嵌入式处理器与芯片设计基础实验报告
D1 开发板 YOLO 算法优化

陈之逸 21307130003

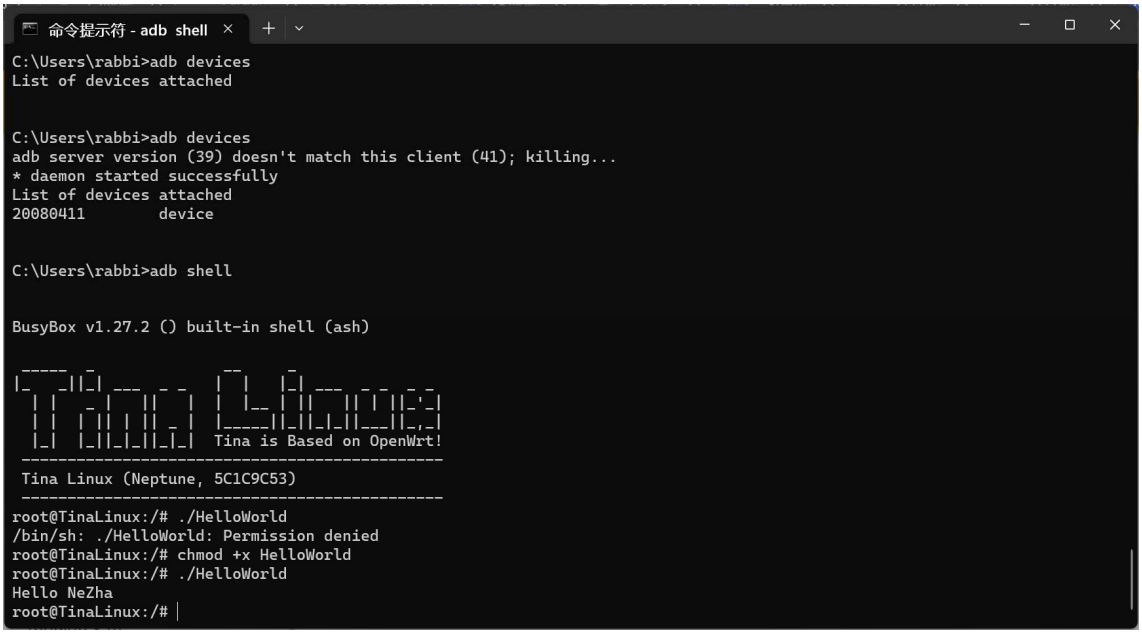
一、 实验目的

熟悉一个 RISC-V 架构的硬件平台，并在之上启动一个 Linux 操作系统。然后在此之上运行 YOLO 算法，之后需要对卷积层进行优化，在此过程会使用嵌入的汇编代码进行性能监测，并对优化的性能进行分析，在这个过程中我们可以把书中学习到的缓存原理知识应用到实际的工程开发中，并真切地感受到其发挥的作用。最后通过 HHB 流程对 RISC-V 的向量扩展的应用有一个初步的认识。

二、 实验步骤（包括实验结果，数据记录、截图等）

1. 将修改过的 Tina Linux 固件烧写到 D1 开发板中

2. 在 D1 开发板上运行 hello_world 的全流程



3. 攥写带有缓存优化版本的卷积函数，并通过 qemu 的测试。（需要写出优化思路，以及优化的原理）

使用的是 gemm 中的代码，优化思路为通过提高代码的执行效率和降低内存访

问开销，从而加快算法的运行速度:

① 引入循环变量 **i, j, k**，用于遍历数组和进行计算。

② 将常量参数存储在局部变量中，减少访存开销。

③ 将内存分配和释放操作移至代码的适当位置，避免重复的内存分配和释放操作。

④ 在进行填充（padding）操作时，使用临时数组 **tmp_pad** 进行存储，避免对输入数组进行频繁的访问和修改。

⑤ 对于填充操作中的边界情况，使用条件判断语句进行处理，将填充部分置为 0。

⑥ 通过优化填充操作，减少了输入数组的访存次数和计算量。

4. 在 D1 开发板上进行三个版本的 YOLO 性能测试, 并攥写分析报告(关于 GEMM 库版本的分析，同学们需要额外查找资料，重点放在 img2col 算法上。下面表格中统计的数据除了“执行时间”，其他的数据可以使用计算量最大的一个层 conv12 来代表。)

	Base	gemm	cache-opt
cycles	48574843412	8102466921	8100080235
instructions	8122896398	5652682178	5652133513
L1D cache read access	1632827323	1607222664	1607089348
L1D cache read miss	270514299	2756249	2758208
L1D cache read hit ratio	83.43%	99.83%	99.83%
L1D cache write access	25232737	803437022	803349651
L1D cache write miss	1049501	272152	269024
L1D cache write hit ratio	95.84%	99.97%	99.97%
执行时间	140.216308s	30.774886s	30.776722s

5. 使用平头哥 HBB 流程编译 mobilenet 生成两个版本的可执行文件（一个带向量扩展，一个不带扩展），并分别在 qemu 和 D1 开发板运行。另行选取三张图片重复上述过程，并将实验结果统计在如下表格中。

表 1 在不同平台下不同输入的执行时间

	qemu_ref	qemu_c906	D1_ref	D1_c906
图片 1	43296.98ms	30272.92ms	54283.68ms	329.92ms
图片 2	53719.39ms	40578.55ms	56849.84ms	488.67ms
图片 3	44987.54ms	28574.11ms	57946.94ms	299.79ms

表 2 在不同平台下不同输入的分类结果

	qemu_ref	qemu_c906	D1_ref	D1_c906
图片 1	263: 0.843262	263: 0.852051	263: 0.843262	263: 0.852539
	264: 0.117798	264: 0.110840	264: 0.117798	264: 0.110107
	248: 0.011482	248: 0.012444	248: 0.011482	248: 0.012070
	249: 0.011482	249: 0.010643	249: 0.011482	249: 0.010483
	250: 0.006699	250: 0.007019	250: 0.006699	250: 0.006916
图片 2	330: 0.755371	330: 0.776855	330: 0.755371	330: 0.773438
	331: 0.197021	331: 0.181763	331: 0.197021	331: 0.183716
	332: 0.032166	332: 0.029190	332: 0.032166	332: 0.029968
	333: 0.008522	333: 0.007725	333: 0.008522	333: 0.007748
	338: 0.003609	338: 0.003252	338: 0.003609	338: 0.003313
图片 3	257: 0.695801	257: 0.700195	257: 0.695801	257: 0.697754
	258: 0.191772	258: 0.191528	258: 0.191772	258: 0.193604
	222: 0.047729	222: 0.049164	222: 0.047729	222: 0.050140
	260: 0.013359	260: 0.012932	260: 0.013359	260: 0.012978
	249: 0.010818	249: 0.009758	249: 0.010818	249: 0.009949

三、 实验分析和总结

在本次实验中，我们在 D1 开发板上进行了 YOLO 算法的运行和软件优化。通过在实验 5 和实验 9 的基础上，我们在 D1 开发板上启动了 Linux 操作系统，并在其上运行了基础版本卷积层实现和原生使用 GEMM 库卷积实现的 YOLO。实验过程中，我们对卷积层进行了手动优化和性能分析，并使用嵌入汇编代码进行性能监测，验证了之前的理论分析结果。

在实验中，我们首先介绍了 D1 开发板的基本信息和架构，包括 D1-H 哪吒开发板的硬件配置和平头哥 RISC-V C906 CPU 的特点。然后，我们详细介绍了如何在 D1 开发板上启动 Linux 操作系统，包括下载 PhoenixSuit 固件烧写工具、安装全志 USB 驱动、连接开发板与电脑、烧写 Linux 固件以及安装 ADB 工具等步骤。

接着,我们讲解了如何在 D1 开发板上运行 Hello World 程序作为简单的测试。通过使用全志提供的平头哥 RISC-V 交叉编译工具链，在 Ubuntu 虚拟机上编写并编译了一个简单的 C 程序，然后使用 qemu 进行功能验证。我们还介绍了将编译好的可执行文件从 Ubuntu 虚拟机导出到 D1 开发板上，并通过 adb 工具将文件上传到开发板并执行的过程。

在运行 YOLO 算法方面，我们提供了 YOLO 的主体程序和相关的 Makefile 文件，并要求同学们在实验 5 的基础上实现缓存优化的卷积层。我们提供了两个版本的卷积层实现，一个是基本版本，另一个是使用 GEMM 库实现的版本。同学们需要根据提供的代码和要求进行优化，并使用 qemu 运行进行验证。之后，将优化后的代码传入 D1 开发板，并在板子上测试运行。

最后，我们介绍了平头哥 HHB 流程，用于了解向量化和部署神经网络模型的工具集。HHB 可以将各种主流的神经网络框架的模型转化为对应的 C 代码或 HHB runtime，并通过 CSI-NN2 库实现算子的向量化。我们提到了 HHB 的基本原理和使用平台，并鼓励有兴趣的同学深入了解官方文档。

四、 实验收获、存在问题、改进措施或建议等

我们通过在 D1 开发板上进行 YOLO 算法的运行与优化，深入了解 YOLO 算法以及在实际硬件平台上的实现和性能优化，同时学习了如何在 D1 开发板上启动 Linux 操作系统，并且掌握了固件烧写工具 PhoenixSuit 的使用方法。

在记录数据的过程中，我们了解了 D1 开发板的硬件配置和平头哥 RISC-V C906 CPU 的特性,包括指令集、性能指标等,也通过运行 Hello World 程序和 YOLO 算法，加深了对平头哥开发板和 RISC-V 架构的理解，并且熟悉了在板上运行程序的流程。

实验非常繁杂，在教程中其实缺少许多实操的步骤，包括替换图片等，做起来实在是头大……或许可以加一些线上的直接的答疑方式。