

乐乐(00:09:37): 然后你也是头皮囊? 他都没在后头。他都没在后头把声音关掉。

刘晓义(00:18:00): 今天我们再次重新定义七点钟。正式大概7点开始, 大家可以拿一点饮料, 零食。Also 今天的 youtube 直播修好。所以。

刘晓义(00:19:56): 是的, 我们的活动会有录制会上, 通常来说网站上会 eventually 更新录播。youtube 因为我们过去一年时间里面活动都有 youtube, 所以直接看 youtube 的回放就可以。

刘晓义(00:21:12): 可以听到我说话吗?

zenkexi(00:21:18): 腾讯会议这边是可以的。

...(00:29:26): 喂我肯定对的, 这个是挂胸口的那种, 用这个, 然后我帮你把摄像头打开。还要开摄像头吗? 没有是他这个摄像头可牛逼了, 他这个摄像头是不是摄像头不好意思, 这个让大家看到了我的脸。吓到了没晚饭吃。因为好像好像不太对劲。可以调互动视角, 不是远程互动, 全景近景反出全景。挺好的就出来了。Ok.

...(00:30:27): 现在可以开始了吗? 我可以先介绍一下你或者你自我介绍一下。待会 PPT 里面会有个自我介绍没有那就。可以开始了, 可以开始! 大家好, 我叫龙英, 然后今天跟大家介绍这个话题是在 LV M 中支持一个。新的向量指令集。在所有事情开始之前要感谢一下 tuna, 感谢喵喵给大家这个机会来讲这个 talk?

...(00:31:06): 介绍一些背景。我在一年前为某个信创 R 的 SIMD 扩展编写了自动向量化编译器。它其实是分为 LV M 跟 GCC 两个不同的项目, 我们就接到了 LV M 的项目, 因为老师比较嫌弃 GCC, 反正就没有接。然后指标有性能指标跟功能指标, 我主要负责就是所有的代码编写工作。比如说所有的 CPP 所有的 TD 包括怎么改之类的, 然后本次 talk 的地址? 可以在这个地方看见 ok. 背影介绍完之后, 我们就直接进入正体。大概要讲这么一些东西, 首先第一个就是什么程序, 它搞向量化是有收益的? 第二个就是我们要做向量化, 需要做什么工作。第三个是介绍一个基本概念。用五分钟的时间入门 LV M。就可以开始一些练习题了。第四个是从标量 AR 到向量 AR 我们说如何去指导一些已有的一些 pass 去完成这个工作。第五个就是去把向量 AR 变成真正的机器指令, 主要是介绍这个指令选择算法。那第五点会是我们今天晚上讲的比较多的一个部分, 然后会涉及到一些骚操作跟踩过坑。第六个就是一些吹牛 future work, 大家都是说着玩的。

...(00:32:48): 然后第七个是小杰, 首先来第一个就是什么样的程序做向量化有收益, 大家可能脑子里最开始想到的是 M copy 之类的。看起来就是很大 loop 的程序。但是这个地方给出了毕生编译器必声这个词在上面很难打出来, 所以我们一般就用拼音代替在 o3 下的 spec2017 这个整体提升情况, 大家可以看到。必胜编译器, 绝大多数都是富有化。只有叉264有75.12%的提升, 最后在75.12%的情况下, 虽然别的东西都是负优化或者没有优化, 但几何平均也能优化个6%真是奇迹。

...(00:33:37): 下面这个是 o3 的浮点提升情况, 浮点就不太一样, 浮点基本上每一个课题都会有一点点提升, 这个提升明显是比 int 多很多的。这个例子可以看出来像叉264就是一些涉及到是什么编码之类的操作, 它其实看起来不是一个很大的 loop, 它就是一些很短的, 比如说长度为16的长度为八的那一些小 vector 拼在一起。大家可能本来会觉得我写很多的循环或者写很多的 m copy 会对这个 SIMD 会有所提升, 但提升比较多的, 反而是这种涉及到视频编码的一些一些。写过 close。

...(00:34:31): 总的来说, 做自动向量化大概要做很多系统工具上的工作在编译器方面就是去实现, 然后需要不同编译器之间需要约定 calling convention, 比如说怎么传参, 比如说返回值要放在哪谁会走站。谁会走计算器。除此之外, 在 JIT, 比如说 v8跟 JVM 是需要专门去做对应的修改的, 当然他们不修改也行, 就慢了一点, 这个 GC?

...(00:35:07): 它的这个 DLTS 这个地方熟悉 GDG 的朋友们都知道, 这是一个解析动态符号的地方, 这个地方它涉及到一件事, 就是它需要把所有寄存器都去保存一遍, 就是保存到站上, 然后再给它返回回来。如果你拿到一个不含向量计算器扩展的指令集的话, 它的 GDP C 很有可能是没有做这件事情的, 然后你的自动现代化程序就会在动态链接时候坏掉, 因为向量计算器只要一解析动态符号, 然后他就整个都记了下来计算器就会出现一些随机性的东西。就是因为在这个 GDP C 里面动态解析符号的时候, 我们需要去存一下。计算器到站上所有寄存器, 然后再给 reload 回来就是 GDP C 涉及到的操作。

...(00:36:19): 然后还有一个很重要的事情, 就是在 kernel 里面需要做什么, 比较重要的事情就是 alignment 有的处理器是不能接受非对齐缓存的, 包括前几天我们另一个组的同学跟我们吐槽香山处理器的 RVV 好像是不支持非对齐缓存的。要从我们这里抄解决方案, 还有一件事就是这个浮点, 它会存在非规格话术, 然后这个玩意会在运算的时候有可能会 trap 到 kernel 上去, 然后它在 SIMD 里面可能是不一

样的，那这就需要一些约定和一些 work 当然这些东西在我们做一个横向的时候。都是不知道的。需要去自己去，然后去猜哪里出了问题？

...(00:37:12): 当然，今天的工作主要是集中在第一个编译器的方面。编译器内部，主要就是做如何从向量标量的 AR 到向量 AR。做一个转换，就我们本来写的程序，它是标量的，但是我们现在需要把它表译成向量的，那这个转换怎么做到，那第二件事是我们假设已经生成了带向量的 IR。我们要怎么去生成 machine code 生成的方法是什么？用什么算法，我们今天会介绍这些细节。

...(00:37:48): 第一点，这个从标量到向量 AR，它主要是 LV M 中会有已有的 pass 来解决这个问题。但是这个 pass 它不是乱来，它是由后端去给他一些信息的。那么第二点，这个向量 AR 怎么断 motion code 就是用这个 selection dag 这个算法去做的之后我们也会介绍这个算法特点是它是 BB 的，它不能跨 BB，它是 BB 内部的。

...(00:38:19): 第二个特点是它是有一堆 pattern 我们去匹配到这些 pattern 之后，再把它重写为另一些东西，然后一直不断重写，直到这个 dag 和我们的 target 的非常像那么。就可以生成 code。这个 overview 就展示了一下刚刚那个过程，那首先，我们从标量的 LV MIR 它过两个 pass。过两个 pass 的时候，会给一些类似于 cos model 给他某个指令是什么代价向量寄存器有多宽之类的信息，方便去他做这种转换。转换完事之后，生成的是 vector 的 LV MI。然后它会过一个指定选择，这指定选择之后也会讲，然后再过一系列的复杂的操作就生成了 assembly object。当然，这些复杂的操作不是特别重要，因为主要的。主要的工作量都在 ICL 部分后面的东西基本上是你写这个 table 之后，它就会自己帮你生成出来。

...(00:39:32): 这个是一些基本概念。那刚刚提到 AR 有没有同学不知道 AR 是什么？举手我感受一下，没有听说过 AR 的。这个 module 就是从一点 C 点 CPP 还有我画画删掉，那大家能看清楚。可以从一个 create 生成。然后还有就是 function 这个 function，它是对应每个程序里面的函数。BB 是函数里面不带控制流的一个指令序列。整个一个 module，它是由很多 function 组成的那一个 function，它就是由很多 BB 组成的。大家大概明白这个逻辑关系吧！

...(00:40:30): IR 就是中间表示。就是我们在源代码和生成汇编代码中间的一种东西。不是红外光谱，那么这个向量数据要怎么表示就是 IR 到 IR 中的一个一个转换过程，我们要怎么表示它就是向量？

...(00:40:56): 一个直观的想法就是假设基本数据类型是 int 跟 float，那我们怎么去表示向量的数据类型，直观想法就是给他们前面加一个数。比如说八成 float 跟四成。那么问题来了，八成和四成可以支持，那我要是程序是 N 成怎么办？这个 N 可能是一个变量，可能是程序中传进来的，有可能是每个 CPU 特有的常量。大家有大家可以猜一猜 N 城是怎么样的？是支持吗。拆成若干个八成？但是答案是这样的，答案是 LV M 只能部分的支持这个 N 乘某个玩意。就它可以支持一个 CPU，它这个 N 是一个常量的情况，就我运行时可以知道这个 N 是一个 CPU 的常量，但是它不能去处理每个循环，它这个 N 都会变的这件事情。不是很支持 RVV。

...(00:42:17): 这个模型其实对 x86 或者你用这种指令集是非常有效的。因为 x86 这个模型就是我们知道它的指令集扩展大概有 sse sse2 sse3 sse42 sse43 sse44 sse45 sse46 sse47 sse48 sse49 sse50 sse51 sse52 sse53 sse54 sse55 sse56 sse57 sse58 sse59 sse60 sse61 sse62 sse63 sse64 sse65 sse66 sse67 sse68 sse69 sse70 sse71 sse72 sse73 sse74 sse75 sse76 sse77 sse78 sse79 sse80 sse81 sse82 sse83 sse84 sse85 sse86 sse87 sse88 sse89 sse90 sse91 sse92 sse93 sse94 sse95 sse96 sse97 sse98 sse99 sse100 sse101 sse102 sse103 sse104 sse105 sse106 sse107 sse108 sse109 sse110 sse111 sse112 sse113 sse114 sse115 sse116 sse117 sse118 sse119 sse120 sse121 sse122 sse123 sse124 sse125 sse126 sse127 sse128 sse129 sse130 sse131 sse132 sse133 sse134 sse135 sse136 sse137 sse138 sse139 sse140 sse141 sse142 sse143 sse144 sse145 sse146 sse147 sse148 sse149 sse150 sse151 sse152 sse153 sse154 sse155 sse156 sse157 sse158 sse159 sse160 sse161 sse162 sse163 sse164 sse165 sse166 sse167 sse168 sse169 sse170 sse171 sse172 sse173 sse174 sse175 sse176 sse177 sse178 sse179 sse180 sse181 sse182 sse183 sse184 sse185 sse186 sse187 sse188 sse189 sse190 sse191 sse192 sse193 sse194 sse195 sse196 sse197 sse198 sse199 sse200 sse201 sse202 sse203 sse204 sse205 sse206 sse207 sse208 sse209 sse210 sse211 sse212 sse213 sse214 sse215 sse216 sse217 sse218 sse219 sse220 sse221 sse222 sse223 sse224 sse225 sse226 sse227 sse228 sse229 sse230 sse231 sse232 sse233 sse234 sse235 sse236 sse237 sse238 sse239 sse240 sse241 sse242 sse243 sse244 sse245 sse246 sse247 sse248 sse249 sse250 sse251 sse252 sse253 sse254 sse255 sse256 sse257 sse258 sse259 sse260 sse261 sse262 sse263 sse264 sse265 sse266 sse267 sse268 sse269 sse270 sse271 sse272 sse273 sse274 sse275 sse276 sse277 sse278 sse279 sse280 sse281 sse282 sse283 sse284 sse285 sse286 sse287 sse288 sse289 sse290 sse291 sse292 sse293 sse294 sse295 sse296 sse297 sse298 sse299 sse300 sse301 sse302 sse303 sse304 sse305 sse306 sse307 sse308 sse309 sse310 sse311 sse312 sse313 sse314 sse315 sse316 sse317 sse318 sse319 sse320 sse321 sse322 sse323 sse324 sse325 sse326 sse327 sse328 sse329 sse330 sse331 sse332 sse333 sse334 sse335 sse336 sse337 sse338 sse339 sse340 sse341 sse342 sse343 sse344 sse345 sse346 sse347 sse348 sse349 sse350 sse351 sse352 sse353 sse354 sse355 sse356 sse357 sse358 sse359 sse360 sse361 sse362 sse363 sse364 sse365 sse366 sse367 sse368 sse369 sse370 sse371 sse372 sse373 sse374 sse375 sse376 sse377 sse378 sse379 sse380 sse381 sse382 sse383 sse384 sse385 sse386 sse387 sse388 sse389 sse390 sse391 sse392 sse393 sse394 sse395 sse396 sse397 sse398 sse399 sse400 sse401 sse402 sse403 sse404 sse405 sse406 sse407 sse408 sse409 sse410 sse411 sse412 sse413 sse414 sse415 sse416 sse417 sse418 sse419 sse420 sse421 sse422 sse423 sse424 sse425 sse426 sse427 sse428 sse429 sse430 sse431 sse432 sse433 sse434 sse435 sse436 sse437 sse438 sse439 sse440 sse441 sse442 sse443 sse444 sse445 sse446 sse447 sse448 sse449 sse450 sse451 sse452 sse453 sse454 sse455 sse456 sse457 sse458 sse459 sse460 sse461 sse462 sse463 sse464 sse465 sse466 sse467 sse468 sse469 sse470 sse471 sse472 sse473 sse474 sse475 sse476 sse477 sse478 sse479 sse480 sse481 sse482 sse483 sse484 sse485 sse486 sse487 sse488 sse489 sse490 sse491 sse492 sse493 sse494 sse495 sse496 sse497 sse498 sse499 sse500 sse501 sse502 sse503 sse504 sse505 sse506 sse507 sse508 sse509 sse510 sse511 sse512 sse513 sse514 sse515 sse516 sse517 sse518 sse519 sse520 sse521 sse522 sse523 sse524 sse525 sse526 sse527 sse528 sse529 sse530 sse531 sse532 sse533 sse534 sse535 sse536 sse537 sse538 sse539 sse540 sse541 sse542 sse543 sse544 sse545 sse546 sse547 sse548 sse549 sse550 sse551 sse552 sse553 sse554 sse555 sse556 sse557 sse558 sse559 sse560 sse561 sse562 sse563 sse564 sse565 sse566 sse567 sse568 sse569 sse570 sse571 sse572 sse573 sse574 sse575 sse576 sse577 sse578 sse579 sse580 sse581 sse582 sse583 sse584 sse585 sse586 sse587 sse588 sse589 sse590 sse591 sse592 sse593 sse594 sse595 sse596 sse597 sse598 sse599 sse600 sse601 sse602 sse603 sse604 sse605 sse606 sse607 sse608 sse609 sse610 sse611 sse612 sse613 sse614 sse615 sse616 sse617 sse618 sse619 sse620 sse621 sse622 sse623 sse624 sse625 sse626 sse627 sse628 sse629 sse630 sse631 sse632 sse633 sse634 sse635 sse636 sse637 sse638 sse639 sse640 sse641 sse642 sse643 sse644 sse645 sse646 sse647 sse648 sse649 sse650 sse651 sse652 sse653 sse654 sse655 sse656 sse657 sse658 sse659 sse660 sse661 sse662 sse663 sse664 sse665 sse666 sse667 sse668 sse669 sse670 sse671 sse672 sse673 sse674 sse675 sse676 sse677 sse678 sse679 sse680 sse681 sse682 sse683 sse684 sse685 sse686 sse687 sse688 sse689 sse690 sse691 sse692 sse693 sse694 sse695 sse696 sse697 sse698 sse699 sse700 sse701 sse702 sse703 sse704 sse705 sse706 sse707 sse708 sse709 sse710 sse711 sse712 sse713 sse714 sse715 sse716 sse717 sse718 sse719 sse720 sse721 sse722 sse723 sse724 sse725 sse726 sse727 sse728 sse729 sse730 sse731 sse732 sse733 sse734 sse735 sse736 sse737 sse738 sse739 sse740 sse741 sse742 sse743 sse744 sse745 sse746 sse747 sse748 sse749 sse750 sse751 sse752 sse753 sse754 sse755 sse756 sse757 sse758 sse759 sse760 sse761 sse762 sse763 sse764 sse765 sse766 sse767 sse768 sse769 sse770 sse771 sse772 sse773 sse774 sse775 sse776 sse777 sse778 sse779 sse780 sse781 sse782 sse783 sse784 sse785 sse786 sse787 sse788 sse789 sse790 sse791 sse792 sse793 sse794 sse795 sse796 sse797 sse798 sse799 sse800 sse801 sse802 sse803 sse804 sse805 sse806 sse807 sse808 sse809 sse810 sse811 sse812 sse813 sse814 sse815 sse816 sse817 sse818 sse819 sse820 sse821 sse822 sse823 sse824 sse825 sse826 sse827 sse828 sse829 sse830 sse831 sse832 sse833 sse834 sse835 sse836 sse837 sse838 sse839 sse840 sse841 sse842 sse843 sse844 sse845 sse846 sse847 sse848 sse849 sse850 sse851 sse852 sse853 sse854 sse855 sse856 sse857 sse858 sse859 sse860 sse861 sse862 sse863 sse864 sse865 sse866 sse867 sse868 sse869 sse870 sse871 sse872 sse873 sse874 sse875 sse876 sse877 sse878 sse879 sse880 sse881 sse882 sse883 sse884 sse885 sse886 sse887 sse888 sse889 sse890 sse891 sse892 sse893 sse894 sse895 sse896 sse897 sse898 sse899 sse900 sse901 sse902 sse903 sse904 sse905 sse906 sse907 sse908 sse909 sse910 sse911 sse912 sse913 sse914 sse915 sse916 sse917 sse918 sse919 sse920 sse921 sse922 sse923 sse924 sse925 sse926 sse927 sse928 sse929 sse930 sse931 sse932 sse933 sse934 sse935 sse936 sse937 sse938 sse939 sse940 sse941 sse942 sse943 sse944 sse945 sse946 sse947 sse948 sse949 sse950 sse951 sse952 sse953 sse954 sse955 sse956 sse957 sse958 sse959 sse960 sse961 sse962 sse963 sse964 sse965 sse966 sse967 sse968 sse969 sse970 sse971 sse972 sse973 sse974 sse975 sse976 sse977 sse978 sse979 sse980 sse981 sse982 sse983 sse984 sse985 sse986 sse987 sse988 sse989 sse990 sse991 sse992 sse993 sse994 sse995 sse996 sse997 sse998 sse999 sse1000 sse1001 sse1002 sse1003 sse1004 sse1005 sse1006 sse1007 sse1008 sse1009 sse1010 sse1011 sse1012 sse1013 sse1014 sse1015 sse1016 sse1017 sse1018 sse1019 sse1020 sse1021 sse1022 sse1023 sse1024 sse1025 sse1026 sse1027 sse1028 sse1029 sse1030 sse1031 sse1032 sse1033 sse1034 sse1035 sse1036 sse1037 sse1038 sse1039 sse1040 sse1041 sse1042 sse1043 sse1044 sse1045 sse1046 sse1047 sse1048 sse1049 sse1050 sse1051 sse1052 sse1053 sse1054 sse1055 sse1056 sse1057 sse1058 sse1059 sse1060 sse1061 sse1062 sse1063 sse1064 sse1065 sse1066 sse1067 sse1068 sse1069 sse1070 sse1071 sse1072 sse1073 sse1074 sse1075 sse1076 sse1077 sse1078 sse1079 sse1080 sse1081 sse1082 sse1083 sse1084 sse1085 sse1086 sse1087 sse1088 sse1089 sse1090 sse1091 sse1092 sse1093 sse1094 sse1095 sse1096 sse1097 sse1098 sse1099 sse1100 sse1101 sse1102 sse1103 sse1104 sse1105 sse1106 sse1107 sse1108 sse1109 sse1110 sse1111 sse1112 sse1113 sse1114 sse1115 sse1116 sse1117 sse1118 sse1119 sse1120 sse1121 sse1122 sse1123 sse1124 sse1125 sse1126 sse1127 sse1128 sse1129 sse1130 sse1131 sse1132 sse1133 sse1134 sse1135 sse1136 sse1137 sse1138 sse1139 sse1140 sse1141 sse1142 sse1143 sse1144 sse1145 sse1146 sse1147 sse1148 sse1149 sse1150 sse1151 sse1152 sse1153 sse1154 sse1155 sse1156 sse1157 sse1158 sse1159 sse1160 sse1161 sse1162 sse1163 sse1164 sse1165 sse1166 sse1167 sse1168 sse1169 sse1170 sse1171 sse1172 sse1173 sse1174 sse1175 sse1176 sse1177 sse1178 sse1179 sse1180 sse1181 sse1182 sse1183 sse1184 sse1185 sse1186 sse1187 sse1188 sse1189 sse1190 sse1191 sse1192 sse1193 sse1194 sse1195 sse1196 sse1197 sse1198 sse1199 sse1200 sse1201 sse1202 sse1203 sse1204 sse1205 sse1206 sse1207 sse1208 sse1209 sse1210 sse1211 sse1212 sse1213 sse1214 sse1215 sse1216 sse1217 sse1218 sse1219 sse1220 sse1221 sse1222 sse1223 sse1224 sse1225 sse1226 sse1227 sse1228 sse1229 sse1230 sse1231 sse1232 sse1233 sse1234 sse1235 sse1236 sse1237 sse1238 sse1239 sse1240 sse1241 sse1242 sse1243 sse1244 sse1245 sse1246 sse1247 sse1248 sse1249 sse1250 sse1251 sse1252 sse1253 sse1254 sse1255 sse1256 sse1257 sse1258 sse1259 sse1260 sse1261 sse1262 sse1263 sse1264 sse1265 sse1266 sse1267 sse1268 sse1269 sse1270 sse1271 sse1272 sse1273 sse1274 sse1275 sse1276 sse1277 sse1278 sse1279 sse1280 sse1281 sse1282 sse1283 sse1284 sse1285 sse1286 sse1287 sse1288 sse1289 sse1290 sse1291 sse1292 sse1293 sse1294 sse1295 sse1296 sse1297 sse1298 sse1299 sse1300 sse1301 sse1302 sse1303 sse1304 sse1305 sse1306 sse1307 sse1308 sse1309 sse1310 sse1311 sse1312 sse1313 sse1314 sse1315 sse1316 sse1317 sse1318 sse1319 sse1320 sse1321 sse1322 sse1323 sse1324 sse1325 sse1326 sse1327 sse1328 sse1329 sse1330 sse1331 sse1332 sse1333 sse1334 sse1335 sse1336 sse1337 sse1338 sse1339 sse1340 sse1341 sse1342 sse1343 sse1344 sse1345 sse1346 sse1347 sse1348 sse1349 sse1350 sse1351 sse1352 sse1353 sse1354 sse1355 sse1356 sse1357 sse1358 sse1359 sse1360 sse1361 sse1362 sse1363 sse1364 sse1365 sse1366 sse1367 sse1368 sse1369 sse1370 sse1371 sse1372 sse1373 sse1374 sse1375 sse1376 sse1377 sse1378 sse1379 sse1380 sse1381 sse1382 sse1383 sse1384 sse1385 sse1386 sse1387 sse1388 sse1389 sse1390 sse1391 sse1392 sse1393 sse1394 sse1395 sse1396 sse1397 sse1398 sse1399 sse1400 sse1401 sse1402 sse1403 sse1404 sse1405 sse1406 sse1407 sse1408 sse1409 sse1410 sse1411 sse1412 sse1413 sse1414 sse1415 sse1416 sse1417 sse1418 sse1419 sse1420 sse1421 sse1422 sse1423 sse1424 sse1425 sse1426 sse1427 sse1428 sse1429 sse1430 sse1431 sse1432 sse1433 sse1434 sse1435 sse1436 sse1437 sse1438 sse1439 sse1440 sse1441 sse1442 sse1443 sse1444 sse1445 sse1446 sse1447 sse1448 sse1449 sse1450 sse1451 sse1452 sse1453 sse1454 sse1455 sse1456 sse1457 sse1458 sse1459 sse1460 sse1461 sse1462 sse1463 sse1464 sse1465 sse1466 sse1467 sse1468 sse1469 sse1470 sse1471 sse1472 sse1473 sse1474 sse1475 sse1476 sse1477 sse1478 sse1479 sse1480 sse1481 sse1482 sse1483 sse1484 sse1485 sse1486 sse1487 sse1488 sse1489 sse1490 sse1491 sse1492 sse1493 sse1494 sse1495 sse1496 sse1497 sse1498 sse1499 sse1500 sse1501 sse1502 sse1503 sse1504 sse1505 sse1506 sse1507 sse1508 sse1509 sse1510 sse1511 sse1512 sse1513 sse1514 sse1515 sse1516 sse1517 sse1518 sse1519 sse1520 sse1521 sse1522 sse1523 sse1524 sse1525 sse1526 sse1527 sse1528 sse1529 sse1530 sse1531 sse1532 sse1533 sse1534 sse1535 sse1536 sse1537 sse1538 sse1539 sse1540 sse1541 sse1542 sse1543 sse1544 sse1545 sse1546 sse1547 sse1548 sse1549 sse1550 sse1551 sse1552 sse1553 sse1554 sse1555 sse1556 sse1557 sse1558 sse1559 sse1560 sse1561 sse1562 sse1563 sse1564 sse1565 sse1566 sse1567 sse1568 sse1569 sse1570 sse1571 sse1572 sse1573 sse1574 sse1575 sse1576 sse1577 sse1578 sse1579 sse1580 sse1581 sse1582 sse1583 sse1584 sse1585 sse1586 sse1587 sse1588 sse1589 sse1590 sse1591 sse1592 sse1593 sse1594 sse1595 sse1596 sse1597 sse1598 sse1599 sse1600 sse1601 sse1602 sse1603 sse1604 sse1605 sse1606 sse1607 sse1608 sse1609 sse1610 sse1611 sse1612 sse1613 sse1614 sse1615 sse1616 sse1617 sse1618 sse1619 sse1620 sse1621 sse1622 sse1623 sse1624 sse1625 sse1626 sse1627 sse1628 sse1629 sse1630 sse1631 sse1632 sse1633 sse1634 sse1635 sse1636 sse1637 sse1638 sse1639 sse1640 sse1641 sse1642 sse1643 sse1644 sse1645 sse1646 sse1647 sse1648 sse1649 sse1650 sse1651 sse1652 sse1653 sse1654 sse1655 sse1656 sse1657 sse1658 sse1659 sse1660 sse1661 sse1662 sse1663 sse1664 sse1665 sse1666 sse1667 sse1668 sse1669 sse1670 sse1671 sse1672 sse1673 sse1674 sse1675 sse1676 sse1677 sse1678 sse1679 sse1680 sse1681 sse1682 sse1683 sse1684 sse1685 sse1686 sse1687 sse1688 sse1689 sse1690 sse1691 sse1692 sse1693 sse1694 sse1695 sse1696 sse1697 sse1698 sse1699 sse1700 sse1701 sse1702 sse1703 sse1704 sse1705 sse1706 sse1707 sse1708 sse1709 sse1710 sse1711 sse1712 sse1713 sse1714 sse1715 sse1716 sse1717 sse1718 sse1719 sse1720 sse1721 sse1722 sse1723 sse1724 sse1725 sse1726 sse1727 sse1728 sse1729 sse1730 sse1731 sse1732 sse1733 sse1734 sse1735 sse1736 sse1737 sse1738 sse1739 sse1740 sse1741 sse1742 sse1743 sse1744 sse1745 sse1746 sse1747 sse1748 sse1749 sse1750 sse1751 sse1752 sse1753 sse1754 sse1755 sse1756 sse1757 sse1758 sse1759 sse1760 sse1761 sse1762 sse1763 sse1764 sse1765 sse1766 sse1767 sse1768 sse1769 sse1770 sse1771 sse1772 sse1773 sse1774 sse1775 sse1776 sse1777 sse1778 sse1779 sse1780 sse1781 sse1782 sse1783 sse1784 sse1785 sse1786 sse1787 sse1788 sse1789 sse1790 sse1791 sse1792 sse1793 sse1794 sse1795 sse1796 sse1797 sse1798 sse1799 sse1800 sse1801 sse1802 sse1803 sse1804 sse1805 sse1806 sse1807 sse1808 sse1809 sse1810 sse1811 sse1812 sse1813 sse1814 sse1815 sse1816 sse1817 sse1818 sse1819 sse1820 sse1821 sse1822 sse1823 sse1824 sse1825 sse1826 sse1827 sse1828 sse1829 sse1830 sse1831 sse1832 sse1833 sse1834 sse1835 sse1836 sse1837 sse1838 sse1839 sse1840 sse1841 sse1842 sse1843 sse1844 sse1845 sse1846 sse1847 sse1848 sse1849 sse1850 sse1851 sse1852 sse1853 sse1854 sse1855 sse1856 sse1857 sse1858 sse1859 sse1860 sse1861 sse1862 sse1863 sse1864 sse1865 sse1866 sse1867 sse1868 sse1869 sse1870 sse1871 sse1872 sse1873 sse1874 sse1875 sse1876 sse1877 sse1878 sse1879 sse1880 sse1881 sse1882 sse1883 sse1884 sse1885 sse1886 sse1887 sse1888 sse1889 sse1890 sse1891 sse1892 sse1893 sse1894 sse1895 sse1896 sse1897 sse1898 sse1899 sse1900 sse1901 sse1902 sse1903 sse1904 sse1905 sse1906 sse1907 sse1908 sse1909 sse1910 sse1911 sse1912 sse1913 sse1914 sse1915 sse1916 sse1917 sse1918 sse1919 sse1920 sse1921 sse1922 sse1923 sse1924 sse1925 sse1926 sse1927 sse1928 sse1929 sse1930 sse1931 sse1932 sse1933 sse1934 sse1935 sse1936 sse1937 sse1938 sse1939 sse1940 sse1941 sse1942 sse1943 sse1944 sse1945 sse1946 sse1947 sse1948 sse1949 sse1950 sse1951 sse1952 sse1953 sse1954 sse1955 sse1956 sse1957 sse1958 sse1959 sse1960 sse1961 sse1962 sse1963 sse1964 sse1965 sse1966 sse1967 sse1968 sse1969 sse1970 sse1971 sse1972 sse1973 sse1974 sse1975 sse1976 sse1977 sse1978 sse1979 sse1980 sse1981 sse1982 sse1983 sse1984 sse1985 sse1986 sse1987 sse1988 sse1989 sse1990 sse1991 sse1992 sse1993 sse1994 sse1995 sse1996 sse1997 sse1998 sse1999 sse2000 sse2001 sse2002 sse2003 sse2004 sse2005 sse2006 sse2007 sse2008 sse2009 sse2010 sse2011 sse2012 sse2013 sse2014 sse2015 sse2016 sse2017 sse2018 sse2019 sse2020 sse2021 sse2022 sse2023 sse2024 sse2025 sse2026 sse2027 sse2028 sse2029 sse2030 sse2031 sse2032 sse2033 sse2034 sse2035 sse2036 sse2037 sse2038 sse2039 sse2040 sse2041 sse2042 sse2043 sse2044 sse2045 sse2046 sse2047 sse2048 sse2049 sse2050 sse2051 sse2052 sse2053 sse2054 sse2055 sse2056 sse2057 sse2058 sse2059 sse2060 sse2061 sse2062 sse2063 sse2064 sse2065 sse2066 sse2067 sse2068 sse2069 sse2070 sse2071 sse2072 sse2073 sse2074 sse2075 sse2076 sse2077 sse2078 sse2079 sse2080 sse2081 sse2082 sse2083 sse2084 sse2085 sse2086 sse2087 sse2088 sse2089 sse2090 sse2091 sse2092 sse2093 sse2094 sse2095 sse2096 sse2097 sse2098 sse2099 s

。除此之外？有些指令可以的。什么副作用？因为我们的 source code 里面就会漏的八个东西，那这八个东西我们也可以一次漏的八成。这是没问题的。就源代码里我们就要缓存这个东西，那么是否可以假定缓存一定不会越界。那么是否可以假定漏的四成也可以？就是漏的四个，那么我就可以漏的四层。但不一定能漏了八成就是这个地方，因为它只漏掉了四个，那我去漏了八个就可能缓存到外面的东西？

...(00:46:52): 除了这个东西是四乘多少有几结构的限制以外，还有一个限制，就是做加法，做减法，做这些东西可能是没有收益的。就是在有些比较拉的体系结构上它 SIMD 的这个算法。它不一定比标量快。那么这就是我们待会要讲的，对每个指令需要一定的代价模型去衡量它。Ok.

...(00:47:28): 回到这个 PPT，我们现在已经知道 IR 到 IR 的变换是可以通过已有基础设施去完成的。所有的 target，比如说 RVV，比如说 on，比如说又 86，它都是共用的这个东西。那么有个问题就是这些 SIMD 指令集之间差别很大。比如说 SSE 跟 RVV 差别就挺大的，它如何共用？LV M 给出的方案是提供一些信息来指导这个公共的 pass。当然，这个信息就不会是每个体系结构特别多细节，而是给出一些大概有多少信息，然后去完成这个任务。具体来说就是它会有一些虚函数。后端在实现的时候就涉及到去重写这些虚函数，比如说有的 cost 在这个体系结构，你可能是一，然后在那个体系结构，它的卖点就是个四？终端可能会感兴趣，什么问题，就是这些公共的 pass，他可能会感兴趣什么信息，大家可以猜一方面是向量有多长，就是他问。你的体系结构向量有多长，是 256 位长还是 128 位长，这是一个很重要的信息，不然这个终端生成 IL 的时候它可能就超了。另外就是运算的这个代价。后端的实现，就是如果硬件支持的话，我就直接输出这个代价。如果后端不支持的话，后端就要诚实的告诉终端我做了什么，然后输出一个根据后端做转换，然后输出这个代价。

...(00:49:21): 当然，我刚刚说到诚实的，那么有没有一些后端是不诚实的，当然是有的，就是在实现一些草案的时候。这个东西并不是根据后端真正做了什么，而返回他的 cost，而是程序员认为后端做了什么，返回这个 cost。待会我们会讲一些认为的例子。

...(00:49:45): 那么既然提到 cost 那么什么是这个 cost，后端就是大家约定了有这么多种代价模型，一个是这个存储量倒数，一个是指令延迟，一个是指令产生的二进制的大小，还有一个是二进制的大小与延迟的加权和。有这么多种代价的类型，那么相当于我就要问后端你的加法指令的存储量倒数是多少？你的加法指令的延迟是多少？你加法指令是多少？这个时候就有一个很有趣的事情。大家可以猜的是？什么唯一。它会根据不同的 CPU 的模型去返回这一值，信息是有的。他就只能估摸着返回一个书，因为这个代价是 LMIR 里的，他就只有一个 A。那比如 CPU 有十个 a，它就只能根据这个 a 的某些操作数的信息去返回它的代价。这个事真是存在的，因为体系结构可能有不同的指令，但是 LBM 中只有一种 sha vector。那么有些 pattern 交错的奶体系结构可能是已经有的，但是随便一个 pattern，它可能就没有那么没有的那种 pattern 的代价肯定就会高一些，因为涉及到去拼一个 pattern。

...(00:51:25): 一件有趣的事情就是。虽然它提供了如此多种代价的类型，但是绝大多数后端对所有类型都会输出同一个数。虽然类型很多，但是其实没有很多人 care 这个类型到底是如何 work 那么就是我们首先设计上支持如此多的类型，但是实现上支持这么多太复杂的加法，就设一个一页。减法也是一乘法就多点二。基本上就是这样，比较随意的完成这件事情。众所周知，浮点是比较慢的，那就来个四或者八一般就这样写了。他是取决于一些实现的，他不会那不一定，那有人家有可能是没有 FPU？他可能要调库，这都是有可能的。我们需要编译器需要处理各种各样乱七八糟的 target 这是正常的。但是这件事情真的至少在 LBM 中。我写到的就哪怕是别人的 target 基本上都是如此处理的，因为这个类型实在是太多了，你要去算的话，其实根本是没有这么多 switch case 的，那没有办法我们就给他。都说是同一个数，简单一点。因为用户没有报你这个性能有问题，那就用？

...(00:53:11): 我们总结一下，刚刚提到 victor，首先这个提供信息，主要是通过 override 虚函数实现大家都知道虚函数是什么吧？Ok 然后这个信息主要是包括每个指令代价，然后计算相应计算器的长度跟数量。数量也是有一定作用，它会衡量这个寄存器的压力大不大。如果它太大的话，它有可能会放弃这个向量化。还有就是这个代价做的粗糙一点也没有问题，实际上大家都这么干的。就这个大模型没有很细。下面这个问题就是我们已经有了向量的 IR 我们要怎么生成？

...(00:54:00): 汇编或者这个 object 生成二进制 ok 这就是我们介绍的一个算法，这个算法就是 LM 中的指定选择算法。它的数据结构叫做 selection。同学们都知道什么是带。有像无环图，可能会在什么离散数学之类的课学到我现在就假定大家都知道什么是 D。哈哈。它的特点是在 BB 内，它是不包含控制流的。就是当代码走到这的时候，我们已经看不到函数，看不到控制流的转换，比如说 if branch，比如说这个 for loop 都是看不见的，所以说想在这个阶段做类似于循环向量化不可能的。

...(00:54:57): 拿到它的时候，它就已经只有 BB 内的一些数据流的过程。那么它的特点就是用每一个节点代表一个运算。这个节点，它会有

一些前驱，这个前驱就是操作数。待会给大家看一些图片来感性认识一下，因为这个挺重要的。然后算法就是很简单，就是去图上做一个 pattern match。Pass 然后给它 rewrite 成一些。我们比较后端会比较喜欢的结构，然后逐渐的把一些非法的操作变得合法。然后这样的话就能生成最后的指令。这个代表结构，大概就长这个样子，大家能看懂这是个什么程序吗？首先这是一个一，然后这是一个二。常量，然后常量做一个加法之后会得到一个新的数值，然后新的数值这里再做一个乘法。是不是很简单？

...(00:55:58): 感受一下。这种设计，在现在的各种别的编译器中挺常见，比如说可以自己给大家看一下 v8 的这个 T。它也是，比如说是一个 function 是 fx。它要返回 X 加一，那么这个地方的节点就是啥，是一个 add 的结果的操作数是啥，是参数 X 和常数一。Ok. 除了表意这种简单的算术运算之外，它还会有一些别的需要表意的东西，比如说。我们知道是会有一些副作用的。前一个 store 和后一个 store 之间在这种结构上是看不出来区别的。还有一个问题比如说 VI 和 F 都是。我们怎么区分它到底是怎样的？这里只有个号的话，你就可能不知道它到底是什么。还有一个问题就是控制流。控制流刚刚已经提到它基本上是不能支持的。

...(00:57:20): 首先讲这个 load store 要怎么办？它的解决方案是我们返程的时候多接触一个参，大家听懂为什么 load store 是不能表意的吗？我可以画一下。假设现在程序是这个 store 这个 PTR 答案是 P，然后我们往 P 这个地方存一个一在。迟到 P 这个位置再放一个二它画成图可能就长这个样子。是 store 节点？它有一个参数是 P。然后它另一个参数是一下面是第二个 store 节点？还有一个参数是二还有另一个参数是 P。那么现在问题来了，假设你只能看到右边的结构，哪个 store 会先发生哪个 store 会后发生？就看不出来了。我们把这个 store 放到前面 store 放到后面，这个时候放或者调换它们的顺序，在这个图上面都是可以的。因为因为这个 store 依赖是 P 这个 store 依赖是二，这个时候也只依赖 P 和一，它们之间是有一个顺序关系的。那么持续这个顺序关系？我们如果把这两个东西的顺序给调换一下。那么同一个内存的 P，它就会导致调换之后，本来我们期望它留下是二。但是调完之后，这个地方就被写了个一进去，那程序就错了。成绩错，可是天大的事情。你做再多的性能程序，错的话都记。

...(00:59:21): 这个问题要怎么解决？大家有没有什么想法？怎么加。我们对所有的 store 都要加我们对所有的 load store 都要加一个边吗？如果我确认他之间是没有依赖他就可以不加对吗？要怎么加边？就是这是第二个 store，那我们给它连一个 B 连一个虚线这样。那然后这样就表示它是依赖。对那真实的答案是这样的，就是我们在缓存的时候，会多接受一个参数。这个参数代表土进来的 memory 状态。然后每一个节点，它都会吐出去一个 memory 的这个状态是一个抽象，你别管我是什么状态，反正我就需要一个这样的参数。对于所有的我们都要让这个参数是存在的。目前解决方案是这样！

...(01:00:48): 解决 VI 的跟 FI 的，这会让我们想到一个方法，就是我们对所有的这些数据类型都带一个 T。这样的话，比如说 VI 的跟 FI 的，我们就能通过它是 V 什么什么什么还是 F 什么什么来区分。

...(01:01:05): 然后控制流，它是不能支持一些各种复杂控制流的，前面也讲过了。比如说这个爱的这个东西，他就是在真正的 LV MR 中就长这个样子，我们可以看到这加了一个 typei32。这个 type 是 i32，然后这个地方最后做个加法，第一个操作是这个玩意儿。第二个操作是这个玩意儿。那刚刚讲这个 load store 的 store1 和 store2 最后要怎么解决，这就要引入一个概念叫做 chain。他的这样，我对所有的。所有的内需要缓存的东西，我都需要接受一个东西就是特殊的内存状态，每个程序开头都会有一个。然后这个 store，它因为在前面，所以说它要先把内存状态作为依赖，它会吐一个状态。我们给它叫 s1！然后这个 store 它就会依赖于这个状态。所有的 store 都必须多接受一个参数。再多吐一个参数熟悉的人大概就会瞬间想到这和某些 mona 的之类的概念其实挺像，当我们需要。需要解决一些副作用的时候，我们就把这个副作用当成参数，传给这个函数，这样就行了。但是这意味着你你多接受一个函数，然后再吐吐一多接受一个参数，再吐一个返回值回来的话，就好像你吃了饭总得给它拉出来一样。

...(01:02:46): 所有的 load store 都需要去多增加一个这样的参数。就比如说刚刚这个二？刚刚在真正的 selection 战略中就长这个样子。我们注意到这个 store 就放到了这个地方，这个时候放到这个地方，它有一条蓝色虚线连到这个 entry。

...(01:03:38): 就让这个拖过来，对一个摄像头看好的，那没事，我们先讲这个吧，就是第2个 storestore1 然后这是第1个 store 第二 store store，这个是第一个 store store 一。

...(01:03:56): 我们可以看到蓝色这条线就是我们感兴趣的東西，它会多接受一个参数，从这个地方连一条蓝色虚线，然后这个地方再连一个蓝色虚线连到第二个 store 这样的话我们就知道它的相对顺序是什么。当然，不是所有的节点都需要接受一个蓝色的，这样特殊的线，只需要对那些有副作用的进行处理就可以了。这是因为我们需要让它之间有一定的，我们可以预测的顺序。不然的话处理这种带副作用的节

点就会出问题。这个设计我们可以看一下谷歌的这个 v8 是怎么搞的，他们相当于说是有一个。非常明确的 start 跟 end 所有的节点，他可能就会接受。除此之外，还有因为 javascript 是有异常。所以为了去异常这个东西，它还需要一个另外蓝色的线去连关系。但是基本上所有的编译器 IR 都会有类似这样的设计。就是我们去多加一个代表副作用的节点去建模程序的设计副作用的，比如内存访问之间的指令的一个依赖关系。

...(01:05:26): 那么下面就介绍一下指定选择的过程。这个指令选择的过程，主要是合法化需要介绍一下 deck to deck is 基本上是。可以自动生成的。那么现在大家就比较关心如何合法化这个 T。然后另一个是合法化这个 operation。type 就是说可能有人来的是一个长度为 256 位的整数，你要怎么处理？还有一个 operation 就是说。我们的体系结构，它只支持加减法，不支持乘法，你是不是需要一些别的玩意来模拟一下？

...(01:06:08): 先看一下如何合法化这个 type。这个依据是后端会给定一系列的，它支持什么 type，比如说支持 I 32，支持 I 64 就是 32 位的整数跟 64 位的整数。可以支持。就类似这样的东西。或者说比如说 v8 i32 长度为八个的 SSR 每个另一项都是一个 SSR。他给出了这些支持和 type 之后，它就会自动完成类似于 promote expand 这类的操作。promote 把。把 i8 变成 i32，就是把它的这个数据类型变大。expand 就是说假设我们是 16 层 i8，但放不下了，我们可能会拆成两个 8 层 I 8 之类的操作。这些东西都是后端会有一个框架，所有 target 都统一完成的。那么现在开始提问这个 type 跟 OP 是否相关，就我们合法化 type 类型这个合法化过程中，跟它的涉及到操作是相关的。无关的。我们如果要假定他无关的话，我们就必须假定这个体系结构要支持 i32 的话，它就支持 I 三的乘法，除法等等所有的操作。我们要认为它是有关的话，我们就需要对每一个 OP 跟 type 都做一些代码操作可能会让程序变复杂。

...(01:07:49): 如果你是来设计编译器的人，你会让合法化 type 的过程跟 OP 相关吗？我们刚刚介绍有合法 type 跟合法 OP。就只有这两个？先不说我的 S。那么我们就说正常一下行吧！这个 OP 是否是跟这个 type 的合法化是否跟 OP 相关的，比如说我们要认为 i32 合法的还是认为成逗号 i32 合法？我们 i32 就永远的合法了？

...(01:08:54): C YY 你讲一下！你觉得这个跟 OP 是否是相关无奖竞猜。是相关的。答案是这样的，答案是在现在这个状态的确不相关了，然后他的确会在下一个 stage 处理，那么这样的话就会带来一些问题，就是某些 as，它的这个知识是相关的。就会很恶心，这个具体的方法是在 LM 代码中编写叫 add register class 这样的东西，然后你就能通知他到底有什么样的类型是合法的。你只能做这件事情就是告诉他什么是合法的。然后具体跟某个 type 是合法，它是这个过程中是不能救你。那么合法 type 的一些例子，比如说 i31 有的人编译程序，他比较逆天，他就喜欢用 31 位的整数，那么就只需要。有什么问题吗？

...(01:10:02): 他就喜欢用 31 位的整数，这个可能也是可能会生成 i7 这样的玩意的。那么对于这种例子，我们就可能会给它放到 s2 里面。比如说四乘 i32，要是这个体结构是 BPF 这样的比较拉的东西的话。他怎么办？终端已经生成四乘 32，但是你你他给的是个 BPF，那我们就把四乘 32 拆成四个 i32。这个就像这样，这是一个节点，它是 add。然后四乘 i32，那么我们在如果这个 target 是不支持四乘三的，我们就把这个拆成。四个 A。大家可以想象一下。听懂这个过程了吗？就是匹配到了四乘 s32 这样的非法节点的时候，我们就给它拆成四个不一样的节点。然后每个节点都是标量，这样的话它就能处理的，比如 BP。Ok。然后比如说 16 乘 8，它可能在某一体系结构上，它只能支持 8 乘 32。我们就需要先把 16 乘 a8 给它变成 16 乘 32，我们就可以用更大的数据类型去表示一个更小的。那但是 16 乘 s2 这玩意瞬间就放不下了，那怎么办，那就只能再把向量又拆成两个。就是我们可以组合拆的过程跟 promote 的过程。

...(01:11:39): 同时的去 promote 1 下再拆一下，这也是可以的。下面来看合法化 OP。合法的 OP，它的对象是 OP 跟 type 元组。为什么它的对象是合法化 OP 跟 type 元组，每个 operation 它都有一个 type。Operation 可能对特定的 type 是有效，但对某些 type 就是没有效果的。我们去做这件事情的时候，永远都是去操作 OP 跟 type 这个。原组的。那么对于每个 OP 跟 type 后端需要决定他要做 expand 和 customer legal。那大家可以猜一下 expand custom lego 都是什么意思？首先，expand 就是四乘 s2 拆成四个 s32，就类似于刚刚的过程。就是请求 LM 的某个流程，把这个向量的操作给它变成一些标量操作，然后我们假定标量操作是支持的，就是说。要不你别帮我写，我要自己来自力更生写 C 加加这个 lego 这个体系结构有非常好的对应，比如说 a 的这个操作在很多体系结构是有这个指令的，那么就它就完美的对应了这件事情，我们就可以说明它是 lego 的。Ok。

...(01:13:22): 下面来做一些小练习，想必大家已经听过了，听懂了李范跟 lego 现在假设某二支持向量指令有这么一些。整数的加法和减法是可行的，然后向量移位，但移位置必须常数不能是一位一位一个不同的长度。还有你可以做被运算。还有就是布置非对齐放。那么我们去合法和艾特和不讲竞猜。没错，现在我们如何实现这个 SRA，大家知道 SR 是什么东西向向右算数一位。对所以说应该选哪个？你

只能三选一？你编写一个 C 加代码，你检查一下它的操作是不是常数的话，我们就不动了，就让它这样继续下去就行了。如果是常数的话，你赶快把它 expand 了一下？下面再看一个乘法怎么办？不对 expand。因为你看这个限量指令是没有乘法的。我只说了加法和减法，他没有说乘法。是不能模拟的，因为这个乘法的操作数可能是一个变的。然后你你去搞一个加法去模拟的话，是会比较困难的。是可以做的，没有必要，很可能不如标量。因为 explain 对那标量是支持乘法的一般体结构都是支持标量乘法？Ok.

...(01:15:27): 再下面来看下一个 load store 怎么办。就必须因为因为它不支持非对齐缓存，那我们就要看一下它是否是对齐的话，就让它继续下去。如果不对齐的话，可能就需要做一些操作，比如生成 copy。B swap 字节交换把大端序变成小端序怎么办。就变成标量了。也可以。就可以凑出来，所以我们可以用给它凑一个出来，因为绝大多数现在能凑出来的东西都是慢于都是快于半成标量的。

...(01:16:23): 还有一个是 ABS 求绝对值。大家感受一下应该用什么，就是要么就变成标量，要么就凑一下，要么就你肯定自杀不行。是能凑出来的待会我们会讲怎么凑挺有意思的。有什么问题，行，这个 load store 刚刚已经讲过了，我们会根据它的这个 point 它的操作数是怎么个对齐的？这个对齐信息是包含在它的操作数里面的，我们就要决定他要不要拆成两个，比如说 mips 里面就会有 L D L L D R 这种指令给它拆成两个。如果当然 L D L D R 也是有对其要求的。如果还不支持的话，就需要来个 M cop。能理解吧，就是我们先吧 M copy 把站上某个不对齐的地方 copy 到一个对齐的地方，然后再从对齐的地方开始漏的，这就非常慢。

...(01:17:30): 这个地址对齐是从前端也是从 GCC 这种玩意一路的分析出来，然后从终端传到后端传到这个位置的。它必须要分析正确传递正确。之所以这样说信息之后会给大家看一个传错了。哈哈。就单靠谱。不是调库是不是 intrinsic 就是 LV M 中去生成 load store 去做 mem copy 这件事情，但不是生成一个 call me copy。这肯定是不行，比如说我们要漏的一个不对齐的 i32，那你可能就需要四个 i8 的。漏的就是你要漏了一个 32 位的整数，但它不对齐，我们能够漏的一个字节，然后把每一个字节的 load 出来，再把每个字节都 store 的对齐的位置，就可以 load。这样的所有体积都是这样的。如果能做到的话，当然是可以的，但是默认的框架是 load。就是它有一个通用框架，如果你的体系结构没有写自己的 C 的话，那么就会走这个框架就是 load 加 store，它非常通用。对一些短平快的开发，一开始都是这样的。现在来看这是 C。不是它是生成 bag 上的 loader 跟 store 就是我们是这个数据结构上操作，它已经不是 LV MIR 了。明白吗。所以我们会在这生成节点。它不是 LMI。

...(01:19:33): 绝大多数优化都在终端后面还会有一些 machine ssa 之类的骚操作是可以优化。但是大家一般都不在那做。Ok 我们继续讲就是这是 chrome 的一个需求。那么它其实是可以通个 1 个移位和 or 拼出来的，比如说我们需要把 a0a — a2a3 它是四个 B，那我们可以移位成这个样子。有成这个样子之后再给它握起来，这是能做到的，就是在向量里面拼一下。下面来看求绝对值。它可以先把。这个 X 给它算术又移它的 size 减一这么多位。然后假设这个是不是 T，然后求 X 的绝对值，就等于把这个 T 和 X 加起来再求 X。是不是很神奇，真人求绝对值。大家可以想一下为什么？T 是一个 T 把符号位移到一个整个上，这个爱的 T 就是假设它是个负数，我们就给它弄个负一。它在一下这个玩意就会变成它的正的版本就是补码小小骚操作。这个例子就是比如说八位的整数负五，它等于这个玩意，那 T 就是负一。那它把它加起来就等于这个玩意在以下这个负一就写错了，这最后会抑或出来这个零。抑或衰就会变成正误？

...(01:21:30): Ok 然后现在是一个怎样选择的小结，它的整个算法结构就是刚刚画过的 selection bag，它是一个有向无环图。对于它的整个的指定选择的过程就是带上面的这个 rewrite。涉及到 type 上的 rewrite 跟 OP 上的 rewrite。OP 的时候，是 OP 的时候，是一个元组，就是我们在 rewrite op 的时候上也会关注它 type。type 是非法的话也会做一些事情。

...(01:22:07): 下面来介绍指令选择中的一些趣事。放松一下，大家听听一些比较严肃的数据结构与算法，我们现在来讲一些有趣的事情。就是前端的对齐信息，我怎么感觉是他瞎编的，怎么感觉错了。第二个是这个 S M D 居然不支持整数乘法。那怎么办？然后是这个 new 是太天顶星了，这谁想出来的，然后还有一个就是别人全 legal 我全得 expand。然后这 i32 跟 a8 都是 I，我们可以倒腾一下，下面来介绍一些趣事，到底怎么去法就是首先是前端这个对接信息，为什么说它是瞎编？就是我们先介绍一下背景，就是很多体系结构会支持单元元素对齐的这个漏的就是我们可以给它拆成两个，那拆成两个，前提是你至少要对齐到 S M D 的一个元素，那么多对齐。这是合理的要求。为了优化，我在编程的时候就假定了 double 至少对齐到八。这是合理的假定，因为。

...(01:23:23): 怎么会有人把 double 放到一个不对齐的地方？我真的拿到了前端给我的东西，他说 double 是一。你不觉得很神奇吗？就是这个 double 怎么他怎么可能是 online1 的玩意，后端就给我生成了 M cop。因为这个。

...(01:23:50): 到底是啥问题，我先给大家看一下这个症状如何，我现在还留着证据。fl nvidia 的编译器。他的这个代码都能看清楚。上面是 f

fortune 什么。上面这个玩意是 fortune 的源代码。能看见。这个源代码就是 complex complex 就是复数是两个 double，我们看下它吐出 LV MI 它在这个 entry 这个地方 a local 一个对齐到16的两个 double**太厉害了，竟然能对齐到16。它下面就开始 load 了 load 中括号的尖括号 double double ptra。这明显是个 bug，就是你看这个程序。你你看常数不是这个程序 a local 明明是个 a16的东西，然后就紧接着它下面几行就开始 online1了。这个 F 浪多少是有点问题了，我觉得在下面说。这个时候发生的事情，我就说。could you please double check why 这个东西是一？他说这个 allocated 是16，然后为什么就对齐到一了，他说什么你能不能给我搞个 reproducer？我说你这 reproduce 就在上面评论里面。下面开始解释是为什么就是这个 LMR 的 verify，它会添加一个一如果这个前端是没有生成这个 alignment 的。然后这个问题就是这个 fl2，这个 fr2在计算这个玩意的时候给漏了，然后他说他将要提一个 PR 来 fix。我们来看一下这个 PR。

...(01:26:02): 这个 PR 大概就是说这个原先的写法是某个特殊的曾经的某个需求给加上去的，它只会在操作数为 complex 的时候是会有问题的。你要是不处理复数的话，它就是对的。但凡处理到复数，它就会进入一个奇怪的三目表示，这个 flag 就错了，然后必须要调它的某个 U 来，才能把它给搞对。

...(01:26:37): 这里也要吐槽一下。很多编译器前端包括 clone 跟 RC，它生成 LI 都不是生成文本。它是生成那个 a st1在代码中就是数据结构是一个树一样的玩意。但是 CF R 就是我们现在给大家看的这个玩意，它生成是文本。文本 A 就是他先吐一个文本到你的 temp 下面，然后再由 LV LLC 来读那个。这就会出一些问题，比如说。他会 LINK LM 中的一些库。它为了某种神奇的原因就会生成 text。不知道，反正 f2已经被那个 nvidia 给弃用了，大家现在说的都是什么，lv N project 里面那个 NEW F 就是 FL 为什么会有两个原因，可能就是因为。

...(01:27:46): 对 nvidia 那个 F这个 FL 是类似于 C 语言的 C 加加写的。这肯定就不太符合 LV M 的标准，并且它的这个技术路线比较奇怪，生成了一个。文本 IR 然后再让 LM 去 pass。比较逆天。当我们发现问题之后，我们肯定是不报给甲方的，因为如果我们不报给甲方的话，我们把这个玩意修了，这就可以算是我们 SM B 的加速比。这个是一个重大的 bug，可以直接导致浮点慢非常多，因为里面有非常多的 complex 这个问题就是这个 CF 会先生成 IR 字符串吐到一个文件里，然后在 MP。这个对齐的信息就丢了，他就没打出来，然后 LM 在 parse 的时候就自动补了个一。这个是我们看到的東西。对就是这就是因为他这个的时候压根就没有生成是多少。我们重新 purse 的时候，这个 all 就变成了一。它是有的数据结构里面是有的，但是它是因为一些历史原因，然后对这个 complex 的处理是漏了，你看它这个修复就是这样。它的别的类型是能生成正确的 alignment，但是 complex 好像有某种奇妙的处理，然后就漏了。也不知道他写的是什么玩意。

...(01:29:38): Ok. 解决前端对齐这个问题，现在来吐槽，第二个就是这个 SM D 它居然不支持整数乘法，那怎么办？其实这个在别的这别的体系结构里面也是存在的，比如说叉86的 CMD，它在一开始的时候就没有乘法。它虽然有这个 PM 什么 U DQ，但是它跟 LV M 中想要的那个 SM B 乘法是不太一样的。他会把两个64乘出来就是一个。I128我们希望两个64相乘也等于两个64就跟普通的整数乘法是一样的。他这样搞的话，就我们每次弄完就需要一下，那相当于高的那个64就没有什么用。他如果不支持整数乘法的话，有些的这个 workload 它的确会涉及到 SM D 乘法，这就会导致 ISEL 或者 cos model 写的很复杂，因为。如果支持乘法的话，那你前面那个 cos 就会比较慢，当然我们前面已经讲过 cos 是比较不太精确的。那么为了省事，cos model 甚至会他说 SM D 乘法代价是99824353。

...(01:30:57): 我随便搞一个比较大的数，终端很可能就不会向量化了，那我就不用支持了。哪个书。它只是一个我随便写的大数字而已。就实际编程的时候，可能是连滚键盘出来的目的就是老师说你这个乘法不行，你别让它生成。那有什么办法就先搞个这种事情，在别的体系结构中，时而也会遇见一般因为开源大家也就休了，但是在一些闭源的神秘架构中，就可能发生。因为前面讲的 cos model 并不是你做了什么，然后按照你做了什么来生成的，而是你告诉终端你做了什么，当然我就可以不诚实的说，我觉得代价就很大。

...(01:31:50): 我说的不是叉86。下面是这个趋势是我觉得你用真是个非常厉害的指令集，就是它这个指令集，虽然只有128位，但是它加速比是比叉86还高的。这是为什么，因为它的这个指令。它的支持特别完备，基本上你想要什么就有什么，比如说乘法它是有的，并且它对特定 pattern shuffle 是有特殊支持的。我说的不是叉86。还有一个特点就是 OP 跟 type 是正交的。就你只要支持了这个 type，那他就支持所有可以支持的 OP。而不是说部分的 type 跟部分的 OP 就搞得编辑器很难受，比如说他要支持 i32加法，他就一定会支持 a 八加法。

...(01:32:48): 比如说这个叉64中的 SATD 是一个比较热的函数。我们看一下类型，它的 pixel 是 U。然后它的别的数据类型是16或32。它的整个比较就计算量比较密集，就是算 a0a1a2a3，然后它的逻辑看起来就能被向量化。这看起来这东西就很规整。下面就是做了一个哈德玛的四，这个哈德玛四大概是说做了1个类似于位置交换，就是 s0加 s1s0又减 s1 s2加 s3s2又减 s3。这样的东西。最后 D 又等于 t0加 t2t0减 t2t1加 t3t1又减 t3，那大家可以发现。看代码可能比较复杂，我这里有一个图来展示整个过程。就是 pixel1假设有这么几个元素，那么它刚

刚的 temp0需要这样的移位才能达到。在这样的移位之后需要求个 sum，并且还要求绝对值，当然刚才已经讲过求绝对值其实并不复杂，可以拼出来，那么我们现在就比较关心如何 SIMD 上面这一块这个过程。

...(01:34:08): 这个 function 大概占18%的运行时间，只要优化这个 function，它就会跑的比较快。我们来看一下 new 大致是什么？这个 arm 的文档，他有专门的一个目录里面就有一个东西叫做 permutation，然后这个上面是大概就是讲的是我们要如何去。

...(01:34:34): 去这个 vector 它的支持什么，它支持这样的 C 把 a 换成 AB 换这 C 换成这 D 换这，然后给他下一下。他也支持这样的 C。就 a 换到这个地方，然后 B 换这 C 换这 D 换在这，然后 E 又放在这，然后 F 又放到这。他又支持这样的 S。a 放这 B 放这 C 放这。它还支持这样的 S 就是 a 放到很远这个地方 a，然后 H 放到这个地方 H 能看懂吧？他还支持这样的下。这样的家伙能看懂吗？就是把它的偶数位放到这儿，然后偶数位放到这个地方。然后偶数位一下放在这儿，然后偶数位放在。他是为了模拟这个矩阵的转制这个过程。然后大家可以看到它的这个各种各样的虾头是非常全的。

...(01:35:56): 他的有人评论看一下。转一下摄像头。大家可以看到我们面对这个需求就跟他的这个 shuffle 能够完全对应上。不得不怀疑他设计这个指定企业的时候是有考虑到这种需求的。那么。所以说他就会。为什么又64能够在 arm 上有特别高的加速比，但是可能别的体系结构就没有这么好。

...(01:36:37): 现在来讲一些 future work 就是一些不太存在的东西。第一个是 global il，这个 global il 是说现在 ICL 是 PER BB 的，它会丢失一些信息。另外一个问题是，in combine combine 会重复代码 in combine 就是说 LV M 终端做的这个 combine combine 变成 selection 之后，它要做一个 combine。它会重复很多一模一样的代码。那么 global icl，它就是说我们全局的含 BB 的进行指定选择这个玩意的。未来可期现在写的还不错。A R 和64跟 R V 的后端还不错。信创后端其实是不太行的。我记得我去年大概。不算特别信创英国信创去年大概这个时候我给 BPF 交了一个 group icl，然后到现在还没开始写。就是交了一个 INIT 代码，但是其实还没写。然后第二个事是这个 RVVSVE，这框架表意是比较困难的。什么。这个地方就变成 future work。因为看起来就比刚刚那个要遥远。然后这个玩意在 MIR 中是有一些实践的，但我觉得做的也一般吧。感觉其实不如手写，因为像 M copy 这种玩意，它其实可以通过手写汇编，就不需要编辑器这种像话像又64这样的东西。

...(01:38:24): 像 RV 那种变长，其实也不太适合，刚刚大家看到又264？就讲完了，下面今天讲的一个总结。这个向量化，这个收益是很看 workload，这是第一点。比如说这个 arm 就在又表现特别好。但是，像别的体系结构或者别的 workload 就表现不行。然后第二个是标量 A R 到向量 AR 我们只需要指导已有的框架就行，主要指导方式就是给他提供一个 cos。这 cos 可以瞎编。第三个是向量 IR 到具体的指令集。然后靠的主要是 select 上的花式操作。还有一个就是增强向量寄存器的长度。不如增加点真正的指令，增加点操作。变长没有什么用。多操作才有用。未来的工作就是这个 RVVSVE。可能还得加油干。大概就没有了。下面提问时间。

...(01:39:45): 哪个地方就是那个地方？我刚才有讲到 cos model 是很粗糙的，那个 loop vectorizer 是会生成特别长的代码，那么我们很多时候能看出这个 loop vectorizer 在这个循环的长度很短的时候，它就有负优化的。主要还是因为 arm 非常多，它不一定适合鲲鹏920。

...(01:40:43): 你说在正经的开源开发中怎么定还是在？

...(01:40:58): 其实是很难衡量的，因为在座也有做处理器的，大家知道这个处理器的每一条指令的 cost 其实你是不太好衡量的，所以一般来说指令集，做 CPU 的会给你一个手册，这个手册会涉及到那个指令它的 latency 是多少，这个数是有的。然后很多 cos model 是参考 latency 来做的。我看一下这有这个地方就是其实很多 cos model 是参考指令延迟做的。它会根据这个指令，从从开始算到有结果会有多少 latency，然后就把这个值来当做一个 C。

...(01:42:01): 一般不会做 normalization。你做的我要怎么写9982435？这个东西，比如说在后端，在实现的时候，它会说我的乘法指令是多少，那么四乘这个乘法是多少，它就会递归的掉自己。那其实这个过程你是比较难帮得住一个范围的。他就是把这个递归掉自己，然后加起来。他挺 work 的。

...(01:43:00): 有很多不正交。

...(01:43:06): 因为你在做合法和 OP 的时候，你就会涉及到这个 T。比如说你对乘法的 i32是 lego，你可以设乘法的 v8s32就是。是可以的。对它主要是为了方便程序员编程。因为我们可以假定绝大多数都是正交的，但是有部分比较奇怪的东西不正交。比如说在前面那个过程，你就可以把 i7i31这种玩意给处理掉，然后把 type 变成某种比较好看的样子，不然的话这个圆组就会非常大。就比如说 OP 你处理了 i32，

那你要不要处理一下 i31? 在前面不做这个事情, 那这个地方就会写非常多的 case。现在那个 global iel 基本上就这样写的, 他会说 legal for 巴拉 type, 然后巴拉 type 它会比现在这个框架要优雅一些。

...(01:44:29): 还有什么问题吗? 展开讲一下怎么做, 就是我们没有做。但是大概思路是这样的, 就是我们知道内向的假设它只能处理 i32 加法, 那我其实可以只用一半来放一个 s6。然后我们现在来做加 s6 的这个 SMD 加法, 它是不会溢出到下一个 line 的。大概就是这样的思路。当然大家发现这个加速比其实不做这件事情就够了。做这件事情在 LM 中会比较抽象是不得已而为之的做法。就是实际上我们做 SMD 的时候就怕的是这个另加法加到下一个令上去。这样他就错了, 但是如果我们支持 i32 加法, 我只给他的第16位填数字, 然后他做加法是不会错的。但这就很危险, 听起来就不太适合刚刚提到这个框架, 所以说一般人也不会这样做, 这只是前期一直没有加速比的时候想着骚操作。最后也没有这样做。

...(01:45:44): 还有什么问题吗? 基本上是他会一开始就求出这个我翻一下。我看看在哪? 在这个地方提到一个 V scale, 这个 V scale 是对每个 CPU 的产量。那么它在做 RVV 的时候, 就会用一个指令去求它的 VL。然后求到这个长度之后, 它就会一直用这个长度。他不会真的用 RVV 那个 VVL 来动态的决定长度。他直接去读 VB, 就根据这个 VB 去做, 剩下的 code 他不会去每个循环用不同的 VVVL。目前的情况就是这样。就是 RVV 你要手写汇编那种很优化的话, 它就是会变的, 但 LV M 它就生成 code 就比较笨, 它不会发生这种事情。比如它 VC 的 VL 时候我们手写汇编会传入这个数组的长度。但是 LM 在 VC 和 VL 的时候, 它会把那个数组长度传一个, 比如说零号寄存器, 然后这样的话, 它就会选择尽量长是这样写的。如果你不知道这个长度, 你就传一个零进去, 我就会选一个尽量长的长度, 那就很符合它现在这个模型。长度的确也不会变。

...(01:47:49): 这个事情其实 RVV 也想做类似于我们把这个东西给它做上去, 但是 LBM 社区在吵架, 你只有 rvv1 家这样的。我要做的终端的话, 别人都不适配了, 就一直合不上去。

...(01:48:19): 对。它对 code size 有优化, 对程序的性能不好说有没有优化, RV 现在有规吗? 他感觉是有优化的, 当然我没有测过。因为他支持不了。类似于当 CD 用的它不会用变长的。也没有白写。

...(01:49:24): 有请香山开发者解答。既然现在的威尔, 所以就不做它的性能了。没有一个。L 里面有加了一个什么玩意儿, 但是反正 in progress 你也不知道他就会写多久。

...(01:50:07): 对还有比如说它生成这个 V 版本, 它会在后面吐一个标量的循环对, 但是我们用变成的话, 那标准选项就可以省掉。那还有什么问题吗?

...(01:50:38): 对怎么。

...(01:50:55): 对是他不会在 BB? 是的。对。对。

...(01:52:20): 对的确是可以的, 这个问题在 LV M 中是这样的! 这就这给大家都讲一下不就是 LM 的这个 BB, 它的 BB 内部其实是有一个顺序的, 就是这 BB 内部它的指令是123456。我们其实是假定这个指令是一条一条执行的, 它这个顺序是严格的。而不是说。

...(01:53:29): 我觉这个舒服, 我就用这个。行。Ok, 我们在 LV MIR 中, 它虽然就是某些指令是没有依赖的, 但是我们在 code 的时候, 我们就生成一个链表, 然后这个链表在生成的时候, 它就决定了这个顺序。

...(01:53:56): 这个顺序是在我们生成那个 code selection dag 的时候是不能去动它的。就是所有的 store 和 load 之间, 它就天然的有一个顺序, 而不是说根据指针分析来决定它到底需不需要顺序。而是在 IR 中, 它就已经有一个顺序, 这个顺序就是 list 是全序的。然后这个序, 你就必须去遵守它。就是 IR 中断的时候就不太好。所以后面的他就会尝试解决这个问题, 就是你刚刚提到 LV M 的这个 BB 在中断的时候就必须是完全有序的。

...(01:54:36): 还有什么问题吗? 也需要的。你说但是你问题你不要快的死的。如果可以去他最需要的就是它其实有足够的。我会做绝对! 然后还有一个问题就是可能漏的会有 exception。然后如果交换顺序, 它 except exception 这地方可能是有问题的。就会有一些妙妙影响。就是我们在编 chrome 的时候上也遇到一些类似这样的问题, 就我们生成 store 的时候, 恰好生成到某个页边界上, 然后它就出它的大。对。

...(01:55:39): 还有什么问题吗? 就其实这样就是假设这是一个配置, 然后这是下一个配置。然后下一个配置是不可读的。那么它现在要非对齐缓存这个位置? 把飞起放在这个位置。我可能就会拆出两条漏的, 因为编辑器不知道它到底那个漏的有没有对齐, 我可能就会拆漏的跟

这然后还漏的这。明白吗。就是它实际上漏的是这个位置，我不知道它对齐了。那我拆成两条路的时候就会拆成一个路的，在这一个路的在这那第二个漏的实际上正确的体系结构会把它解释成一个 L。就 LDR 会是一个 not，我们的体系结构？它是恰好对齐的话，按照语义来说，它是一个 L。他不有问题。因为我们 LDR 的话相当于你这个东西就漏的东西就不缓存。没有什么方法。我觉得你说的可能是对的，我没有试过。因为它在 IR 设计上的时候就会有那条边，所以你不可能交换所有的位置。是快到了。还有什么问题吗？

...(01:57:42): 要不要开一下投影？

...(01:57:46): 线上我刚看了好像没有人提问对。我把评论给关了就不想。

...(01:58:14): 我们可以看到前面那个有个系统软件的地方跑哪去有一个系统软件有个 DLS 这个有没有谁能给我解释一下？他为什么会出问题反问观众，就这个 GBC 它会在？什么 GDP 当然是标量的。没开。它用了它是？

...(01:58:51): 但是 GDC 的这个 DLL 为什么会就是它会涉及到保存所有寄存器，就在别的体系结构上也看到是它会把所有寄存器都 save 一遍再漏出来。就为什么他有没有可能是遵守什么？你说它涉及到函数的时候，它可能就坏了。我为什么不能在外面就是我什么的，那他也不会坏？我要遵守什么 calling 的话，感觉也不会有问题。为什么。

...(01:59:51): 这个 DL 是在 resolve 的时候涉及到的操作就是什么 PLT 一开始进刚刚进去的时候。他会 resolve 符号到底是啥？就感觉不会有的体系结构是会的就是它的浮点跟向量是共用的话，那你去改浮点可能会动到向量。对，就是你动浮点低位，它可能给你把向量高位给搞坏。如它的浮点跟向量是共用的话。反正这的确是我们曾经遇到过一个。他就会在他要是不去故意保存相应寄存器的话。他后面向量计算器就坏了。这个事情只能去通知 G。你编译器也改不了？

...(02:01:17): Ok 大家还有什么问题吗？cc1 下大美。这聊天有人说话。sm da bi. 不是很懂。哈哈。

...(02:01:53): 他说，你必须的。这个 API 反正等我拿到这个东西的时候，他已经存在了，我已经动不了，就我有一个手册，他告诉我这些东西需要存那些东西，不需要存传开走哪个。因为这种东西是一个约定。就是因为比如说人家开发 GCC 的，也需要去遵守这个他可能不会跟你去同步，这些约定是什么？还有什么问题吗？

...(02:02:35): 接近吃饭。

...(02:02:50): Ok 那我们今天就结束吧！感觉不会有人问。