Lab2 实验报告

修改配置

1. 基础默认配置
    a. CPU: DerivO3CPU 使用se.py的选项--cpu-type配置
    b. Compiler:gcc -O3 --ffast-math -ftree-vectorive 修改Makefile文件配置
    c. Frequency: 1Ghz 使用se.py的--cpu-clock 配置
    d. Memory: DDR3_1600_8x8 使用se.py的--memory-type 配置
    e. 2-level cache hierarchy (64KB L1 icache, 64KB L1 dcache, 2MB L2) 使用se.py 的--l1isize,--l1dsize,--caches,--l2size,--l2cache配置。
    f. Issue Width=8 修改se.py源文件进行修改，在合适位置加入system.cpu[i].issueWidth=8

makefile 核心修改如下：

```
CC=gcc
OPT = -O3

CFLAGS   = $(OPT) -ffast-math -ftree-vectorize

EXEC=lfsr merge mm sieve spmv

all: $(EXEC)

%: %.c
    $(CC) -o $@ $< $(CFLAGS)
```

se.py 默认配置：

```
GEM5_BASE=~/gem5
GEM5_OPT=~/gem5/build/X86/gem5.opt
SRC_DIR=~/lab2/cs251a-microbench-master
RESULT_DIR=~/lab2/result
SE1_PY=~/gem5/configs/example/se.py
SE1_PY=~/gem5/configs/example/se1.py
SE2_PY=~/gem5/configs/example/se2.py
```

```
${GEM5_OPT} ${SE1_PY} --cmd=${SRC_DIR}/${FILE} --cpu-type=DerivO3CPU \
    --l1d_size=64kB --l1i_size=64kB --caches \
    --sys-clock=1GHz --cpu-clock=1GHz --mem-type=DDR3_1600_8x8
```

IssueWidth=8为se1.py,IssueWidth=2为se2.py

```
lumos@gem:~/lab2$ cat ~/gem5/configs/example/se1.py |grep issue
    system.cpu[i].issueWidth=8
lumos@gem:~/lab2$ cat ~/gem5/configs/example/se2.py |grep issue
    system.cpu[i].issueWidth=2
```

2. 配置列表

编写shell文件，测试各种配置。

Bash | 复制代码

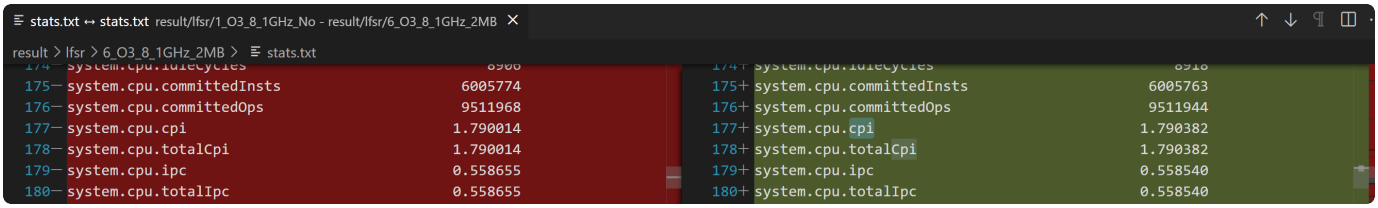
```bash
#!/bin/bash

GEM5_BASE=~/gem5
GEM5_OPT=~/gem5/build/X86/gem5.opt
SRC_DIR=~/lab2/cs251a-microbench-master
RESULT_DIR=~/lab2/result
SE1_PY=~/gem5/configs/example/se.py
SE1_PY=~/gem5/configs/example/se1.py
SE2_PY=~/gem5/configs/example/se2.py

M5OUT=m5out
OUT_TXT=${M5OUT}/stats.txt
OUT_INI=${M5OUT}/config.ini
rm -rf ${RESULT_DIR}/*
for FILE in mm lfsr merge sieve spmv; do
    mkdir -p ${RESULT_DIR}/${FILE}
done

for FILE in mm lfsr merge sieve spmv; do
    #O3 8 1GHz No
    ${GEM5_OPT} ${SE1_PY} --cmd=${SRC_DIR}/${FILE} --cpu-type=DerivO3CPU \
        --l1d_size=64kB --l1i_size=64kB --caches \
        --sys-clock=1GHz --cpu-clock=1GHz --mem-type=DDR3_1600_8x8
    mkdir ${RESULT_DIR}/${FILE}/1_O3_8_1GHz_No
    cp  ${OUT_TXT} ${OUT_INI} ${RESULT_DIR}/${FILE}/1_O3_8_1GHz_No

    #MinorCPU 8 1GHz No
    ${GEM5_OPT} ${SE_PY} --cmd=${SRC_DIR}/${FILE} --cpu-type=MinorCPU \
        --l1d_size=64kB --l1i_size=64kB --caches \
        --sys-clock=1GHz --cpu-clock=1GHz --mem-type=DDR3_1600_8x8
    mkdir ${RESULT_DIR}/${FILE}/2_Mi_8_1GHz_No
    cp ${OUT_TXT} ${OUT_INI} ${RESULT_DIR}/${FILE}/2_Mi_8_1GHz_No

    #O3 2 1GHz No
    ${GEM5_OPT} ${SE2_PY} --cmd=${SRC_DIR}/${FILE} --cpu-type=DerivO3CPU \
        --l1d_size=64kB --l1i_size=64kB --caches \
        --sys-clock=1GHz --cpu-clock=1GHz --mem-type=DDR3_1600_8x8
    mkdir ${RESULT_DIR}/${FILE}/3_O3_2_1GHz_No
    cp ${OUT_TXT} ${OUT_INI} ${RESULT_DIR}/${FILE}/3_O3_2_1GHz_No

    #O3 8 4GHz No
    ${GEM5_OPT} ${SE1_PY} --cmd=${SRC_DIR}/${FILE} --cpu-type=DerivO3CPU \
        --l1d_size=64kB --l1i_size=64kB --caches \
        --sys-clock=4GHz --cpu-clock=4GHz --mem-type=DDR3_1600_8x8
    mkdir ${RESULT_DIR}/${FILE}/4_O3_8_4GHz_No
```

```
    cp ${OUT_TXT} ${OUT_INI} ${RESULT_DIR}/${FILE}/4_O3_8_4GHz_No

    #O3 8 1GHz 256kB
    ${GEM5_OPT} ${SE1_PY} --cmd=${SRC_DIR}/${FILE} --cpu-type=DerivO3CPU \
        --l1d_size=64kB --l1i_size=64kB --caches --l2_size=256kB --l2cache \
        --sys-clock=1GHz --cpu-clock=1GHz --mem-type=DDR3_1600_8x8
    mkdir ${RESULT_DIR}/${FILE}/5_O3_8_1GHz_256kB
    cp ${OUT_TXT} ${OUT_INI} ${RESULT_DIR}/${FILE}/5_O3_8_1GHz_256kB

    #O3 8 1GHz 2MB
    ${GEM5_OPT} ${SE1_PY} --cmd=${SRC_DIR}/${FILE} --cpu-type=DerivO3CPU \
        --l1d_size=64kB --l1i_size=64kB --caches --l2_size=2MB --l2cache \
        --sys-clock=1GHz --cpu-clock=1GHz --mem-type=DDR3_1600_8x8
    mkdir ${RESULT_DIR}/${FILE}/6_O3_8_1GHz_2MB
    cp ${OUT_TXT} ${OUT_INI} ${RESULT_DIR}/${FILE}/6_O3_8_1GHz_2MB

    #O3 8 1GHz 16MB
    ${GEM5_OPT} ${SE1_PY} --cmd=${SRC_DIR}/${FILE} --cpu-type=DerivO3CPU \
        --l1d_size=64kB --l1i_size=64kB --caches --l2_size=16MB --l2cache \
        --sys-clock=1GHz --cpu-clock=1GHz --mem-type=DDR3_1600_8x8
    mkdir ${RESULT_DIR}/${FILE}/7_O3_8_1GHz_16MB
    cp  ${OUT_TXT} ${OUT_INI} ${RESULT_DIR}/${FILE}/7_O3_8_1GHz_16MB
done
```

结果分析

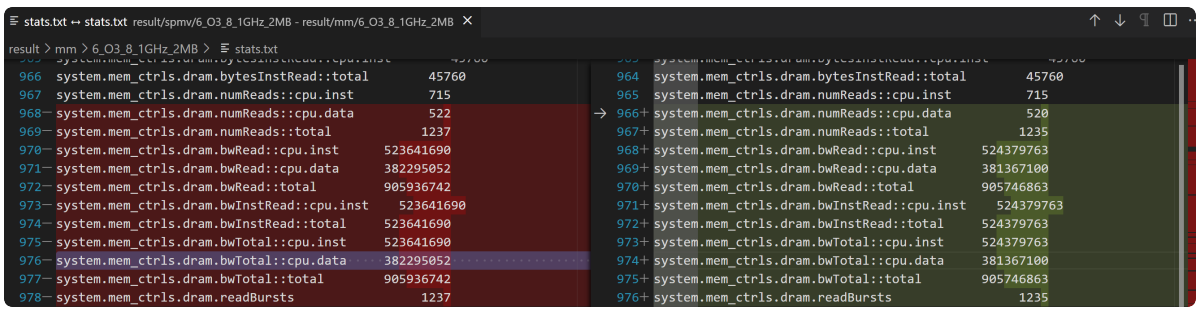
1. 应该使用什么指标来比较不同系统配置之间的性能？为什么？
    a. CPI：`system.cpu.cpi` 每条指令的平均时钟周期数。CPI较低表示该系统执行指令的效率更高。
    b. IPC：`system.cpu.ipc` 每个时钟周期内执行的指令数。IPC较高表示该系统具有更高的并行度和吞吐量。
2. 是否有任何基准测试受益于删除 L2 缓存？请说明理由。

一般情况下，L2缓存可以提高存储器系统的效率，减少处理器与主存之间的通信次数，但特殊情况下删除L2缓存的情况性能更佳。

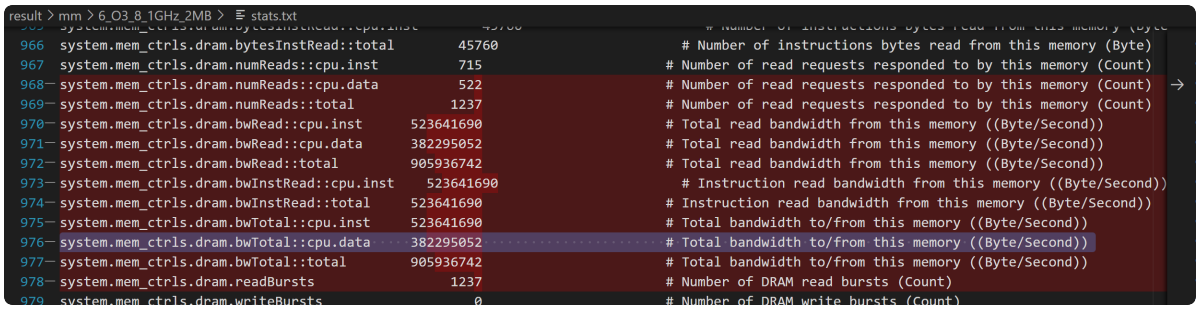


原因：lfsr.c程序访问的数据量较少且不需要频繁从内存中读取，此时L2缓存的优势将不在明显，甚至由于结构简单，没有L2缓存的设计性能更好，当也不会好很多。如上图所示，没有L2缓存的CPI=1.790014，而L2缓存=2MB的CPI=1.790382，性能反而更低，但也低的不明显

3. 在讨论程序的运行行为时，我们会遇到a) memory regularity，b) control regularity，和 c) memory locality，请谈一谈你对他们的理解。
    a. memory regularity: 内存规律性。指程序中对内存的访问是否具有连续性和规律性，即内存地址是否能够以一种预测的方式被访问。其典型例子是数组访问，如果数组元素的存储是连续的，那么内存规律性就比较好。由于现代计算机在处理器和存储设备之间加入了缓存层，因此内存规律性也会对缓存的局部性产生影响
    b. control regularity：控制规律性。指程序代码流的规律性，即程序中的分支、跳转、循环等逻辑结构。控制规律性高的程序可以更容易地被处理器预测，从而减少分支误判和访问跳转地址所需的开销。
    c. memory locality：内存局部性。指的是程序在执行期间访问内存的趋势，特别是一个时间段内执行过的指令或数据在未来也可能被再次访问的概率。按照时间和空间分为时间局部性和空间局部性。具有良好的内存局部性的程序可以最小化因应用程序需要而进行的主存访问，从而提高计算机的效率。
4. 对于这三个程序属性——a) memory regularity，b) control regularity，和 c) memory locality——从 stats.txt 中举出一个统计指标（或统计指标的组合），通过该指标你可以区分一个workload是否具有上述的某一个属性。 （例如，对于control regularity，它与分支指令的数量成反比。但你一定可以想到一个更好的）。
    a. 内存规律性
        i. 与内存带宽成正比。内存带宽指单位时间内能够传输的内存数据量大小。
        ii. 使用stats.txt中的 `system.mem_ctrls.dram.bwTotal::cpu.data # Total bandwidth to/from this memory ((Byte/Second))` 作为衡量指标。
        iii. 如下图，spmv程序的内存规律性优于mm程序



iv. 上面出现的结果参数的注释如下：

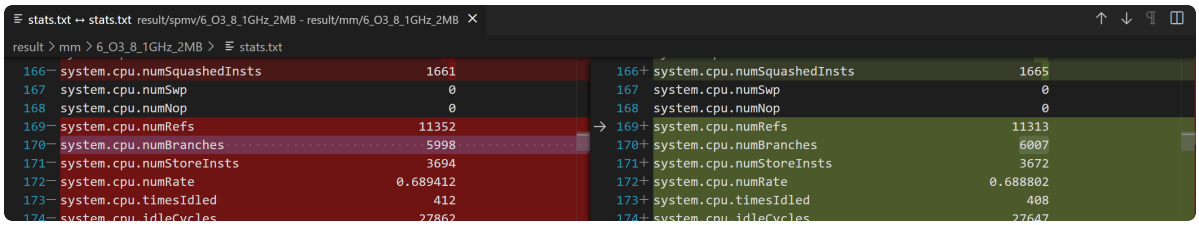


b. 控制规律性

i. 与分支指令的数量成反比

ii. 使用stats.txt中的 `system.cpu.numBranches  # Number of branches executed (Count)` 作为指标

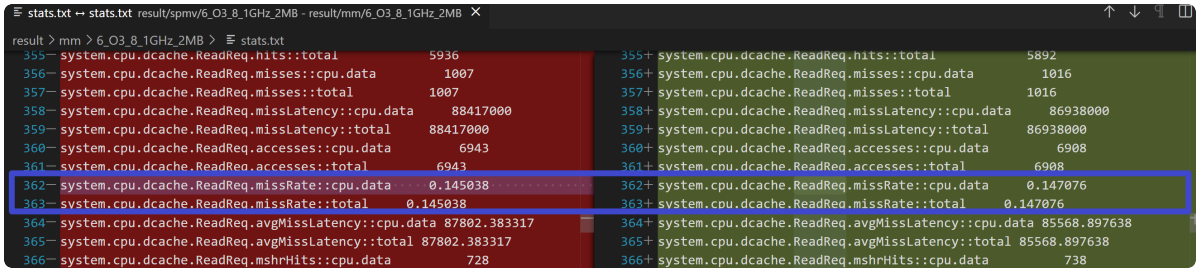
iii. 如下图所示，spmv程序的控制规律性劣于mm程序

iv. 

a. 内存局部性。

i. 与dcache的缺失率成反比。

ii. 使用stats.txt中的 `system.cpu.dcache.ReadReq.missRate::cpu.data # miss rate for ReadReq accesses (Ratio)` 作为指标

iii. 

iv. 如上图所示spmv的内存局部性优于mm程序

5. 对于每一个实验中用到的benchmark，描述它的a) memory regularity, b) control regularity, c) locality；解释该benchmark对哪个微架构参数最敏感（换句话说，你认为“瓶颈”是什么），并使用推理或统计数据来证明其合理性。

使用如下shell命令获取各个实验的部分参数，具体参数见上所述。

```bash
> stats.txt
for FILE in mm lfsr merge sieve spmv; do
    for DIR in 1_O3_8_1GHz_No 2_Mi_8_1GHz_No 3_O3_2_1GHz_No 4_O3_8_4GHz_No 5_O3_8_1GHz_256kB 6_O3_8_1GHz_2MB 7_O3_8_1GHz_16MB ;do
        cat ./result/${FILE}/${DIR}/stats.txt | grep system.mem_ctrls.dram.bwTotal::cpu.data >> stats.txt
        cat ./result/${FILE}/${DIR}/stats.txt | grep system.cpu.numBranches >> stats.txt
        cat ./result/${FILE}/${DIR}/stats.txt | grep system.cpu.dcache.ReadReq.missRate::cpu.data >> stats.txt
    done
done

cat stats.txt | awk '{print $2}'|xargs -n3;
```

结果整理如下：

		内存规律性（正比）	控制规律性（反比）	内存局部性（反比）	CPI
mm	1	571322500	5946	0.145055	2.997471
	2	571322500	5946	0.145055	2.997471
	3	526547498	5202	0.172273	3.240061
	4	886942311	6003	0.15328	7.737785
	5	381367100	6006	0.147076	4.414062
	6	381367100	6006	0.147076	4.414062
	7	381367100	6006	0.147076	4.414062

lfsr	1	572762646	5978	0.145059	2.989934
	2	572762646	5978	0.145059	2.989934
	3	526563939	5197	0.171352	3.23996
	4	889005373	6001	0.152848	7.719828
	5	381367100	6003	0.146829	4.414062
	6	381367100	6003	0.146829	4.414062
	7	381367100	6003	0.146829	4.414062
merge	1	571322500	5946	0.145055	2.997471
	2	571322500	5946	0.145055	2.997471
	3	526547498	5202	0.172273	3.240061
	4	886942311	6003	0.15328	7.737785
	5	381367100	6006	0.147055	4.414062
	6	381367100	6006	0.147055	4.414062
	7	381367100	6006	0.147055	4.414062
sieve	1	568845876	5945	0.146523	3.024903

	2	568845876	5945	0.146523	3.024903
	3	528027910	5164	0.172253	3.234092
	4	888713890	5968	0.154397	7.744613
	5	381597620	5964	0.149464	4.440994
	6	381597620	5964	0.149464	4.440994
	7	381597620	5964	0.149464	4.440994
spmv	1	568731207	5985	0.142449	3.006718
	2	568731207	5985	0.142449	3.006718
	3	526090422	5203	0.169758	3.231981
	4	885194778	5981	0.15	7.741704
	5	382295052	5997	0.145038	4.413809
	6	382295052	5997	0.145038	4.413809
	7	382295052	5997	0.145038	4.413809

分析方式：找出同一个benchmark中性能最好和最差的两种架构，比较内存规律性，控制规律性及控制规律性的相对差值（相对误差）。在结合其它数据整体分析。

由上表可知各个benchmark的瓶颈为：

- mm：内存局部性。因为当内存局部性提高时，性能提高显著（CPI降低）
- lfsr: 内存局部性。该程序主要受内存局部性影响
- merge: 控制规律性。该程序性能受控制规律性影响较大
- sieve:受各方面影响比较均衡，没有明显的内存规律性和控制规律性，但相比之下内存规律性影响较大
- spmv:内存规律性。性能主要受内存规律性影响。

6. 选择一个benchmark，提出一种你认为对该benchmark非常有效的应用程序增强、ISA 增强和微体系结构增强。(即，在应用层面、ISA层面、微体系结构层面对该benchmark进行改进)。

选择mm程序。

- 应用层面：调整循环顺序，使连续访问的地址在内存中是连续的，提高内存规律性。将矩阵分块计算。
- ISA层面：设计指令并行化的ISA架构。使用向量化指令集可以以单一指令处理器同时计算多个数据元素，可以大大提高矩阵乘法效率
- 微体系结构：使用预取机制，即访问某一地址时，连同附近的一些地址一起访问。采用超标量技术，允许多个指令同时执行。