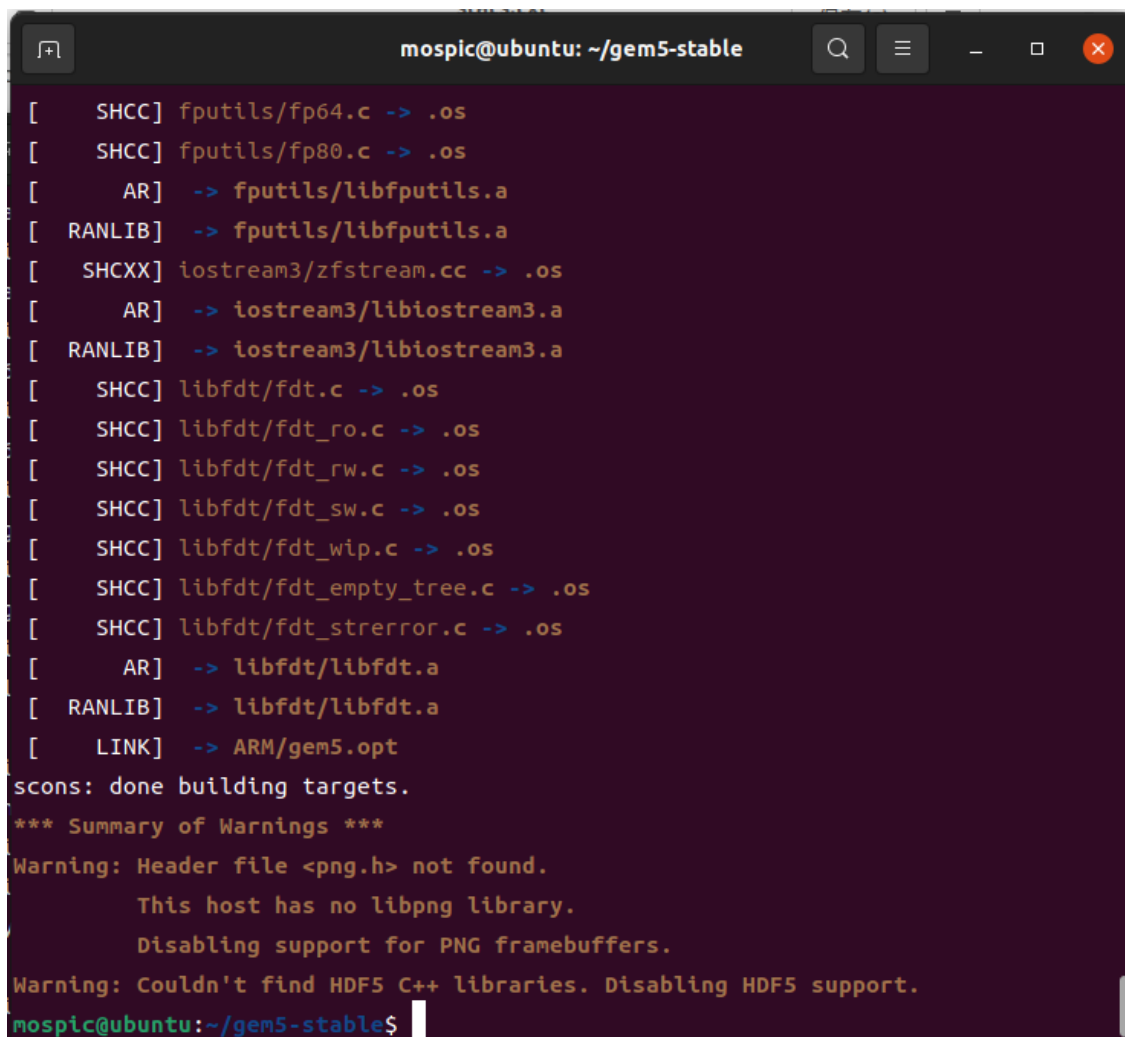


CALab4_Report

PB19051183 吴承泽

实验过程

1. 安装新的gem5文件，执行命令"python3 which sconsbld/ARM/gem5.opt"编译ARM版GEM5，编译完如下所示。



```
[ SHCC] fputils/fp64.c -> .os
[ SHCC] fputils/fp80.c -> .os
[ AR] -> fputils/libfputils.a
[ RANLIB] -> fputils/libfputils.a
[ SHCXX] iostream3/zfstream.cc -> .os
[ AR] -> iostream3/libiostream3.a
[ RANLIB] -> iostream3/libiostream3.a
[ SHCC] libfdt/fdt.c -> .os
[ SHCC] libfdt/fdt_ro.c -> .os
[ SHCC] libfdt/fdt_rw.c -> .os
[ SHCC] libfdt/fdt_sw.c -> .os
[ SHCC] libfdt/fdt_wip.c -> .os
[ SHCC] libfdt/fdt_empty_tree.c -> .os
[ SHCC] libfdt/fdt_strerror.c -> .os
[ AR] -> libfdt/libfdt.a
[ RANLIB] -> libfdt/libfdt.a
[ LINK] -> ARM/gem5.opt
scons: done building targets.
*** Summary of Warnings ***
Warning: Header file <png.h> not found.
        This host has no libpng library.
        Disabling support for PNG framebuffers.
Warning: Couldn't find HDF5 C++ libraries. Disabling HDF5 support.
mospic@ubuntu:~/gem5-stable$
```

2. 修改提供的MakeFile路径如下（修改为O0）：

```
CC=/home/mospic/gem5-stable/daxpy+gcc/gcc-linaro-6.4.1-2017.11-
x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu-g++

daxpy: daxpy.cc
    $(CC) -Iinclude --std=c++11 -static daxpy.cc util/m5/m5op_arm_A64.S -o
daxpy -O1
```

编译得到可执行文件。

3. 执行命令 `bld/ARM/gem5.opt configs/example/arm/starter_se.py`
`daxpy+gcc/daxpy/daxpy --cpu=hpi` 执行样例程序，执行得到结果如下：

```
mospic@ubuntu: ~/gem5-stable
1Hash
build/ARM/cpu/minor/execute.cc:170: warn: No functional unit for OpClass SimdSha
1Hash2
build/ARM/cpu/minor/execute.cc:170: warn: No functional unit for OpClass SimdSha
256Hash
build/ARM/cpu/minor/execute.cc:170: warn: No functional unit for OpClass SimdSha
256Hash2
build/ARM/cpu/minor/execute.cc:170: warn: No functional unit for OpClass SimdSha
Sigma2
build/ARM/cpu/minor/execute.cc:170: warn: No functional unit for OpClass SimdSha
Sigma3
build/ARM/cpu/minor/execute.cc:170: warn: No functional unit for OpClass SimdPre
dAlu
0: system.remote_gdb: listening for remote gdb on port 7000
build/ARM/sim/simulate.cc:194: info: Entering event queue @ 0. Starting simulat
ion...
build/ARM/sim/mem_state.cc:443: info: Increasing stack size by one page.
build/ARM/sim/syscall_emul.hh:1014: warn: readlink() called on '/proc/self/exe'
may yield unexpected results in various settings.
    Returning '/home/mospic/gem5-stable/daxpy'
build/ARM/sim/mem_state.cc:443: info: Increasing stack size by one page.
958601.094721
exiting with last active thread context @ 47757235000
mospic@ubuntu:~/gem5-stable$
```

程序执行结果为正确的。

4. 重写函数后，得到的执行结果如下所示：

```
mospic@ubuntu: ~/gem5-stable
1Hash
build/ARM/cpu/minor/execute.cc:170: warn: No functional unit for OpClass SimdSha
1Hash2
build/ARM/cpu/minor/execute.cc:170: warn: No functional unit for OpClass SimdSha
256Hash
build/ARM/cpu/minor/execute.cc:170: warn: No functional unit for OpClass SimdSha
256Hash2
build/ARM/cpu/minor/execute.cc:170: warn: No functional unit for OpClass SimdSha
Sigma2
build/ARM/cpu/minor/execute.cc:170: warn: No functional unit for OpClass SimdSha
Sigma3
build/ARM/cpu/minor/execute.cc:170: warn: No functional unit for OpClass SimdPre
dAlu
0: system.remote_gdb: listening for remote gdb on port 7000
build/ARM/sim/simulate.cc:194: info: Entering event queue @ 0. Starting simulat
ion...
build/ARM/sim/mem_state.cc:443: info: Increasing stack size by one page.
build/ARM/sim/syscall_emul.hh:1014: warn: readlink() called on '/proc/self/exe'
may yield unexpected results in various settings.
    Returning '/home/mospic/gem5-stable/daxpy+gcc/daxpy/daxpy'
build/ARM/sim/mem_state.cc:443: info: Increasing stack size by one page.
958601.094721
exiting with last active thread context @ 47725380000
mospic@ubuntu:~/gem5-stable$
```

得到的执行结果仍然为正确结果，因此我所做的修改不影响函数的执行结果，即对于相同输入，改写后的函数仍能得到相同的结果。

得到的CPI，执行时长，指令条数如下：

	CPI	执行时长	指令条数
daxpy	1.778026	0.000036	80014
daxpy_unroll	2.005042	0.000035	70013
daxsbpxxy	2.096595	0.000063	120017
daxsbpxxy_unroll	2.254765	0.000062	110015
stencil	1.962439	0.000049	109995
stencil_unroll	2.187345	0.000044	85001

5. 修改 HPI.py 文件，如下所示：

```
1321 ~ class HPI_FUPool(MinorFUPool):
1322 ~     funcUnits = [HPI_IntFU(), # 0
1323 ~                 HPI_Int2FU(), # 1
1324 ~                 HPI_IntMulFU(), # 2
1325 ~                 HPI_IntDivFU(), # 3
1326 ~                 HPI_FloatSimdFU(), # 4
1327 ~                 HPI_MemFU(), # 5
1328 ~                 HPI_MiscFU(), # 6
1329 ~                 HPI_FloatSimdFU(),
1330 ~                 HPI_FloatSimdFU(),
1331 ~                 HPI_FloatSimdFU()
1332 ~                 ]
1333
```

添加三个 `HPI_FloatSimdFU()` 个仿真文件，修改后重新使用gem5进行编译，得到对应 `stats.txt`。

6. 修改Makefile为O3优化，将 `HPI.py` 中的硬件删去，使用命令运行 `daxpy.cc`，并得到统计文件。

回答问题：

1. 如何证明展开循环后的函数产生了正确的结果？

在未修改任何函数时，得到的最终结果为**958601.094721**，修改unroll函数展开后得到的结果为仍然为**958601.094721**，因此可以判定是产生的正确结果。

2.对于每一个函数，循环展开是否提升了性能？循环展开减少了哪一种hazard？

对于每一种函数，循环展开都使运行时间降低，对于执行相同的程序来说，循环展开提升了性能（表见实验过程第4步）。循环展开减少了Branch时产生的Hazard，因为展开循环相对于减少了Branch指令的数目；同时循环展开有利于指令调度使指令之间的Stall减少，因为瓶颈部分（如Float相乘）会被调度至循环式子的较前面部分。

BranchPred.lookups(可显著表征Branch指令的数目)

	daxpy	daxsbxpxy	stencil
未展开	10025	10026	10028
展开	2515	2520	5014

可以看出，展开后Branch的查找数显著降低，从而减少了因为控制相关而产生的Stall。

3.你应该展开循环多少次？每个循环都一样吗？如果你没有展开足够多或展开太多会影响程序性能吗？

我在 `daxpy` 与 `daxsbxpxy` 中循环展开四次，在 `stencil` 中循环展开两次。循环展开的次数应依赖于其可被展开成循环数可以整除的因子，且增加展开的次数对性能提升的幅度不再有很大的提升时。

每个循环都不一样，因为对于不同的函数，循环中的代码不同会导致指令是不相同的，从而导致调用的硬件资源也是不相同的，对于不同的指令来说，循环展开使得性能最优化的次数也应该是不一样的。

如果展开的太少，则在每个循环Branch指令结束前需要产生足够的Stall来等待瓶颈指令的执行（如FloatMult），此时性能会差于循环展开的多一些的程序的性能；当循环数增加到某个特定的值时，此时性能会变化不大，而循环展开较多时仅Branch指令会减少。

4.增加硬件对循环展开版本的函数和原函数有什么影响？添加更多硬件会减少哪种或哪些hazard？

添加硬件与为添加硬件的循环展开与原函数的各个指标如下：

	CPI(未添加硬件)	执行时长(未添加硬件)	指令条数(未添加硬件)	CPI(添加硬件)	执行时长(添加硬件)	指令条数(添加硬件)
daxpy	1.778001	0.000036	80014	1.778026	0.000036	80014
daxpy_unroll	2.005042	0.000035	70013	2.005042	0.000035	70013
daxsbpxy	2.013281	0.000060	120017	2.096595	0.000063	120017
daxsbpxy_unroll	2.175294	0.000060	110015	2.254765	0.000062	110015
stencil	1.962439	0.000049	109995	1.962439	0.000049	109995
stencil_unroll	2.124859	0.000042	85001	2.187345	0.000044	85001

可以看出，对比同一个函数，不论展开或是未展开函数的指令条数均相等，而CPI都有所提升，因此增加硬件可以有效减少在同一个循环中多次使用FloatSimdFU器件所带来的的开销，即添加更多硬件会减少**结构相关**的冲突。

5.选择你认为合适的指标比较四个版本函数的性能表现，为什么选择该指标？

我选择**执行时间**作为版本函数的性能表现。

原因：

- 对于同一个函数，展开与未展开程序所进行的工作是一样的。而对于相同的工作，**执行时间**越短，则代表**性能**越好。

6.你认为本次实验中你所进行的手动循环展开优化有意义吗？还是说编译器优化代码就已经足够了？说明理由。

未添加硬件时O3优化与O1优化如下所示：

	CPI (O1优化)	执行时长(O1优化)	指令条数(O1优化)	CPI (O3优化)	执行时长(O3优化)	指令条数(O3优化)
daxpy	1.778026	0.000036	80014	1.822548	0.000018	45022
daxpy_unroll	2.005042	0.000035	70013	2.016820	0.000015	37523
daxsbpxy	2.096595	0.000063	120017	2.061265	0.000028	60023
daxsbpxy_unroll	2.254765	0.000062	110015	1.456981	0.000017	52525
stencil	1.962439	0.000049	109995	2.239337	0.000034	69998
stencil_unroll	2.187345	0.000044	85001	3.088532	0.000035	60010

- 对于同一个函数与同种优化，展开后的**执行时间**与**指令条数**基本均小于未展开的函数的**执行时间**与**指令条数**(tencil中O3优化为一个例外，可能存在数据相关的原因)，因此手动循环展开对于执行的优化是有意义的，可以降低执行时长。
- 可以看出，对于同一个函数，不论手动循环展开与否，O3优化的**执行时间**与**指令条数**均少于O1优化，因此编译器多带来的优化也是十分显著的。

因此，我认为手动循环展开是有意义的，其仍然可以基于编译器优化后进一步提升性能，降低指令的数量，减少执行时长。

