西安交通大学
XI'AN JIAOTONG UNIVERSITY

《计算机系统综合设计实验》

计算机科学与技术学院 实验中心

2023年10月

西安交通大学
XI'AN JIAOTONG UNIVERSITY

实验五、乱序处理器

实验五 乱序处理器

实验内容

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）提高你对乱序处理器体系结构的理解。

实验内容

工作负载 -矩阵乘法

本次实验使用 DAXPY 作为工作负载。DAXPY 循环（双精度 aX + Y）是在处理矩阵和向量的程序中经常使用的操作。以下是 C++ 代码实现的 DAXPY。

```
void multiply(double* A, double* B, double* C, int size)
{
    for (int i = 0; i < size; i++) {
        for (int k = 0; k < size; k++) {
            for (int j = 0; j < size; j++) {
                C[i * size + j] += A[i * size + k] * B[k * size + j];
            }
        }
    }
}
```

实验内容

工作负载 -广度优先搜索（BFS）

```
#include <iostream>
#include <vector>
#include "graph.h"
#ifdef GEM5
#include "gem5/m5ops.h"
#endif
int main()
{
    std::vector<int> frontier;
    std::vector<int> next;
    frontier.clear();
    next.clear();
    frontier.push_back(0);
    std::cout << "Beginning BFS ..." << std::endl;
#ifdef GEM5
    m5_work_begin(0,0);
#endif
```

查看 workloads/bfs_workload.py 以了解有关实例化BFS工作负载的信息。

```
while (!frontier.empty()) {
    for (auto vertex: frontier) {
        int start = columns[vertex];
        int end = columns[vertex + 1];
        for (int i = start; i < end; i++){
            int neighbor = edges[i];
            if (visited[neighbor] == 0) {
                visited[neighbor] = 1;
                next.push_back(neighbor);
            }
        }
    }
    frontier = next;
    next.clear();
}

#ifdef GEM5
    m5_work_end(0,0);
#endif

    std::cout << "Finished BFS." << std::endl;
    return 0;
}
```

实验内容

工作负载 -冒泡排序

```
#include <iostream>

#include "array.h"

#ifdef GEM5
#include "gem5/m5ops.h"
#endif

int main()
{
   std::cout << "Beginning bubble sort ... " << std::endl;

#ifdef GEM5
   m5_work_begin(0,0);
#endif
```

```
for (int i = 0; i < ARRAY_SIZE - 1; i++) {
      for (int j = i + 1; j < ARRAY_SIZE; j++) {
         if (data[i] > data[j]) {
            int temp = data[i];
            data[i] = data[j];
            data[j] = temp;
         }
      }
   }

#ifdef GEM5
   m5_work_end(0,0);
#endif

   std::cout << "Finished bubble sort." << std::endl;

   return 0;
}
```

实验内容

实验设置

- 先将gem-assighment-template切换到assign-3

```
git checkout assign-3
```

- 在这个实验中，你需要为实验设计自己的乱序处理器模型。系统的其余组件说明如下：
  1）你需要使用HW3RISCVBoard作为计算机系统的主板。你可以在components/boards.py中找到该主板的模型。

  2）你需要使用HW3MESICache作为计算机系统的缓存层次结构。你可以在components/cache_hierarchies.py中找到它的模型。

  3）你需要使用HW3DDR4作为计算机系统中的内存。你可以在components/memories.py中找到它的模型。

  4）你需要使用2 GHz作为系统中的时钟频率clk_freq。

  5） 对于处理器，你需要使用HW3O3CPU来分别模拟一个高性能和高效能的处理器核心。HW3O3CPU基于O3CPU，它是gem5的内部模型。阅读gem5文档中关于O3CPU的内容。

实验设置-CPU参数

■ 流水线宽度

为了完成本次实验，你需要配置一个在所有阶段中具有相同宽度的处理器。流水线宽度表示在流水线中同时处理的指令数量，通常以“级”（stages）为单位。每个级代表指令执行的一个阶段。例如，一个4级流水线可能包括取指（IF）、译码（ID）、执行（EX）、写回（WB）等阶段。在HW3O3CPU的构造函数中，此属性命名为width。

■ 重排序缓冲区大小

- ✓ 重排序缓冲区（ROB）的大小是指其能够容纳的指令条目数量。这个大小在处理器设计中是一个重要的参数，直接影响处理器的性能和能力。

- ✓ ROB的大小决定了处理器能够同时保持和调度的指令数量。较大的ROB允许更多的指令在飞行中（in flight）进行执行，从而提高指令级并行性（ILP），充分利用处理器中的各个执行单元，提高性能。然而，较大的ROB也会占用更多的硬件资源，可能增加成本和功耗。在HW3O3CPU的构造函数中，此属性命名为rob_size。

实验设置-CPU参数

- 物理寄存器数量
  - ✓ 指物理寄存器组中的寄存器数量。处理器将架构寄存器重命名为物理寄存器以解决伪依赖。它还在寄存器组中跟踪真实依赖（写后读）。要了解有关寄存器重命名的更多信息，请阅读Tomasulo算法。
  - ✓ HW3O3CPU具有两个物理寄存器文件。一个用于整数寄存器，另一个用于浮点寄存器。在HW3O3CPU的构造函数中，num_int_regs是整数物理寄存器的数量，num_fp_regs是浮点物理寄存器的数量。
  - ✓ 注意：两个寄存器组都必须大于32个条目，否则gem5将会挂起。这是因为物理寄存器的数量必须大于或等于逻辑寄存器的数量。RISC-V ISA定义了32个逻辑寄存器。

实验内容

实验设置-CPU参数

- 大核心和小核心
- 在这次实验中，你需要设计自己的高性能和高效处理器核心。你需要在components/processors.py中添加两个核心设计。在components/processors.py中，创建一个基于HW3O3CPU的模型，并命名为HW3BigCore。这个核心将是你在任务中的高性能核心。在components/processors.py中创建另一个模型，并命名为HW3LittleCore。这个核心将是你在任务中的高效核心。你可以使用互联网上关于不同核心微体系结构的信息来配置你的HW3BigCore和HW3LittleCore。作为参考，查看WikiChip和AnandTech上的信息。HW3BigCore基于Intel Sunny Cove，而HW3LittleCore基于Intel Gracemont。
- 注意：仅使用网上处理器型号信息作为设计的指导，而不是严格匹配其规格。你可能会发现很难匹配网上找到的处理器规格。例如，基本模型HW3O3CPU假定流水线的所有阶段都具有相同的宽度，而这在现代处理器设计中通常是不确切的。

可供参考：big core

width = 12　　rob_size=352　　num_int_regs=280　　num_fp_regs=224

little core

width = 4　　rob_size=152　　num_int_regs=100　　num_fp_regs=84

分析与模拟：步骤 I 性能比较

- 在运行模拟之前，在你的报告中回答以下问题。

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。

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？

实验内容

分析与模拟：步骤 I 性能比较

- 既然你已经完成了HW3BigCore和HW3LittleCore的设计过程，让我们使用三个工作负载作为基准来比较它们的性能。用每个核心运行每个工作负载。对于每个工作负载，比较HW3BigCore的性能与HW3LittleCore的性能。

- 在你的报告中回答以下问题，使用模拟数据进行合理的推理。。

1 HW3BigCore相对于HW3LittleCore的加速比是多少？
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。
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？查看测试代码（.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？
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（Region of Interest）的汇编代码以获得灵感。

分析与模拟：步骤 II：中等核心 I

- 在这一步中，你的任务是在HW3BigCore和HW3LittleCore之间找到一个折中。这个核心需要在尽量使用HW3LittleCore资源的同时尽量接近HW3BigCore的性能。我们将称这个核心为HW3MediumCore。

- 为了寻找HW3MediumCore设计的最优值，我们需要设计一种方法。首先，我们需要为增加硬件资源定义一个成本函数。我们将使用面积作为制作硬件的成本。流水线的所有参数对面积的影响并不相同。例如，增加流水线的宽度对硬件面积有平方增长的影响，而增加寄存器组条目则对硬件面积有线性影响。此外，同时增加这两个资源的成本应该大于单独增加每个资源的和。我们将使用以下公式来为流水线面积进行评分：

$$area_{score} = 2 * width^2 * rob\_size + width^2 * (num\_int\_regs + num\_fp\_regs) + 4 * width + 2 * rob\_size + (num\_int\_regs + num\_fp\_regs)$$

你还可以通过在处理器上调用get_area_score方法来获取流水线设计的面积分数。

实验内容

分析与模拟：步骤 II：中等核心 I

- 现在我们有了成本函数，让我们制定一种测量收益的方法。在你的报告中回答以下问题。

- 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？

- 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个中等核心设计。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个中等核心设计。然后，在“最佳”设计上“连接这些点”。

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

实验内容

提交

- 在报告中回答步骤1、2的问题。使用清晰的推理和可视化来支持你的结论。 请确保包括以下代码/脚本。

- Instruction.md：包含有关如何运行你的模拟的说明。
- Automation：运行模拟的代码/脚本。
- Configuration：配置你需要模拟的系统的python文件。你应该将最终的核心设计添加到components/processors.py中。在components/processor.py中应包含6个核心定义。它们应该包括1个用于HW3BigCore的设计，1个用于HW3LittleCore的设计，以及4个用于HW3MediumCore的设计。你可以为HW3MediumCore的设计添加数字以区分它们的设计。例如，HW3MediumCore0、HW3MediumCore1、HW3MediumCore2和HW3MediumCore3。

谢谢！