支预测就我们主要的特点也写在这里 |  |
| 45 | 00:02:27,200 --> 00:02:29,400 | 一个是多级的BTB |  |
| 46 | 00:02:30,893 --> 00:02:32,740 | 那另一个就是 |  |
| 47 | 00:02:32,740 --> 00:02:36,800 | 多种分支预测算法的混合体 |  |
| 48 | 00:02:37,215 --> 00:02:40,287 | 典型的就是我这里列了三个TAGE |  |
| 49 | 00:02:41,496 --> 00:02:43,695 | 间接跳转和RAS |  |
| 50 | 00:02:44,195 --> 00:02:47,095 | 我想真正做CPU的应该不用展开说 |  |
| 51 | 00:02:47,096 --> 00:02:49,575 | 应该都知道我的含义是什么 |  |
| 52 | 00:02:52,212 --> 00:02:56,000 | 接下来就是我们整个流水线的中断 |  |
| 53 | 00:02:56,660 --> 00:03:01,120 | 这个中断其实是所谓的outta word的 |  |
| 54 | 00:03:01,420 --> 00:03:03,260 | 中间的控制这个模块 |  |
| 55 | 00:03:03,825 --> 00:03:06,135 | 一个outta word的一个库 |  |
| 56 | 00:03:06,136 --> 00:03:08,562 | 它其实真正做【】cycle的 |  |
| 57 | 00:03:08,562 --> 00:03:10,115 | 就是在这个地方 |  |
| 58 | 00:03:10,515 --> 00:03:14,135 | 那我们主要的第一个是解码 |  |
| 59 | 00:03:15,015 --> 00:03:19,575 | 那解码部分的话我们支持整型 |  |
| 60 | 00:03:19,576 --> 00:03:22,935 | 支持浮点 支持vector1.0 |  |
| 61 | 00:03:23,515 --> 00:03:27,715 | 那我们一个cycle最多可以解码5条指令 |  |
| 62 | 00:03:28,305 --> 00:03:29,385 | 就是解码部分 |  |
| 63 | 00:03:29,745 --> 00:03:34,365 | 那register rename和dispatch部分 |  |
| 64 | 00:03:34,525 --> 00:03:37,945 | 我们主要的一个设计上的一个决定就是 |  |
| 65 | 00:03:38,255 --> 00:03:41,975 | 整型 浮点 vector是独立的 |  |
| 66 | 00:03:42,525 --> 00:03:45,875 | 其实我这里没有把那个参数列出来 |  |
| 67 | 00:03:45,876 --> 00:03:47,975 | 就是我们默认的情况下 |  |
| 68 | 00:03:48,490 --> 00:03:55,255 | 我们的整型 vector 浮点的物理寄存器 |  |
| 69 | 00:03:55,295 --> 00:03:57,015 | 目前默认都是256 |  |
| 70 | 00:03:57,075 --> 00:04:00,832 | 这个其实也是旗舰CPU的 |  |
| 71 | 00:04:00,832 --> 00:04:02,915 | 一个典型的设计参数 |  |
| 72 | 00:04:03,325 --> 00:04:07,175 | 还有register rename和dispatch的带宽 |  |
| 73 | 00:04:07,176 --> 00:04:08,415 | 每个cycle 5条指令 |  |
| 74 | 00:04:09,150 --> 00:04:13,805 | 最后就是【】的数目 |  |
| 75 | 00:04:14,505 --> 00:04:17,685 | 也就是说你同时能够允许多少条指令 |  |
| 76 | 00:04:18,085 --> 00:04:20,805 | 在你的流水线中去执行 |  |
| 77 | 00:04:21,045 --> 00:04:24,565 | 我们目前的那个值是160 |  |
| 78 | 00:04:25,800 --> 00:04:27,920 | 当然实际上理论上 |  |
| 79 | 00:04:28,120 --> 00:04:30,720 | 我还可以根据我的【】去做【】 |  |
| 80 | 00:04:31,043 --> 00:04:35,340 | 但是这个其实已经是我们目前 |  |
| 81 | 00:04:36,445 --> 00:04:38,665 | 目前所看到的参数搭配下面 |  |
| 82 | 00:04:38,666 --> 00:04:40,185 | 其实它已经性能很好了 |  |
| 83 | 00:04:40,325 --> 00:04:41,825 | 我后面我会去讲我们现在 |  |
| 84 | 00:04:41,826 --> 00:04:43,005 | 达到的性能是什么 |  |
| 85 | 00:04:43,625 --> 00:04:44,665 | 最后就是commit |  |
| 86 | 00:04:45,345 --> 00:04:47,665 | 就是你整个【】执行完 |  |
| 87 | 00:04:47,665 --> 00:04:50,725 | 最后你要送到流水线中执行 |  |
| 88 | 00:04:50,726 --> 00:04:52,265 | 执行完了以后你要提交 |  |
| 89 | 00:04:52,409 --> 00:04:55,195 | submit好像是被翻译成提交的 |  |
| 90 | 00:04:55,195 --> 00:04:57,794 | 每个cycle可以提交10条指令 |  |
| 91 | 00:05:01,665 --> 00:05:04,250 | 最后就是执行单元 |  |
| 92 | 00:05:04,251 --> 00:05:05,543 | 执行单元 |  |
| 93 | 00:05:07,050 --> 00:05:10,950 | 那我们可以看到我们最多有10个执行单元 |  |
| 94 | 00:05:10,990 --> 00:05:13,980 | 所以理论上在你的那个前面 |  |
| 95 | 00:05:14,020 --> 00:05:15,120 | 你数据都准备好 |  |
| 96 | 00:05:15,121 --> 00:05:17,160 | 你没有底盘震碎的情况下 |  |
| 97 | 00:05:17,440 --> 00:05:18,680 | 你最多一个cycle |  |
| 98 | 00:05:19,620 --> 00:05:23,380 | 可以发射10条指令到你的后端执行单元去 |  |
| 99 | 00:05:24,075 --> 00:05:27,000 | 我们这里把这十个单元 |  |
| 100 | 00:05:27,000 --> 00:05:29,625 | 我这里又重新列出来一下 |  |
| 101 | 00:05:30,975 --> 00:05:32,850 | 现在页面上显示的这一点 |  |
| 102 | 00:05:32,850 --> 00:05:35,921 | 所以主要的这个流水线是就是这样 |  |
| 103 | 00:05:37,150 --> 00:05:40,290 | 所以我想这种流水线我也不打算去展开讲 |  |
| 104 | 00:05:40,291 --> 00:05:42,610 | 因为可能你最后你只要去展开讲 |  |
| 105 | 00:05:42,611 --> 00:05:44,862 | 你可能你就收不住了 |  |
| 106 | 00:05:44,862 --> 00:05:46,470 | 就会太多了 |  |
| 107 | 00:05:46,510 --> 00:05:49,170 | 但是整体我想给大家的一个印象就是 |  |
| 108 | 00:05:49,535 --> 00:05:53,075 | 我们现在从我的设计参数上看 |  |
| 109 | 00:05:53,915 --> 00:05:59,895 | 我是超过Arm三年前的旗舰水平 |  |
| 110 | 00:06:01,540 --> 00:06:04,480 | 也就是说实际上我的那个设计 |  |
| 111 | 00:06:04,600 --> 00:06:07,140 | 我今天已经达到的性能是超过176的 |  |
| 112 | 00:06:07,620 --> 00:06:10,900 | 但是我不知道现在今天Arm的水平怎么样 |  |
| 113 | 00:06:10,901 --> 00:06:11,880 | 因为我也没有见过 |  |
| 114 | 00:06:12,335 --> 00:06:15,375 | 但是我想就是首先我们跟除了跟别人比 |  |
| 115 | 00:06:15,376 --> 00:06:17,046 | 我们也要自己拿出一些 |  |
| 116 | 00:06:17,046 --> 00:06:19,175 | 真正有意义的数据来看 |  |
| 117 | 00:06:19,176 --> 00:06:20,295 | 所以我们接着往下看 |  |
| 118 | 00:06:21,631 --> 00:06:25,680 | 后端很重要的一个部分就是vector |  |
| 119 | 00:06:25,720 --> 00:06:28,540 | 其实我今天听下来有很多 |  |
| 120 | 00:06:28,700 --> 00:06:31,260 | 有很多其他公司的同事也都在说 |  |
| 121 | 00:06:31,340 --> 00:06:32,660 | 我们也都支持vector |  |
| 122 | 00:06:32,884 --> 00:06:35,090 | 所以我们这里也把我们 |  |
| 123 | 00:06:35,362 --> 00:06:36,850 | 这个天枢这个项目的vector |  |
| 124 | 00:06:36,851 --> 00:06:37,990 | 单独拿出来讲一下 |  |
| 125 | 00:06:38,590 --> 00:06:40,210 | 我们vector的第一个标准 |  |
| 126 | 00:06:40,210 --> 00:06:42,910 | 就是我们首先我是完全的 |  |
| 127 | 00:06:43,385 --> 00:06:47,045 | 支持最新版本的vector1.0 |  |
| 128 | 00:06:47,806 --> 00:06:48,965 | 这个实际上很重要 |  |
| 129 | 00:06:48,965 --> 00:06:53,645 | 就是RISC-V vector是一个很灵活的一个指令集 |  |
| 130 | 00:06:53,905 --> 00:06:54,805 | 所以 |  |
| 131 | 00:06:55,050 --> 00:06:57,305 | 这个灵活带来的一个副作用 |  |
| 132 | 00:06:57,306 --> 00:06:59,945 | 就是它每一个版本它的变化非常的大 |  |
| 133 | 00:07:00,125 --> 00:07:02,225 | 所以你如果你的项目做得太早 |  |
| 134 | 00:07:02,226 --> 00:07:05,185 | 你会发现你现在你之前做的 |  |
| 135 | 00:07:05,245 --> 00:07:07,045 | 比如0.7版本 0.8版本 |  |
| 136 | 00:07:07,105 --> 00:07:09,205 | 跟今天的1.0是完全不一样的 |  |
| 137 | 00:07:09,460 --> 00:07:11,340 | 所以你这个事情你也不能做得太早 |  |
| 138 | 00:07:11,660 --> 00:07:13,220 | 那我们这个时间点非常好 |  |
| 139 | 00:07:13,220 --> 00:07:16,060 | 就是我今天我做出来就是1.0 |  |
| 140 | 00:07:16,060 --> 00:07:17,240 | 而他1.0的版本 |  |
| 141 | 00:07:17,971 --> 00:07:20,065 | 那接下来两三个月他就会【】 |  |
| 142 | 00:07:20,850 --> 00:07:24,609 | 另一个就是我们主要的设计参数 |  |
| 143 | 00:07:24,609 --> 00:07:26,443 | 就是我们有两个vector pipe |  |
| 144 | 00:07:26,585 --> 00:07:29,405 | 也就是说【】我可以发射两个 |  |
| 145 | 00:07:29,900 --> 00:07:31,005 | vector指令下去 |  |
| 146 | 00:07:31,006 --> 00:07:35,225 | 然后vector要有独立的物理寄存器堆 |  |
| 147 | 00:07:36,205 --> 00:07:38,605 | 另外下面两项是很重要的 |  |
| 148 | 00:07:38,885 --> 00:07:42,829 | 就是我们的vector是可以【】的 |  |
| 149 | 00:07:43,825 --> 00:07:47,503 | 所以你看主要的VLEN基本上 |  |
| 150 | 00:07:47,928 --> 00:07:49,665 | 比如像Arm |  |
| 151 | 00:07:49,666 --> 00:07:53,465 | 他其实今天一直做到他最新的710 |  |
| 152 | 00:07:54,065 --> 00:07:56,089 | 他的VLEN |  |
| 153 | 00:07:56,089 --> 00:07:59,185 | 实际上他其实他也有一个对应的vector |  |
| 154 | 00:07:59,186 --> 00:08:00,805 | 他的VLEN是128 |  |
| 155 | 00:08:01,075 --> 00:08:04,450 | 其实我们可以configuration成256 512 |  |
| 156 | 00:08:04,845 --> 00:08:06,205 | 甚至一直到1024 |  |
| 157 | 00:08:06,625 --> 00:08:10,065 | DLEN我们可以同样可以configuration128 256 |  |
| 158 | 00:08:10,385 --> 00:08:11,337 | 所以这样就确保说 |  |
| 159 | 00:08:11,337 --> 00:08:12,890 | 如果我真的是把这个东西 |  |
| 160 | 00:08:12,890 --> 00:08:14,609 | 作为一个IP卖给客户的话 |  |
| 161 | 00:08:14,790 --> 00:08:16,350 | 它在vector这个参数上是 |  |
| 162 | 00:08:16,351 --> 00:08:17,730 | 有很多东西可以选择的 |  |
| 163 | 00:08:18,300 --> 00:08:22,203 | 下面我们有一个理论上的 |  |
| 164 | 00:08:22,203 --> 00:08:25,050 | 峰值性能的比较 计算 |  |
| 165 | 00:08:25,050 --> 00:08:26,350 | 大家可以看一下 |  |
| 166 | 00:08:27,090 --> 00:08:29,970 | 完全的根据我们的设计参数来给出来的 |  |
| 167 | 00:08:32,200 --> 00:08:38,235 | 这个table是我们试图在我们的项目里面 |  |
| 168 | 00:08:38,375 --> 00:08:40,895 | 去cover的vector应用 |  |
| 169 | 00:08:41,250 --> 00:08:44,275 | 我前面讲你可能一个非常小的核 |  |
| 170 | 00:08:44,415 --> 00:08:45,615 | 你也可以支持vector |  |
| 171 | 00:08:45,675 --> 00:08:48,075 | 但是你非常小的核支持的vector |  |
| 172 | 00:08:48,430 --> 00:08:53,425 | 它绝对不可能和正常的核一样 |  |
| 173 | 00:08:53,751 --> 00:08:55,811 | 你真正一个数据中心用的vector |  |
| 174 | 00:08:55,811 --> 00:08:57,430 | 和一个手机上用的vector |  |
| 175 | 00:08:57,530 --> 00:08:58,570 | 那肯定是不一样的 |  |
| 176 | 00:08:58,870 --> 00:09:00,982 | 所以我们红线框出来的部分 |  |
| 177 | 00:09:00,982 --> 00:09:06,285 | 是我们试图在我们天枢这个项目里面去cover的 |  |
| 178 | 00:09:06,845 --> 00:09:09,485 | 主要的vector的应用场景 |  |
| 179 | 00:09:14,250 --> 00:09:19,705 | 那另一部分就是所有CPU的 |  |
| 180 | 00:09:20,925 --> 00:09:23,865 | 最困难的在设计上和验证上 |  |
| 181 | 00:09:24,025 --> 00:09:26,465 | 我觉得不可否认的都是最困难的部分 |  |
| 182 | 00:09:26,466 --> 00:09:27,685 | 就是memory子系统 |  |
| 183 | 00:09:28,085 --> 00:09:31,065 | memory子系统它决定了你整个CPU |  |
| 184 | 00:09:31,305 --> 00:09:33,205 | 特别是高性能CPU的性能 |  |
| 185 | 00:09:33,725 --> 00:09:37,065 | 同时你能不能把memory做得好 |  |
| 186 | 00:09:37,105 --> 00:09:38,685 | 你能不能支持多核 |  |
| 187 | 00:09:39,005 --> 00:09:40,845 | 你能不能支持root cluster |  |
| 188 | 00:09:41,885 --> 00:09:44,785 | 你能不能支持在一个复杂系统里面 |  |
| 189 | 00:09:45,240 --> 00:09:47,992 | 支持32核 64核 128核 |  |
| 190 | 00:09:47,992 --> 00:09:49,180 | 它所有的这些事情 |  |
| 191 | 00:09:49,181 --> 00:09:50,165 | 实际上归根到底 |  |
| 192 | 00:09:50,468 --> 00:09:52,380 | 都是你memory子系统的设计 |  |
| 193 | 00:09:52,381 --> 00:09:53,200 | 或者换句话说 |  |
| 194 | 00:09:53,201 --> 00:09:54,280 | 你cache性 |  |
| 195 | 00:09:54,640 --> 00:09:56,150 | 在这种多核环境 |  |
| 196 | 00:09:56,150 --> 00:09:58,020 | 这种多核多cluster |  |
| 197 | 00:09:58,180 --> 00:10:00,000 | 高性能的情况下是怎么做的 |  |
| 198 | 00:10:00,280 --> 00:10:03,259 | 所以我们的主要的设计参数 |  |
| 199 | 00:10:03,259 --> 00:10:04,540 | 我这里也列出来了 |  |
| 200 | 00:10:04,540 --> 00:10:08,940 | 默认情况下64KB的DCache |  |
| 201 | 00:10:09,580 --> 00:10:10,800 | 两个load store pipe |  |
| 202 | 00:10:10,880 --> 00:10:12,380 | 四个cycle load-to-use |  |
| 203 | 00:10:13,620 --> 00:10:16,120 | 另外我想也提一下 |  |
| 204 | 00:10:16,320 --> 00:10:18,400 | 我们同时支持write-back和write-through |  |
| 205 | 00:10:18,820 --> 00:10:20,700 | 这个是有【】的可以config的 |  |
| 206 | 00:10:21,520 --> 00:10:23,540 | 那在不同的应用场景下的话 |  |
| 207 | 00:10:23,540 --> 00:10:26,000 | 有的人说我希望支持的 |  |
| 208 | 00:10:26,240 --> 00:10:28,225 | 我希望【】 我希望怎么样 |  |
| 209 | 00:10:28,225 --> 00:10:29,760 | 它实际上有很多参数可以调 |  |
| 210 | 00:10:30,250 --> 00:10:35,655 | 我们完全支持乱序的memory【】 |  |
| 211 | 00:10:36,268 --> 00:10:38,135 | 我们支持CMO |  |
| 212 | 00:10:38,136 --> 00:10:39,775 | CMO实际上 |  |
| 213 | 00:10:39,775 --> 00:10:41,715 | 是在整个RISC-V架构定义上 |  |
| 214 | 00:10:42,010 --> 00:10:44,030 | 我认为是很失败 |  |
| 215 | 00:10:44,031 --> 00:10:48,090 | 或者是至少是很滞后的一点 |  |
| 216 | 00:10:48,125 --> 00:10:49,110 | 直到今天 |  |
| 217 | 00:10:49,430 --> 00:10:50,470 | 它还没有一个 |  |
| 218 | 00:10:50,470 --> 00:10:54,510 | 真正符合RISC-V标准的一套CMO出来 |  |
| 219 | 00:10:54,855 --> 00:10:57,375 | 那我们支持非对齐的load store |  |
| 220 | 00:10:57,795 --> 00:10:59,035 | 所有的这些事情 |  |
| 221 | 00:10:59,095 --> 00:11:01,243 | 那另外我们支持【】 |  |
| 222 | 00:11:02,000 --> 00:11:06,875 | 这是L1 DCache的一些主要参数 |  |
| 223 | 00:11:06,876 --> 00:11:08,415 | L2 Cache开始的话 |  |
| 224 | 00:11:09,720 --> 00:11:11,020 | 我们是可以config的 |  |
| 225 | 00:11:11,020 --> 00:11:14,880 | 我们其实目前这里写512 |  |
| 226 | 00:11:14,881 --> 00:11:16,159 | 其实真正的系统里面 |  |
| 227 | 00:11:16,159 --> 00:11:17,540 | 其实一般来说是会 |  |
| 228 | 00:11:17,580 --> 00:11:18,780 | 远远超过这个值的 |  |
| 229 | 00:11:19,556 --> 00:11:21,835 | 在一些大系统里面 |  |
| 230 | 00:11:21,836 --> 00:11:24,155 | 它可能往往就是一兆 两兆 四兆 |  |
| 231 | 00:11:24,415 --> 00:11:28,675 | 甚至像Arm今天最新的参考设计 |  |
| 232 | 00:11:28,676 --> 00:11:29,715 | 他给的是八兆 |  |
| 233 | 00:11:30,750 --> 00:11:32,721 | 但是我想我的【】 |  |
| 234 | 00:11:32,721 --> 00:11:34,210 | 就是说我们其实这个 |  |
| 235 | 00:11:34,610 --> 00:11:36,550 | 大小这个东西在我们的项目 |  |
| 236 | 00:11:36,551 --> 00:11:37,710 | 里面是可以config的 |  |
| 237 | 00:11:38,030 --> 00:11:39,450 | 另外我们支持多cluster |  |
| 238 | 00:11:39,950 --> 00:11:42,230 | 多cluster的意思就是我可以在 |  |
| 239 | 00:11:42,721 --> 00:11:46,660 | 让多库 多个天枢 |  |
| 240 | 00:11:46,661 --> 00:11:50,660 | 或者是多个其他的支持开实质性的扩 |  |
| 241 | 00:11:51,120 --> 00:11:52,040 | 它能够共享【】 |  |
| 242 | 00:11:53,275 --> 00:11:55,470 | 我们支持ECC |  |
| 243 | 00:11:55,990 --> 00:11:58,870 | 最后我想需要我特意提出来一点 |  |
| 244 | 00:11:59,230 --> 00:12:00,930 | 我们支持多种总线协议 |  |
| 245 | 00:12:02,000 --> 00:12:07,910 | 其实今天RISC-V领域的总线接口有两种 |  |
| 246 | 00:12:08,290 --> 00:12:12,250 | 一种就是从伯克利来 |  |
| 247 | 00:12:12,830 --> 00:12:14,950 | 或者从【】过来的一些东西 |  |
| 248 | 00:12:15,035 --> 00:12:16,525 | 它有link |  |
| 249 | 00:12:16,975 --> 00:12:18,570 | 另一种就是可能 |  |
| 250 | 00:12:18,610 --> 00:12:20,670 | 以前这个公司是做Arm CPU的 |  |
| 251 | 00:12:21,010 --> 00:12:23,090 | 它天然就会支持arm的总线接口 |  |
| 252 | 00:12:23,670 --> 00:12:25,510 | 那所以永远有人会来问 |  |
| 253 | 00:12:26,190 --> 00:12:27,730 | 你出来支持这个TileLink |  |
| 254 | 00:12:27,731 --> 00:12:30,350 | 你能不能支持ACE 你能不能支持CHI |  |
| 255 | 00:12:30,765 --> 00:12:31,745 | 其实反过来也一样 |  |
| 256 | 00:12:31,785 --> 00:12:35,565 | 所以我们把所有的东西全部都做进去 |  |
| 257 | 00:12:41,915 --> 00:12:45,015 | 接下来就是我们的那个主要的应用场景 |  |
| 258 | 00:12:45,016 --> 00:12:46,415 | 这个是一个概述了 |  |
| 259 | 00:12:46,416 --> 00:12:48,915 | 就是大概我们希望它用在什么地方 |  |
| 260 | 00:12:49,395 --> 00:12:51,575 | 这个我想就是很粗的一个概念 |  |
| 261 | 00:12:51,576 --> 00:12:54,340 | 就是说我用的绝对是一些需要高性能 |  |
| 262 | 00:12:54,341 --> 00:12:55,660 | 需要复杂应用的场景 |  |
| 263 | 00:12:57,925 --> 00:13:01,225 | 下面是它典型的一些用法 |  |
| 264 | 00:13:01,685 --> 00:13:03,625 | 一种用法就是大小核的设计 |  |
| 265 | 00:13:03,626 --> 00:13:05,209 | 大小核的设计 |  |
| 266 | 00:13:05,209 --> 00:13:08,004 | 天枢它的定位是大核 |  |
| 267 | 00:13:08,540 --> 00:13:09,920 | 除了天枢之外 |  |
| 268 | 00:13:09,960 --> 00:13:12,575 | 我们赛昉还会有其他的CPU项目 |  |
| 269 | 00:13:12,843 --> 00:13:14,178 | 他们天然的 |  |
| 270 | 00:13:14,178 --> 00:13:15,620 | 我们自己做的项目 |  |
| 271 | 00:13:15,960 --> 00:13:18,200 | 它是能够在一个系统里面共存的 |  |
| 272 | 00:13:18,200 --> 00:13:20,678 | 也就是说我们会有自己的大小 |  |
| 273 | 00:13:20,734 --> 00:13:21,875 | 和这种config |  |
| 274 | 00:13:22,255 --> 00:13:24,175 | 我们会有自己的开启执行总线 |  |
| 275 | 00:13:26,685 --> 00:13:30,065 | 另一种就是更大规模的应用 |  |
| 276 | 00:13:31,200 --> 00:13:35,121 | 比如说像一些HPC的应用 |  |
| 277 | 00:13:35,121 --> 00:13:36,505 | 或者数据中心的应用 |  |
| 278 | 00:13:36,865 --> 00:13:39,805 | 它需要16核 32核 64核 |  |
| 279 | 00:13:40,245 --> 00:13:43,065 | 所以它会需要一些更复杂的一些总线架构 |  |
| 280 | 00:13:43,065 --> 00:13:45,700 | 比如【】比如【】 |  |
| 281 | 00:13:46,105 --> 00:13:48,325 | 那在这种总线架构的情况下 |  |
| 282 | 00:13:48,325 --> 00:13:51,344 | 只要你支持的总线接口是TileLink |  |
| 283 | 00:13:51,705 --> 00:13:54,025 | ACE或者CHI的其中一种 |  |
| 284 | 00:13:54,085 --> 00:13:55,805 | 我们就可以把天枢 |  |
| 285 | 00:13:55,805 --> 00:13:58,285 | 这个盒放到放到上面去 |  |
| 286 | 00:13:58,515 --> 00:14:01,035 | 所以我们天然就是支持这种大规模的 |  |
| 287 | 00:14:01,395 --> 00:14:02,815 | 这种数据中心应用的 |  |
| 288 | 00:14:04,505 --> 00:14:08,545 | 最后我想前面我说了很多 |  |
| 289 | 00:14:08,585 --> 00:14:11,796 | 那我的point实际上 |  |
| 290 | 00:14:11,796 --> 00:14:16,850 | 我们想做的是高性能的CPU |  |
| 291 | 00:14:16,850 --> 00:14:19,175 | 它的特点 第一个是高性能 |  |
| 292 | 00:14:19,450 --> 00:14:21,150 | 第二个它的功能非常全 |  |
| 293 | 00:14:21,190 --> 00:14:24,750 | 我前面讲我们做了很多很多的指令扩展 |  |
| 294 | 00:14:25,010 --> 00:14:28,650 | 所以作为我今天这个presentation的结束 |  |
| 295 | 00:14:28,925 --> 00:14:30,805 | 我会用一个demo来证明这一点 |  |
| 296 | 00:14:31,425 --> 00:14:33,712 | 所以我想我们最后这个 |  |
| 297 | 00:14:33,712 --> 00:14:35,525 | 是一个KVM的booting |  |
| 298 | 00:14:35,585 --> 00:14:36,765 | 我们把它录下来 |  |
| 299 | 00:14:37,225 --> 00:14:39,205 | KVM实际上如果大家了解一点 |  |
| 300 | 00:14:39,245 --> 00:14:43,905 | 实际上它是一个构建在linux之上的 |  |
| 301 | 00:14:45,015 --> 00:14:46,355 | 一个应用 |  |
| 302 | 00:14:51,625 --> 00:14:53,852 | 所以它首先会请【】kernel |  |
| 303 | 00:14:54,215 --> 00:14:56,255 | 然后会在【】之上 |  |
| 304 | 00:14:56,675 --> 00:14:58,935 | 去【】多个虚拟机 |  |
| 305 | 00:14:58,995 --> 00:15:00,387 | 然后在虚拟机之上 |  |
| 306 | 00:15:00,984 --> 00:15:03,075 | 再去跑应用程序 |  |
| 307 | 00:15:03,115 --> 00:15:04,696 | 所以我们能做到这一点 |  |
| 308 | 00:15:04,696 --> 00:15:05,955 | 就意味着实际上我们 |  |
| 309 | 00:15:05,995 --> 00:15:08,937 | 已经把所有的环节 |  |
| 310 | 00:15:08,937 --> 00:15:10,395 | 全部都已经做好了 |  |
| 311 | 00:15:10,435 --> 00:15:12,900 | 而且它今天已经可以完全工作了 |  |
| 312 | 00:15:56,100 --> 00:15:58,546 | 最后总结一下 |  |
| 313 | 00:15:58,546 --> 00:15:59,835 | 这个demo说明了什么 |  |
| 314 | 00:15:59,915 --> 00:16:02,235 | 这个demo其实说明了三件事情 |  |
| 315 | 00:16:02,915 --> 00:16:06,995 | 第一就是RISC-V架构定的没问题 |  |
| 316 | 00:16:07,490 --> 00:16:10,990 | 所以像kvm这样复杂的系统软件 |  |
| 317 | 00:16:11,030 --> 00:16:12,710 | 现在已经能够跑起来 |  |
| 318 | 00:16:13,325 --> 00:16:16,994 | 第二个实际上大家所有人都会问的 |  |
| 319 | 00:16:16,994 --> 00:16:18,790 | 这个软件生态没有问题 |  |
| 320 | 00:16:19,225 --> 00:16:23,237 | 因为实际上在我们的CPU项目过程中 |  |
| 321 | 00:16:23,237 --> 00:16:25,305 | 去【】kvm的时候 |  |
| 322 | 00:16:25,405 --> 00:16:27,285 | 虽然我们也发现了几个bug |  |
| 323 | 00:16:27,405 --> 00:16:28,525 | 几个软件的bug |  |
| 324 | 00:16:29,010 --> 00:16:31,030 | 但是实际上整个过程很顺利 |  |
| 325 | 00:16:31,090 --> 00:16:32,590 | 没有碰到太多的困难 |  |
| 326 | 00:16:33,170 --> 00:16:34,150 | 第三个 |  |
| 327 | 00:16:34,850 --> 00:16:38,270 | 实际上说明我们赛昉今天的高性能CPU |  |
| 328 | 00:16:38,271 --> 00:16:41,480 | 已经为将来的这种高性能的 |  |
| 329 | 00:16:41,480 --> 00:16:43,270 | 复杂应用做好了 |  |
| 330 | 00:16:43,700 --> 00:16:45,760 | 硬件或者芯片上的准备 |  |
| 331 | 00:16:45,761 --> 00:16:46,700 | 好 谢谢 |  |