实验五——乱序处理器

一、 实验目的

1) 探索应用程序在微体系结构改变时具有不同的行为

2) 探索特定体系结构中的瓶颈

3) 提高对乱序处理器体系结构的理解

二、 实验步骤

步骤 I：性能比较

请按照以下要求设置模拟配置参数：

1）分别设置 CPU 模型为 HW3BigCore 和 HW3LittleCore

2）分别设置工作负载为 MatMulWorkload、BFSWorkload 以及 BubbleSortWorkload

步骤 II：中等核心

请按照以下要求设置模拟配置参数：

1）分别设置 CPU 模型为四个 HW3MediumCore

2）选择合适的工作负载来计算加速比

三、 实验结果

（一）模拟前的问题

1)HW3BigCore 相对于 HW3LittleCore 的平均加速比将是多少？你能使用你的流水线参数预测一个上限吗？提示：你可以使用 Amdahl's 定律对速度上限进行预测，使用最优值。

CPU	width	rob_size	num_int_regs	num_fp_regs
LittleCore	4	152	100	84
BigCore	12	352	280	224

Amdahl 定律可以表示为

$$S_{latency} = \frac{1}{(1-p) + \frac{p}{s}}$$

其中，$S_{latency}$ 是整个任务执行的理论加速比，s 是从改进的系统资源中获益的任务部分的加速比，p 是受益于改进资源的任务部分占总执行时间的比例。

假设 p=1，则 $S_{latency}$=s。当流水线宽度 width 从 4 增加到 12 时，加速比为 3。当重排序缓冲区大小 rob_size 从 152 增加到 352 时，加速比为 2.32。由于寄存器数量的增加对于并行度没有直接的影响，因此忽略其对加速比的影响。

故 HW3BigCore 相对于 HW3LittleCore 的平均加速比为 2.66。

2) 你认为所有的工作负载都会在 HW3BigCore 和 HW3LittleCore 之间获得相同的加速比吗？

我认为不同的工作负载会获得不同的加速比，因为不同负载的算法特征不同，对硬件资源的利用率也不相同。

（二）步骤 I

性能指标	仿真时间（ms）			IPC		
工作负载	矩阵	广度优	冒泡	矩阵乘	广度优	冒泡排

	乘法	先搜索	排序	法	先搜索	序
BigCore	11.220	25.917	4.699	0.751345	0.611580	1.340200
LittleCore	11.225	33.319	4.610	0.750983	0.475727	1.366245

1）HW3BigCore 相对于 HW3LittleCore 的加速比是多少？

矩阵乘法加速比：1.000446

广度优先搜索加速比：1.285604

冒泡排序加速比：0.981060

2）HW3BigCore 相对于 HW3LittleCore 的平均 IPC 改善是多少？注意：确保使用正确的平均值报告平均 IPC 改善。

模拟数据	时钟周期数			指令数		
工作负载	矩阵乘法	广度优先搜索	冒泡排序	矩阵乘法	广度优先搜索	冒泡排序
BigCore	22440000	5183400	9398000	16914837	31756325	12650477
LittleCore	22450000	6663800	9220000			

其中，时钟周期数由主频（2GHz）乘以仿真时间（simSeconds）计算得到，指令数由 stats.txt 中的 simInsts 给出。

平均 IPC 可由三个工作负载的指令数之和除以时钟周期数之和得到。因此，HW3BigCore 的平均 IPC 为 0.997807，HW3LittleCore 的平均 IPC 为 0.623771。

HW3BigCore 相对于 HW3LittleCore 的平均 IPC 改善 59.96%。

3）一些工作负载显示出更多的加速比。哪些工作负载显示出较高的加速比，哪些显示出较低的加速比？查看基准代码（.c 和.s 文件都

可能有用），并推测影响 HW3BigCore 和 HW3LittleCore 之间 IPC 差异的算法特性。哪些特性会导致性能改善较低，哪些特性会导致性能改善较高？

广度优先搜索显示出较高加速比，冒泡排序显示出较低加速比。

BigCore 和 LittleCore 之间 IPC 的差异可能受到以下算法特性的影响：

①数据依赖性和并行性：如果算法中存在较多的数据依赖关系，即后续指令需要等待前面的指令完成才能执行，这可能导致指令级并行度较低。在这种情况下，BigCore 和 LittleCore 之间的 IPC 差异可能较小。

②访存模式：如果算法具有大量的内存访问操作，并且存在较多的内存相关延迟，那么 BigCore 和 LittleCore 之间的 IPC 差异可能会增大。这是因为 BigCore 通常具有更大的缓存容量和更高的内存带宽，可以更好地处理内存访问。

③分支预测：如果算法中存在大量的分支指令（例如条件语句、循环等），并且分支预测准确率较低，这可能导致指令流水线的中断和清空，使得 BigCore 和 LittleCore 之间的 IPC 差异不明显。

4）哪个工作负载对于 HW3BigCore 具有最高的 IPC？这个工作负载有什么独特之处？提示：查看 ROI 的汇编代码以获得灵感。

冒泡排序对于 HW3BigCore 具有最高的 IPC。

在乱序处理器中，指令级并行的实现是通过指令重排序来实现的。指令重排序允许乱序处理器对指令进行重新排序，以充分利用执行单

元的资源。由于冒泡排序算法简单，指令之间的依赖关系较少，乱序处理器可以更好地对指令进行重排序，从而提高整体的指令执行效率，即提高 IPC。相比之下，矩阵乘法和广度优先搜索是更复杂的算法，它们涉及到更多的指令和数据依赖关系。矩阵乘法涉及大量的乘法和加法操作，而广度优先搜索需要维护一个队列和进行大量的节点遍历。因此，这两种算法无法充分发挥乱序处理器的优势。

（三）步骤 II

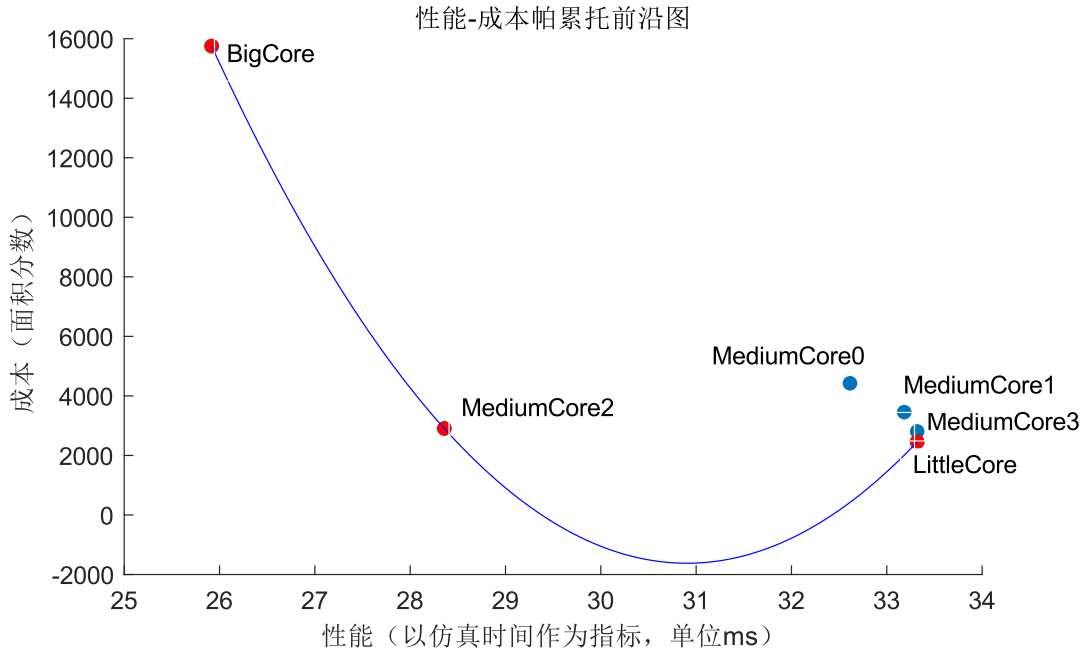
1）如果你只能在之前使用的 3 个工作负载中选择一个来计算加速比，你会选择哪个工作负载？为什么？

应选择广度优先搜索（BFS）作为工作负载来计算加速比。因为 LittleCore 和 BigCore 在矩阵乘法和冒泡排序这两个工作负载上的仿真时间相差较小，在设计 MediumCore 时，性能差距体现不明显。而在广度优先搜索这个工作负载上，LittleCore 和 BigCore 的仿真时间相差较大，有助于衡量 MediumCore 的性能提升。

2）现在我们已经制定了测量成本和收益的函数，为流水线配置 4 个中间设计。这些设计中并不是所有的设计都有“最佳”成本-收益权衡。在你的报告中包括以下图。创建一个带有成本在 y 轴上和性能在 x 轴上的帕累托前沿图。这将是一个散点图，有 6 个点：两个“Big”和“Little”核心，以及你的 4 个中间设计。然后，在“最佳”设计上“连接这些点”。

CPU	width	rob_size	num_int_regs	num_fp_regs	面积分数	仿真时间（ms）

Little	4	152	100	84	2456	33.319
Medium0	8	152	100	84	4424	32.614
Medium1	4	252	100	84	3456	33.183
Medium2	4	152	190	84	2906	28.358
Medium3	4	152	100	154	2806	33.319
Big	12	352	280	224	15752	25.917



3）假设你是一名工程师，正在设计这个中间的核心。根据这次早期分析，你会建议你的团队追求开发哪些设计，如果有的话？解释原因。（注意：你可能需要在上面的图中注释）

建议在 LittleCore 的基础上增加整数寄存器的数量。因为增加 90%的整数寄存器后，在 BFS 工作负载上，相对于 LittleCore 的加速比为 1.174942，达到了 BigCore 性能的 91.39%，但成本只增加了 18.32%。

四、 实验心得体会

通过本次实验，我对乱序处理器体系结构有了更深的理解，也学会了如何分析体系结构的性能瓶颈，并据此对CPU的硬件资源进行改善，从而实现性能与成本的平衡。