计算机体系结构实验 1 - 3 实验报告

实现流程

Lab 1

Task 1

- 安装编译的基础环境

```
sudo apt update
sudo apt install build-essential net-tools git vim cmake gdb make gfortran
libnuma-dev libtirpc-dev
```

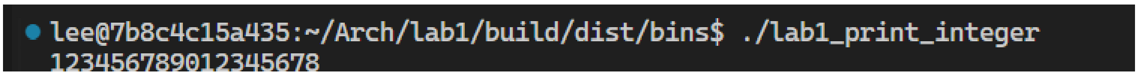
- 编译代码

```
cd lab1
mkdir -p build && cd build
cmake -B . -S ../ && cmake --build ./ --target lab1_print_integer
```

- 执行可执行文件

```
cd dist/bins/ && ./lab1_print_integer
```

- 得到正确输出



Task 2

- 完善`gemm_kernel.s`代码，实现矩阵B地址保存，加载`A[m][k]`到FPU寄存器栈，加载`B[k][n]`到FPU寄存器栈，加载`C[m][n]`到FPU寄存器栈。

```
.text;
.p2align 2;
.global gemm_kernel;
.type gemm_kernel, %function;

// 以下是宏定义，方便按逻辑梳理
#define     MAT_C               %rdi
#define     MAT_A               %rsi
#define     MAT_B               %r14
#define     DIM_M               %rcx
#define     DIM_N               %r8
#define     DIM_K               %r9
#define     loop_m              %r10
#define     loop_k              %r11
#define     loop_n              %r12
#define     mat_elem_idx        %r13


.macro PUSHD                                        // 保存原通用寄存器值
```

```
    push %rax
    push %rbx
    push %rcx
    push %rdx
    push %rsi
    push %rdi
    push %rbp
    push %r8
    push %r9
    push %r10
    push %r11
    push %r12
    push %r13
    push %r14
    push %r15
.endm

.macro POPD                                             // 恢复原通用寄存器值
    pop %r15
    pop %r14
    pop %r13
    pop %r12
    pop %r11
    pop %r10
    pop %r9
    pop %r8
    pop %rbp
    pop %rdi
    pop %rsi
    pop %rdx
    pop %rcx
    pop %rbx
    pop %rax
.endm

.macro GEMM_INIT                                        // 初始化
    // TODO: 将矩阵B的地址存入MAT_B宏对应的寄存器
    mov %rdx, MAT_B
    xor loop_m, loop_m
    xor loop_k, loop_k
    xor loop_n, loop_n
.endm

.macro DO_GEMM                                           // 使用kij遍历方式计算矩阵乘
法
DO_LOOP_K:                                               // 最外层的K维度的循环
    xor loop_m, loop_m                                   // 清空M维度的循环计数器

DO_LOOP_M:                                               // M维度的循环
    xor loop_n, loop_n                                   // 清空M维度的循环计数器

    // TODO:  加载A[m][k]


    mov loop_m, mat_elem_idx                // mat_elem_idx = m
    imul  DIM_K,mat_elem_idx                // mat_elem_idx = m * K
```

```
    add loop_k, mat_elem_idx            // mat_elem_idx = m * K + k

    flds (MAT_A, mat_elem_idx, 4)                   // 加载 A[m][k]

DO_LOOP_N:
    // TODO:  加载B[k][n]

    mov loop_k, mat_elem_idx            // mat_elem_idx = k
    imul DIM_N , mat_elem_idx         // mat_elem_idx = k * N
    add loop_n, mat_elem_idx            // mat_elem_idx = k * N + n

    flds (MAT_B, mat_elem_idx, 4)                   // 加载 B[k][n]

    fmul %st(1), %st(0)                             // 计算A[m][k] * B[k][n]

    // TODO: 加载C[m][n]


    mov loop_m, mat_elem_idx            // mat_elem_idx = m
    imul  DIM_N ,mat_elem_idx         // mat_elem_idx = m * N
    add loop_n, mat_elem_idx            // mat_elem_idx = m * N + n

    flds (MAT_C, mat_elem_idx, 4)                   // 加载 C[m][n]

    faddp %st(1), %st(0)                            // 计算 C[m][n] + A[m][k]
* B[k][n]
    fstps (MAT_C, mat_elem_idx, 4)                  // 写回 C[m][n]

    add $1, loop_n                                  // N维度的循环计数器加1
    cmp DIM_N, loop_n
    jl DO_LOOP_Na                // 清空st(0)，此时矩阵A的元素不再使用
    add $1, loop_m                                  // M维度的循环计数器加1
    cmp DIM_M, loop_m
    jl DO_LOOP_M

    add $1, loop_k                                  // K维度的循环计数器加1
    cmp DIM_K, loop_k
    jl DO_LOOP_K
.endm

gemm_kernel:
    PUSHD
    GEMM_INIT
    DO_GEMM
    POPD
    ret
```

- 验证正确性

```
mkdir -p build && cd build
cmake -B . -S ../ && cmake --build ./ --target lab1_test_gemm_kernel.unittest
./dist/bins/lab1_test_gemm_kernel.unittest --gtest_filter=gemm_kernel.test0
```

```
lee@7b8c4c15a435:~/Arch/lab1/build$ cmake -B . -S ../ && cmake --build ./ --target lab1_test_gemm_kernel.unittest
./dist/bins/lab1_test_gemm_kernel.unittest --gtest_filter=gemm_kernel.test0
-- CMAKE_BUILD_TYPE is Debug
-- Configuring done
-- Generating done
-- Build files have been written to: /home/lee/Arch/lab1/build
[ 28%] Built target gtest
[ 57%] Built target gtest_main
[100%] Built target lab1_test_gemm_kernel.unittest
Running main() from /home/lee/Arch/lab1/build/_deps/googletest-src/googletest/src/gtest_main.cc
Note: Google Test filter = gemm_kernel.test0
[==========] Running 1 test from 1 test suite.
[----------] Global test environment set-up.
[----------] 1 test from gemm_kernel
[ RUN      ] gemm_kernel.test0
[       OK ] gemm_kernel.test0 (52 ms)
[----------] 1 test from gemm_kernel (52 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (52 ms total)
[  PASSED  ] 1 test.
```

- 编译上层代码并执行并执行

```
mkdir -p build && cd build
cmake -B . -S ../ && cmake --build ./ --target lab1_gemm
./dist/bins/lab1_gemm 256 256 256
```

```
lee@7b8c4c15a435:~/Arch/lab1/build$ ./dist/bins/lab1_gemm 256 256 256
GEMM performance info:
                M, K, N: 256, 256, 256
                Ops: 0.0335544
                Total compute time(s): 2.2742
                Cost(s): 0.011371
                Benchmark(Gflops): 2.95088
lee@7b8c4c15a435:~/Arch/lab1/build$
```

Task 3

- 通过`lscpu`命令查看处理器型号、缓存层级信息

```
lee@7b8c4c15a435:~/Arch/lab1/build$ lscpu
Architecture:             x86_64
  CPU op-mode(s):         32-bit, 64-bit
  Address sizes:          39 bits physical, 48 bits virtual
  Byte Order:             Little Endian
CPU(s):                   8
  On-line CPU(s) list:    0-7
Vendor ID:                GenuineIntel
  Model name:             12th Gen Intel(R) Core(TM) i3-12300T
    CPU family:           6
    Model:                151
    Thread(s) per core:   2
    Core(s) per socket:   4
    Socket(s):            1
    Stepping:             5
    CPU(s) scaling MHz:   19%
    CPU max MHz:          4200.0000
    CPU min MHz:          800.0000
    BogoMIPS:             4608.00
    Flags:                fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc art arch_perfmon pebs
                           bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf tsc_known_freq pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe
                          popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch cpuid_fault epb ssbd ibrs ibpb stibp ibrs_enhanced tpr_shadow flexpriority ept vpid ept_ad fsgsbase tsc_a
                          djust bmi1 avx2 smep bmi2 erms invpcid rdseed adx smap clflushopt clwb intel_pt sha_ni xsaveopt xsavec xgetbv1 xsaves split_lock_detect user_shstk avx_vnni dtherm ida arat pln pts hwp
                           hwp_notify hwp_act_window hwp_epp hwp_pkg_req hfi vnmi umip pku ospke waitpkg gfni vaes vpclmulqdq rdpid movdiri movdir64b fsrm md_clear serialize arch_lbr ibt flush_l1d arch_capabil
                          ities
Virtualization features:
  Virtualization:         VT-x
Caches (sum of all):
  L1d:                    192 KiB (4 instances)
  L1i:                    128 KiB (4 instances)
  L2:                     5 MiB (4 instances)
  L3:                     12 MiB (1 instance)
NUMA:
  NUMA node(s):           1
  NUMA node0 CPU(s):      0-7
Vulnerabilities:
  Gather data sampling:   Not affected
  Itlb multihit:          Not affected
  L1tf:                   Not affected
  Mds:                    Not affected
  Meltdown:               Not affected
  Mmio stale data:        Not affected
  Reg file data sampling: Not affected
  Retbleed:               Not affected
  Spec rstack overflow:   Not affected
  Spec store bypass:      Mitigation; Speculative Store Bypass disabled via prctl
  Spectre v1:             Mitigation; usercopy/swapgs barriers and __user pointer sanitization
  Spectre v2:             Mitigation; Enhanced / Automatic IBRS; IBPB conditional; RSB filling; PBRSB-eIBRS SW sequence; BHI BHI_DIS_S
  Srbds:                  Not affected
  Tsx async abort:        Not affected
```

- 查看`cpu0`的`L1D`缓存的组数、组相联数和缓存行大小

```
lee@7b8c4c15a435:/sys/devices/system/cpu/cpu0/cache/index0$ # 查看缓存行大小
cat coherency_line_size
64
lee@7b8c4c15a435:/sys/devices/system/cpu/cpu0/cache/index0$ # 查看组数
cat number_of_sets
64
lee@7b8c4c15a435:/sys/devices/system/cpu/cpu0/cache/index0$ # 查看way数
cat ways_of_associativity
12
```

- 使用相同的方法查看L2, L3缓存的组数、组相联数和缓存行大小

```
● lee@7b8c4c15a435:/sys/devices/system/cpu/cpu0/cache/index2$ cat coherency_line_size
64
● lee@7b8c4c15a435:/sys/devices/system/cpu/cpu0/cache/index2$ cat number_of_sets
2048
● lee@7b8c4c15a435:/sys/devices/system/cpu/cpu0/cache/index2$ cat ways_of_associativity
10
● lee@7b8c4c15a435:/sys/devices/system/cpu/cpu0/cache/index2$ cd ../index3/
● lee@7b8c4c15a435:/sys/devices/system/cpu/cpu0/cache/index3$  cat coherency_line_size
64
● lee@7b8c4c15a435:/sys/devices/system/cpu/cpu0/cache/index3$ cat number_of_sets
16384
● lee@7b8c4c15a435:/sys/devices/system/cpu/cpu0/cache/index3$ cat ways_of_associativity
12
○ lee@7b8c4c15a435:/sys/devices/system/cpu/cpu0/cache/index3$ 
```

Task 4

- 命令安装`perf`

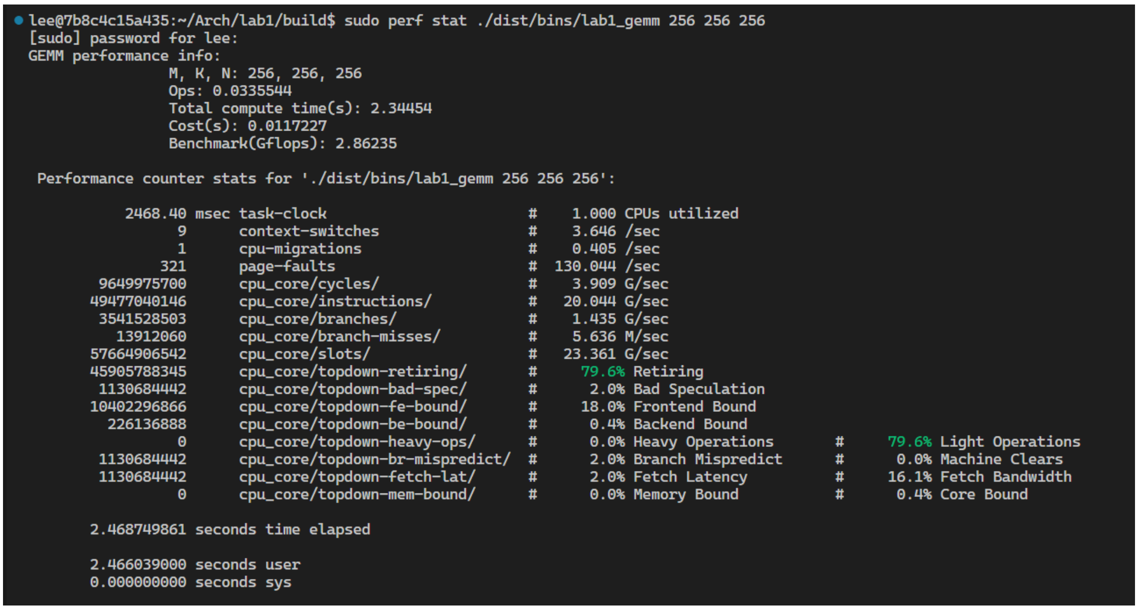
```
sudo apt install linux-perf
```

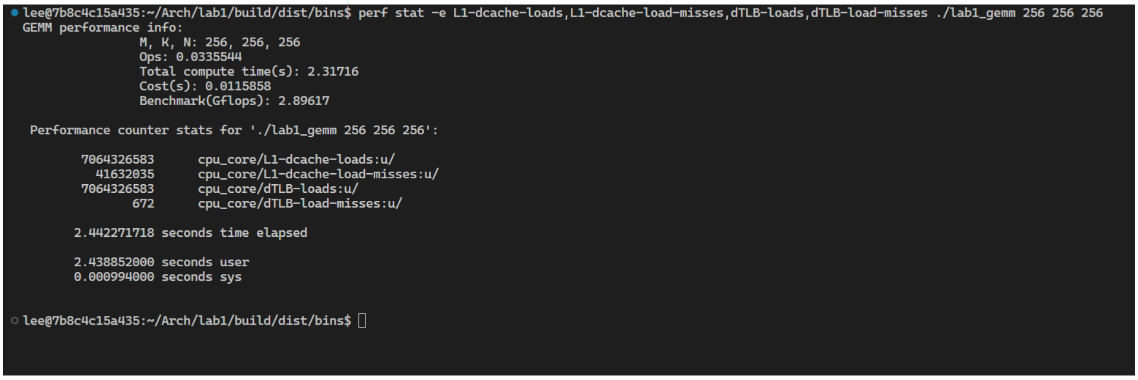
- 查看perf支持的性能事件

```
○ lee@7b8c4c15a435:/sys/devices/system/cpu/cpu0/cache/index3$ perf list

List of pre-defined events (to be used in -e or -M):

  branch-instructions OR branches                    [Hardware event]
  branch-misses                                      [Hardware event]
  bus-cycles                                         [Hardware event]
  cache-misses                                       [Hardware event]
  cache-references                                   [Hardware event]
  cpu-cycles OR cycles                               [Hardware event]
  instructions                                       [Hardware event]
  ref-cycles                                         [Hardware event]

  alignment-faults                                   [Software event]
  bpf-output                                         [Software event]
  cgroup-switches                                    [Software event]
  context-switches OR cs                             [Software event]
  cpu-clock                                          [Software event]
  cpu-migrations OR migrations                       [Software event]
  dummy                                              [Software event]
  emulation-faults                                   [Software event]
  major-faults                                       [Software event]
  minor-faults                                       [Software event]
  page-faults OR faults                              [Software event]
  task-clock                                         [Software event]

  duration_time                                      [Tool event]
  user_time                                          [Tool event]
  system_time                                        [Tool event]

  L1-dcache-load-misses                              [Hardware cache event]
  L1-dcache-loads                                    [Hardware cache event]
  L1-dcache-stores                                   [Hardware cache event]
  L1-icache-load-misses                              [Hardware cache event]
  LLC-load-misses                                    [Hardware cache event]
  LLC-loads                                          [Hardware cache event]
  LLC-store-misses                                   [Hardware cache event]
  LLC-stores                                         [Hardware cache event]
  branch-load-misses                                 [Hardware cache event]
  branch-loads                                       [Hardware cache event]
  dTLB-load-misses                                   [Hardware cache event]
  dTLB-loads                                         [Hardware cache event]
```

- 查看`lab1_gemm`程序的缓存使用情况

```
# 查看基本性能事件
perf stat ./dist/bins/lab1_gemm 256 256 256
# 查看指定的性能事件(-e)，支持的性能事件可由perf list查看
perf stat -e L1-dcache-loads,L1-dcache-load-misses,dTLB-loads,dTLB-load-misses ./lab1_gemm 256 256 256
```





Lab 2

Task 1

- 编译生成待分析的二进制程序`lab2_gemm_baseline`

```
mkdir -p build && cd build
cmake -B . -S ../ && cmake --build ./ --target lab2_gemm_baseline
```

- 使用perf stat命令结合-e参数指定事件查看程序的性能事件并根据结果分析定位性能瓶颈点

```
perf stat -e l2_rqsts.code_rd_hit,l2_rqsts.references,l1d.replacement,l1d_pend_miss.pending,l2_rqsts.swpf_hit,l2_rqsts.swpf_miss,L1-dcache-loads,L1-dcache-load-misses ./dist/bins/lab2_gemm_baseline 256 1024 256
```

```
lee@7b8c4c15a435:~/Arch/lab2/build$ perf stat -e l2_rqsts.code_rd_hit,l2_rqsts.references,l1d.replacement,l1d_pend_miss.pending,l2_rqsts.swpf_hit,l2_rqsts.swpf_miss,L1-dcache-loads,L1-dcache-load-misses ./dist/bins/lab2_gemm_baseline 256 1024 256
GEMM performance info:
                M, K, N: 256, 1024, 256
                Ops: 0.134218
                Total compute time(s): 10.6763
                Cost(s): 0.0533814
                Benchmark(Gflops): 2.51432

 Performance counter stats for './dist/bins/lab2_gemm_baseline 256 1024 256':

            47451      cpu_core/l2_rqsts.code_rd_hit:u/                     (50.00%)
       1274105478      cpu_core/l2_rqsts.references:u/                      (50.00%)
        950603008      cpu_core/l1d.replacement:u/                          (50.00%)
        819858286      cpu_core/l1d_pend_miss.pending:u/                    (49.99%)
                0      cpu_core/l2_rqsts.swpf_hit:u/                        (50.00%)
                0      cpu_core/l2_rqsts.swpf_miss:u/                       (50.00%)
      28254448497      cpu_core/L1-dcache-loads:u/                          (50.00%)
         45754179      cpu_core/L1-dcache-load-misses:u/                    (50.01%)

     11.223498781 seconds time elapsed

     11.180175000 seconds user
      0.003945000 seconds sys
```

Task 2

- 为练习1的代码增加数据预取优化，形成`src/lab2/gemm_kernel_opt_prefetch.S`中`DO_GEMM`过程（第64行）中的矩阵计算逻辑

```
.text;
.p2align 2;
.global gemm_kernel_opt_prefetch;
.type gemm_kernel_opt_prefetch, %function;

#define     MAT_C               %rdi
#define     MAT_A               %rsi
#define     MAT_B               %r14
#define     DIM_M               %rcx
#define     DIM_N               %r8
#define     DIM_K               %r9
#define     loop_m              %r10
#define     loop_k              %r11
#define     loop_n              %r12
#define     mat_elem_idx        %r13
#define     prefetch_elem_idx   %r15


.macro PUSHD   // 保存原通用寄存器值
    push %rax
    push %rbx
    push %rcx
    push %rdx
    push %rsi
    push %rdi
    push %rbp
    push %r8
    push %r9
    push %r10
    push %r11
    push %r12
    push %r13
    push %r14
    push %r15
```

```
.endm

.macro POPD    // 恢复原通用寄存器值
    pop %r15
    pop %r14
    pop %r13
    pop %r12
    pop %r11
    pop %r10
    pop %r9
    pop %r8
    pop %rbp
    pop %rdi
    pop %rsi
    pop %rdx
    pop %rcx
    pop %rbx
    pop %rax
.endm

.macro GEMM_INIT
    mov %rdx, MAT_B

    xor loop_m, loop_m
    xor loop_k, loop_k
    xor loop_n, loop_n
.endm

.macro DO_GEMM
DO_LOOP_K:
    xor loop_m, loop_m

DO_LOOP_M:
    xor loop_n, loop_n

    mov loop_m, %rax
    mul DIM_K
    mov %rax, mat_elem_idx
    add loop_k, mat_elem_idx                    // 计算 m*K+k
    prefetch (MAT_A, mat_elem_idx, 8)
    flds (MAT_A, mat_elem_idx, 4)     // 加载 A[m][k]

DO_LOOP_N:
    mov DIM_N, %rax
    mul loop_k
    mov %rax, mat_elem_idx
    add loop_n, mat_elem_idx
    prefetch (MAT_B, mat_elem_idx, 8)                // 计算 k*N+n
    flds (MAT_B, mat_elem_idx, 4)     // 加载 B[k][n]

    fmul %st(1), %st(0)                 // 计算A[m][k] * B[k][n]

    mov DIM_N, %rax
    mul loop_m
    mov %rax, mat_elem_idx
    add loop_n, mat_elem_idx                    // 计算 m*N+n
```

```
    prefetch (MAT_C, mat_elem_idx, 8)
    flds (MAT_C, mat_elem_idx, 4)     // 加载 C[m][n]

    faddp %st(1), %st(0)              // 计算 C[m][n] + A[m][k] * B[k][n]
    fstps (MAT_C, mat_elem_idx, 4)

    add $1, loop_n
    cmp DIM_N, loop_n
    jl DO_LOOP_N

    fstp %st(0)                    // 仅弹出元素
    add $1, loop_m
    cmp DIM_M, loop_m
    jl DO_LOOP_M

    add $1, loop_k
    cmp DIM_K, loop_k
    jl DO_LOOP_K
.endm

gemm_kernel_opt_prefetch:
    PUSHD
    GEMM_INIT
    DO_GEMM
    POPD
    ret
```

- 验证kernel结果的正确性

```
# 项目根目录下执行
mkdir -p build && cd build
cmake -B . -S ../ && cmake --build ./ --target
lab2_gemm_kernel_opt_prefetch.unittest

cd dist/bins && ./lab2_gemm_kernel_opt_prefetch.unittest
```

```
lee@7b8c4c15a435:~/Arch/lab2$ # 项目根目录下执行
mkdir -p build && cd build
cmake -B . -S ../ && cmake --build ./ --target lab2_gemm_kernel_opt_prefetch.unittest

cd dist/bins && ./lab2_gemm_kernel_opt_prefetch.unittest
-- unknown CMAKE_BUILD_TYPE =
-- Configuring done
-- Generating done
-- Build files have been written to: /home/lee/Arch/lab2/build
[ 14%] Building CXX object _deps/googletest-build/googletest/CMakeFiles/gtest.dir/src/gtest-all.cc.o
[ 28%] Linking CXX static library ../../../lib/libgtest.a
[ 28%] Built target gtest
[ 42%] Building CXX object _deps/googletest-build/googletest/CMakeFiles/gtest_main.dir/src/gtest_main.cc.o
[ 57%] Linking CXX static library ../../../lib/libgtest_main.a
[ 57%] Built target gtest_main
[ 71%] Building CXX object src/lab2/CMakeFiles/lab2_gemm_kernel_opt_prefetch.unittest.dir/test_gemm_kernel_opt_prefetch.cpp.o
[ 85%] Building ASM object src/lab2/CMakeFiles/lab2_gemm_kernel_opt_prefetch.unittest.dir/gemm_kernel_opt_prefetch.S.o
/home/lee/Arch/lab2/src/lab2/gemm_kernel_opt_prefetch.S: Assembler messages:
/home/lee/Arch/lab2/src/lab2/gemm_kernel_opt_prefetch.S:94: Warning: translating to `faddp %st,%st(1)'
/home/lee/Arch/lab2/src/lab2/gemm_kernel_opt_prefetch.S:114:  Info: macro invoked from here
[100%] Linking CXX executable ../../dist/bins/lab2_gemm_kernel_opt_prefetch.unittest
/usr/bin/ld: warning: CMakeFiles/lab2_gemm_kernel_opt_prefetch.unittest.dir/gemm_kernel_opt_prefetch.S.o: missing .note.GNU-sta
/usr/bin/ld: NOTE: This behaviour is deprecated and will be removed in a future version of the linker
[100%] Built target lab2_gemm_kernel_opt_prefetch.unittest
Running main() from /home/lee/Arch/lab2/build/_deps/googletest-src/googletest/src/gtest_main.cc
[==========] Running 4 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 4 tests from gemm_kernel_opt_prefetch
[ RUN      ] gemm_kernel_opt_prefetch.test0
[       OK ] gemm_kernel_opt_prefetch.test0 (18 ms)
[ RUN      ] gemm_kernel_opt_prefetch.test1
[       OK ] gemm_kernel_opt_prefetch.test1 (3 ms)
[ RUN      ] gemm_kernel_opt_prefetch.test2
[       OK ] gemm_kernel_opt_prefetch.test2 (4 ms)
[ RUN      ] gemm_kernel_opt_prefetch.test3
[       OK ] gemm_kernel_opt_prefetch.test3 (0 ms)
[----------] 4 tests from gemm_kernel_opt_prefetch (26 ms total)

[----------] Global test environment tear-down
[==========] 4 tests from 1 test suite ran. (26 ms total)
[  PASSED  ] 4 tests.
```

- 对比优化后的算法与基线的性能

```
mkdir -p build && cd build
cmake -B . -S ../ && cmake --build ./ --target lab2_gemm_opt_prefetch
cd dist/bins && ./lab2_gemm_opt_prefetch 1024 128 4
```

```
lee@7b8c4c15a435:~/Arch/lab2$ mkdir -p build && cd build
cmake -B . -S ../ && cmake --build ./ --target lab2_gemm_opt_prefetch
cd dist/bins && ./lab2_gemm_opt_prefetch 1024 128 4
-- unknown CMAKE_BUILD_TYPE =
-- Configuring done
-- Generating done
-- Build files have been written to: /home/lee/Arch/lab2/build
[ 25%] Building ASM object src/lab2/CMakeFiles/lab2_gemm_opt_prefetch.dir/gemm_kernel_baseline.S.o
/home/lee/Arch/lab2/src/lab2/gemm_kernel_baseline.S: Assembler messages:
/home/lee/Arch/lab2/src/lab2/gemm_kernel_baseline.S:91: Warning: translating to `faddp %st,%st(1)'
/home/lee/Arch/lab2/src/lab2/gemm_kernel_baseline.S:111:  Info: macro invoked from here
[ 50%] Building ASM object src/lab2/CMakeFiles/lab2_gemm_opt_prefetch.dir/gemm_kernel_opt_prefetch.S.o
/home/lee/Arch/lab2/src/lab2/gemm_kernel_opt_prefetch.S: Assembler messages:
/home/lee/Arch/lab2/src/lab2/gemm_kernel_opt_prefetch.S:94: Warning: translating to `faddp %st,%st(1)'
/home/lee/Arch/lab2/src/lab2/gemm_kernel_opt_prefetch.S:114:  Info: macro invoked from here
[ 75%] Linking CXX executable ../../dist/bins/lab2_gemm_opt_prefetch
/usr/bin/ld: warning: CMakeFiles/lab2_gemm_opt_prefetch.dir/gemm_kernel_opt_prefetch.S.o: missing .note.G
/usr/bin/ld: NOTE: This behaviour is deprecated and will be removed in a future version of the linker
[100%] Built target lab2_gemm_opt_prefetch
--- Performance before prefetch optimization ---
GEMM performance info:
                M, K, N: 1024, 128, 4
                Ops: 0.00104858
                Total compute time(s): 0.093957
                Cost(s): 0.000469785
                Benchmark(Gflops): 2.23203
--- Performance for after prefetch optimization ---
GEMM performance info:
                M, K, N: 1024, 128, 4
                Ops: 0.00104858
                Total compute time(s): 0.109788
                Cost(s): 0.00054894
                Benchmark(Gflops): 1.91018
----------------------------
Performance difference(Gflops): -0.321851
```

Lab 3

Task 1

- 补充`src/lab3/gemm_kernel_opt_loop_unrolling.s`中缺失的代码

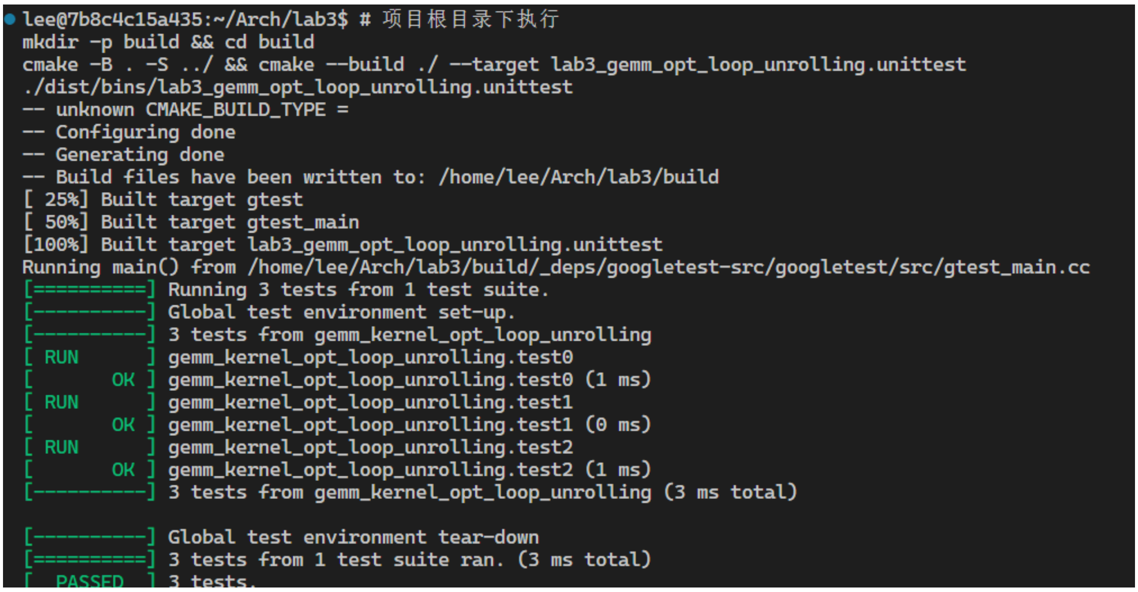
```
.text;
.p2align 2;
.global gemm_kernel_opt_loop_unrolling;
.type gemm_kernel_opt_loop_unrolling, %function;

#define     MAT_C               %rdi
#define     MAT_A               %rsi
#define     MAT_B               %r14
#define     DIM_M               %rcx
#define     DIM_N               %r8
#define     DIM_K               %r9
#define     loop_m              %r10
#define     loop_k              %r11
#define     loop_n              %r12
#define     mat_elem_idx        %r13


.macro PUSHD    // 保存原通用寄存器值
    push %rax
    push %rbx
    push %rcx
    push %rdx
```

```
    push %rsi
    push %rdi
    push %rbp
    push %r8
    push %r9
    push %r10
    push %r11
    push %r12
    push %r13
    push %r14
    push %r15
.endm

.macro POPD    // 恢复原通用寄存器值
    pop %r15
    pop %r14
    pop %r13
    pop %r12
    pop %r11
    pop %r10
    pop %r9
    pop %r8
    pop %rbp
    pop %rdi
    pop %rsi
    pop %rdx
    pop %rcx
    pop %rbx
    pop %rax
.endm

.macro GEMM_INIT
    mov %rdx, MAT_B

    xor loop_m, loop_m
    xor loop_k, loop_k
    xor loop_n, loop_n
.endm

.macro DO_GEMM
DO_LOOP_K:
    xor loop_m, loop_m

DO_LOOP_M:
    xor loop_n, loop_n

    mov loop_m, %rax
    mul DIM_K
    mov %rax, mat_elem_idx
    add loop_k, mat_elem_idx                    // 计算 m*K+k
    flds (MAT_A, mat_elem_idx, 4)     // 加载 A[m][k]

DO_LOOP_N:
    mov DIM_N, %rax
    mul loop_k
    mov %rax, mat_elem_idx
```

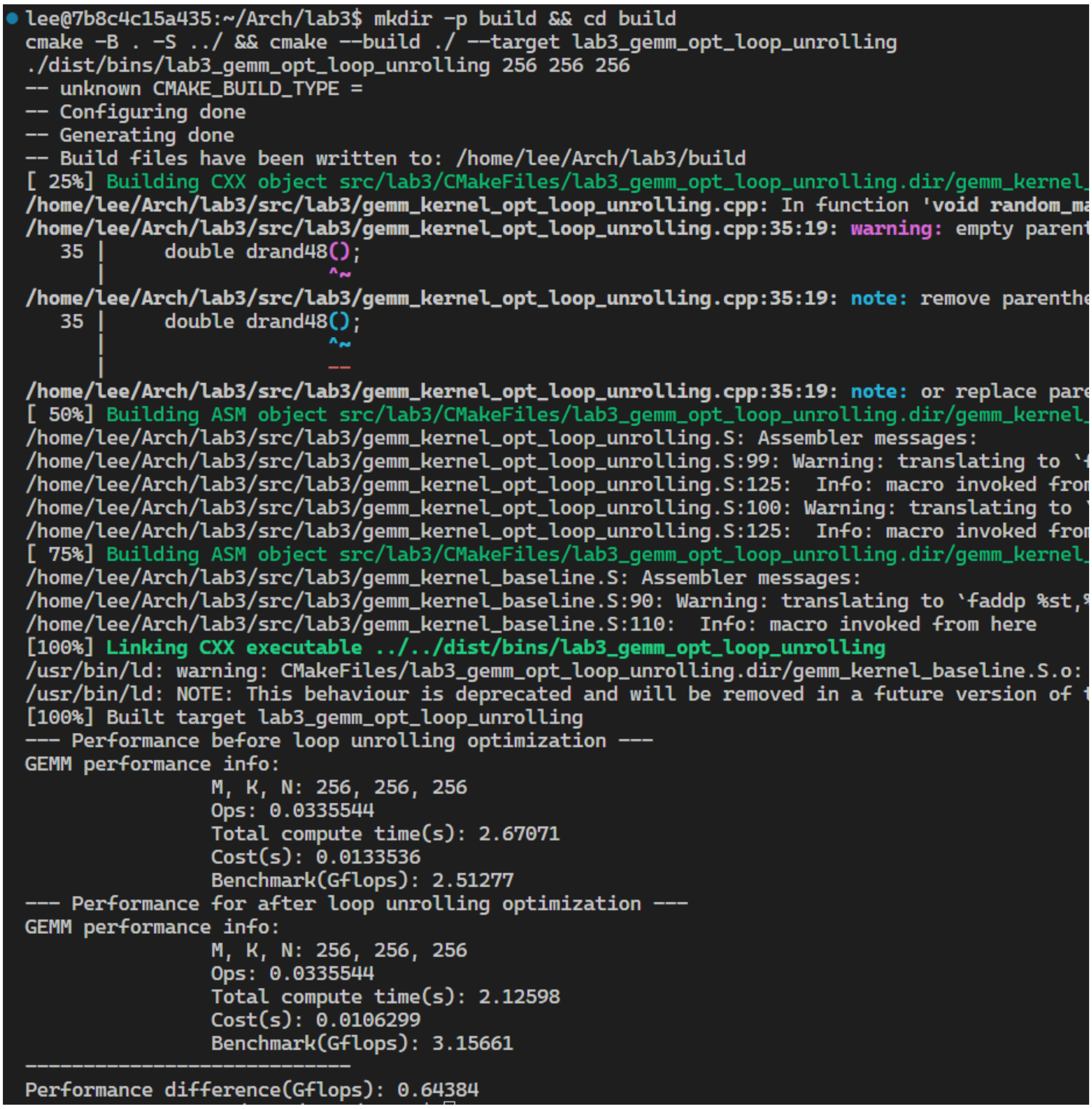
```
    add loop_n, mat_elem_idx
    flds (MAT_B, mat_elem_idx, 4)     // 加载 B[k][n]
    fmul %st(1), %st(0)                 // 计算A[m][k] * B[k][n]  --> st(0)

    // 添加计算A[m][k] * B[k][n+1] --> st(0)的逻辑
    add $1, mat_elem_idx                // 移动到B[k][n+1]的位置
    flds (MAT_B, mat_elem_idx, 4)       // 加载 B[k][n+1]
    fmul %st(2), %st(0)                 // 计算A[m][k] * B[k][n+1]  --> st(0)

    mov DIM_N, %rax
    mul loop_m
    mov %rax, mat_elem_idx
    add loop_n, mat_elem_idx                    // 计算 m*N+n

    // 添加加载C[m][n] --> st(1) 和 C[m][n+1] --> st(0)的逻辑
    flds (MAT_C, mat_elem_idx, 4)      // 加载 C[m][n] --> st(1)
    add $1, mat_elem_idx                // 移动到C[m][n+1]的位置
    flds (MAT_C, mat_elem_idx, 4)      // 加载 C[m][n+1] --> st(0)

    // 添加部分和累加逻辑: C[m][n+1] + A[m][k] * B[k][n+1] 和 C[m][n] + A[m][k]
* B[k][n]
    faddp %st(2), %st(0)                // st(0) = C[m][n+1] + A[m][k] * B[k]
[n+1]
    faddp %st(2), %st(0)                // st(0) = C[m][n] + A[m][k] * B[k]
[n]

    // 保存C[m][n+1] 和 C[m][n]
    fstps (MAT_C, mat_elem_idx, 4)      // 保存C[m][n+1]
    sub $1, mat_elem_idx               // 移动到C[m][n]的位置
    fstps (MAT_C, mat_elem_idx, 4)      // 保存C[m][n]

    // 更新N维度的循环控制变量
    add $2, loop_n                      // 每次迭代增加2，因为同时处理了两个元素
    cmp DIM_N, loop_n
    jl DO_LOOP_N

    fstp %st(0)                 // 仅弹出元素
    add $1, loop_m
    cmp DIM_M, loop_m
    jl DO_LOOP_M

    add $1, loop_k
    cmp DIM_K, loop_k
    jl DO_LOOP_K
.endm

gemm_kernel_opt_loop_unrolling:
    PUSHD
    GEMM_INIT
    DO_GEMM
    POPD
    ret
```

- 验证循环展开方案的正确性

```
# 项目根目录下执行
mkdir -p build && cd build
cmake -B . -S ../ && cmake --build ./ --target
lab3_gemm_opt_loop_unrolling.unittest
./dist/bins/lab3_gemm_opt_loop_unrolling.unittest
```



- 对比优化后的算法与基线的性能



Task 2

- 补充`src/lab3/gemm_kernel_opt_avx.S中缺失的代码

```
.text;
.p2align 2;
.global gemm_kernel_opt_avx;
.type gemm_kernel_opt_avx, %function;


#define     AVX_REG_BYTE_WIDTH  32

#define     MAT_C               %rdi
#define     MAT_A               %rsi
#define     MAT_B               %r13
#define     DIM_M               %rcx
#define     DIM_N               %r8
#define     DIM_K               %r9
#define     loop_m              %r10
#define     loop_k              %r11
#define     loop_n              %r12
#define     mat_elem_idx        %r14
#define     temp_reg            %r15

// 以下是计算过程中用到的avx寄存器
#define     mat_c0_0_8           %ymm0
#define     mat_c0_8_16          %ymm1
#define     mat_c0_16_24         %ymm2
#define     mat_c0_24_32         %ymm3
#define     mat_c1_0_8           %ymm4
#define     mat_c1_8_16          %ymm5
#define     mat_c1_16_24         %ymm6
#define     mat_c1_24_32         %ymm7
#define     mat_a0_0_8           %ymm8
#define     mat_a1_0_8           %ymm9
#define     mat_b0_0_8           %ymm10
#define     mat_b0_8_16          %ymm11
#define     mat_b0_16_24         %ymm12
#define     mat_b0_24_32         %ymm13

.macro PUSHD   // 保存原通用寄存器值
    push %rax
    push %rbx
    push %rcx
    push %rdx
    push %rsi
    push %rdi
    push %rbp
    push %r8
    push %r9
    push %r10
    push %r11
    push %r12
    push %r13
    push %r14
```

```
    push %r15
.endm

.macro POPD    // 恢复原通用寄存器值
    pop %r15
    pop %r14
    pop %r13
    pop %r12
    pop %r11
    pop %r10
    pop %r9
    pop %r8
    pop %rbp
    pop %rdi
    pop %rsi
    pop %rdx
    pop %rcx
    pop %rbx
    pop %rax
.endm

.macro GEMM_INIT
    mov %rdx, MAT_B
.endm

.macro LOAD_MAT_A     // 每次装载矩阵A同一列的2个元素，即A[m][k]，A[m+1][k]
    // 装载A[m][k]的数据
    mov loop_m, %rax
    mul DIM_K
    mov %rax, temp_reg
    add loop_k, temp_reg

    // 计算A[m][k]的字节地址
    mov temp_reg, mat_elem_idx
    shl $2, mat_elem_idx        // 左移，相当于乘4

    vbroadcastss (MAT_A, mat_elem_idx), mat_a0_0_8    // 将A[m][k]广播到AVX寄存器的8个单元

    ;// TODO 练习3：请添加加载并广播A[m+1][k]-->mat_a1_0_8的逻辑
    mov  temp_reg,mat_elem_idx
    add DIM_K ,mat_elem_idx
    shl $2 ,mat_elem_idx
    vbroadcastss (MAT_A, mat_elem_idx), mat_a1_0_8

.endm

.macro LOAD_MAT_B    // 每次装载矩阵B一行32个元素，即B[k][n:n+32]

    ;// TODO 练习3：请添加加载B[k][n:n+32]-->mat_b0_0_8，mat_b0_8_16，mat_b0_16_24，mat_b0_24_32的逻辑
    mov loop_k, %rax
    mul DIM_N
    mov %rax, temp_reg
    add loop_n, temp_reg
```

```
    // 计算B[k][n]的字节地址
    mov temp_reg, mat_elem_idx
    shl $2, mat_elem_idx        // 左移，相当于乘4

    // 加载B[k][n:n+8]到mat_b0_0_8
    vmovups (MAT_B, mat_elem_idx), mat_b0_0_8

    // 加载B[k][n+8:n+16]到mat_b0_8_16
    add $32, mat_elem_idx       // 偏移量为8个float，即32字节
    vmovups (MAT_B, mat_elem_idx), mat_b0_8_16

    // 加载B[k][n+16:n+24]到mat_b0_16_24
    add $32, mat_elem_idx       // 再偏移32字节
    vmovups (MAT_B, mat_elem_idx), mat_b0_16_24

    // 加载B[k][n+24:n+32]到mat_b0_24_32
    add $32, mat_elem_idx       // 再偏移32字节
    vmovups (MAT_B, mat_elem_idx), mat_b0_24_32

.endm

.macro LOAD_MAT_C
    mov loop_m, %rax
    mul DIM_N
    mov %rax, temp_reg
    add loop_n, temp_reg

    // 装载矩阵C第一行的数据，即C[m][n:n+32]
    mov temp_reg, mat_elem_idx
    shl $2, mat_elem_idx        // 左移，相当于乘4

    // 加载C[m][n:n+8]到mat_c0_0_8
    vmovups (MAT_C, mat_elem_idx), mat_c0_0_8

    // 加载C[m][n+8:n+16]到mat_c0_8_16
    add $32, mat_elem_idx       // 偏移量为8个float，即32字节
    vmovups (MAT_C, mat_elem_idx), mat_c0_8_16

    // 加载C[m][n+16:n+24]到mat_c0_16_24
    add $32, mat_elem_idx       // 再偏移32字节
    vmovups (MAT_C, mat_elem_idx), mat_c0_16_24

    // 加载C[m][n+24:n+32]到mat_c0_24_32
    add $32, mat_elem_idx       // 再偏移32字节
    vmovups (MAT_C, mat_elem_idx), mat_c0_24_32

    // 装载矩阵C第二行的数据，即C[m+1][n:n+32]
    mov temp_reg, mat_elem_idx
    add DIM_N, mat_elem_idx
    shl $2, mat_elem_idx        // 左移，相当于乘4

    // 加载C[m+1][n:n+8]到mat_c1_0_8
    vmovups (MAT_C, mat_elem_idx), mat_c1_0_8

    // 加载C[m+1][n+8:n+16]到mat_c1_8_16
    add $32, mat_elem_idx       // 偏移量为8个float，即32字节
```

```
    vmovups (MAT_C, mat_elem_idx), mat_c1_8_16

    // 加载C[m+1][n+16:n+24]到mat_c1_16_24
    add $32, mat_elem_idx       // 再偏移32字节
    vmovups (MAT_C, mat_elem_idx), mat_c1_16_24

    // 加载C[m+1][n+24:n+32]到mat_c1_24_32
    add $32, mat_elem_idx       // 再偏移32字节
    vmovups (MAT_C, mat_elem_idx), mat_c1_24_32
.endm

.macro STORE_MAT_C
    // 保存矩阵C第一行的数据
    mov loop_m, %rax
    mul DIM_N
    mov %rax, temp_reg
    add loop_n, temp_reg

    // 保存矩阵C第一行的数据，即C[m][n:n+32]
    mov temp_reg, mat_elem_idx
    shl $2, mat_elem_idx        // 左移，相当于乘4

    // 保存mat_c0_0_8到C[m][n:n+8]
    vmovups mat_c0_0_8, (MAT_C, mat_elem_idx)

    // 保存mat_c0_8_16到C[m][n+8:n+16]
    add $32, mat_elem_idx       // 偏移量为8个float，即32字节
    vmovups mat_c0_8_16, (MAT_C, mat_elem_idx)

    // 保存mat_c0_16_24到C[m][n+16:n+24]
    add $32, mat_elem_idx       // 再偏移32字节
    vmovups mat_c0_16_24, (MAT_C, mat_elem_idx)

    // 保存mat_c0_24_32到C[m][n+24:n+32]
    add $32, mat_elem_idx       // 再偏移32字节
    vmovups mat_c0_24_32, (MAT_C, mat_elem_idx)

    // 保存矩阵C第二行的数据，即C[m+1][n:n+32]
    mov temp_reg, mat_elem_idx
    add DIM_N, mat_elem_idx
    shl $2, mat_elem_idx        // 左移，相当于乘4

    // 保存mat_c1_0_8到C[m+1][n:n+8]
    vmovups mat_c1_0_8, (MAT_C, mat_elem_idx)

    // 保存mat_c1_8_16到C[m+1][n+8:n+16]
    add $32, mat_elem_idx       // 偏移量为8个float，即32字节
    vmovups mat_c1_8_16, (MAT_C, mat_elem_idx)

    // 保存mat_c1_16_24到C[m+1][n+16:n+24]
    add $32, mat_elem_idx       // 再偏移32字节
    vmovups mat_c1_16_24, (MAT_C, mat_elem_idx)

    // 保存mat_c1_24_32到C[m+1][n+24:n+32]
    add $32, mat_elem_idx       // 再偏移32字节
    vmovups mat_c1_24_32, (MAT_C, mat_elem_idx)
```

```
.endm

.macro DO_COMPUTE       // 计算 C[m:m+2][n:n+32] += A[m:m+2][k] * B[k:k+8]
[n:n+32]

    // 计算 C[m][n:n+32] += A[m][k] * B[k][n:n+32]
    vfmadd231ps mat_a0_0_8, mat_b0_0_8, mat_c0_0_8
    vfmadd231ps mat_a0_0_8, mat_b0_8_16, mat_c0_8_16
    vfmadd231ps mat_a0_0_8, mat_b0_16_24, mat_c0_16_24
    vfmadd231ps mat_a0_0_8, mat_b0_24_32, mat_c0_24_32

    // 计算 C[m+1][n:n+32] += A[m+1][k] * B[k][n:n+32]
    vfmadd231ps mat_a1_0_8, mat_b0_0_8, mat_c1_0_8
    vfmadd231ps mat_a1_0_8, mat_b0_8_16, mat_c1_8_16
    vfmadd231ps mat_a1_0_8, mat_b0_16_24, mat_c1_16_24
    vfmadd231ps mat_a1_0_8, mat_b0_24_32, mat_c1_24_32

.endm


.macro DO_GEMM
    xor loop_n, loop_n
DO_LOOP_N:

    xor loop_m, loop_m
DO_LOOP_M:
    // 装载矩阵C的数据
    LOAD_MAT_C

    xor loop_k, loop_k
DO_LOOP_K:
    // 装载矩阵A和矩阵B分块的数据
    LOAD_MAT_A
    LOAD_MAT_B

    DO_COMPUTE

    add $1, loop_k              // kr=1
    cmp DIM_K, loop_k
    jl DO_LOOP_K

    // 保存结果
    STORE_MAT_C

    add $2, loop_m              // mr=2
    cmp DIM_M, loop_m
    jl DO_LOOP_M

    add $32, loop_n             // nr=32
    cmp DIM_N, loop_n
    jl DO_LOOP_N

.endm
```

```
gemm_kernel_opt_avx:
    PUSHD
    GEMM_INIT
    DO_GEMM
    POPD
    ret
```

- 验证新内核的正确性

```
# 项目根目录下执行
mkdir -p build && cd build
cmake -B . -S ../ && cmake --build ./ --target lab3_gemm_opt_avx.unittest
./dist/bins/lab3_gemm_opt_avx.unittest
```

```
lee@7b8c4c15a435:~/Arch/lab3$ # 项目根目录下执行
mkdir -p build && cd build
cmake -B . -S ../ && cmake --build ./ --target lab3_gemm_opt_avx.unittest
./dist/bins/lab3_gemm_opt_avx.unittest
-- unknown CMAKE_BUILD_TYPE =
-- Configuring done
-- Generating done
-- Build files have been written to: /home/lee/Arch/lab3/build
[ 28%] Built target gtest
[ 57%] Built target gtest_main
[100%] Built target lab3_gemm_opt_avx.unittest
Running main() from /home/lee/Arch/lab3/build/_deps/googletest-src/googletest/src/gtest_main.cc
[==========] Running 3 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 3 tests from gemm_kernel_opt_avx
[ RUN      ] gemm_kernel_opt_avx.test0
[       OK ] gemm_kernel_opt_avx.test0 (0 ms)
[ RUN      ] gemm_kernel_opt_avx.test1
[       OK ] gemm_kernel_opt_avx.test1 (0 ms)
[ RUN      ] gemm_kernel_opt_avx.test2
[       OK ] gemm_kernel_opt_avx.test2 (1 ms)
[----------] 3 tests from gemm_kernel_opt_avx (1 ms total)

[----------] Global test environment tear-down
[==========] 3 tests from 1 test suite ran. (2 ms total)
[  PASSED  ] 3 tests.
lee@7b8c4c15a435:~/Arch/lab3/build$
```

- 对比优化后的算法与基线的性能

```
mkdir -p build && cd build
cmake -B . -S ../ && cmake --build ./ --target lab3_gemm_opt_avx
./dist/bins/lab3_gemm_opt_avx 256 256 256
```

```
lee@7b8c4c15a435:~/Arch/lab3$ mkdir -p build && cd build
cmake -B . -S ../ && cmake --build ./ --target lab3_gemm_opt_avx
./dist/bins/lab3_gemm_opt_avx 256 256 256
-- unknown CMAKE_BUILD_TYPE =
-- Configuring done
-- Generating done
-- Build files have been written to: /home/lee/Arch/lab3/build
[100%] Built target lab3_gemm_opt_avx
--- Performance before avx optimization ---
GEMM performance info:
                M, K, N: 256, 256, 256
                Ops: 0.0335544
                Total compute time(s): 2.62524
                Cost(s): 0.0131262
                Benchmark(Gflops): 2.55629
--- Performance for after avx optimization ---
GEMM performance info:
                M, K, N: 256, 256, 256
                Ops: 0.0335544
                Total compute time(s): 0.101179
                Cost(s): 0.000505895
                Benchmark(Gflops): 66.3269
----------------------------
Performance difference(Gflops): 63.7706
lee@7b8c4c15a435:~/Arch/lab3/build$
```

测试结果与原理分析

- 使用`pre_fetch`相较于`base_line`没有提升，甚至更差
    - 现代处理器已经内置了复杂的硬件预取机制，手动预取可能会干扰这些优化机制，导致性能下降。
- 使用`x87 FPU`能提升矩阵乘法性能，但提升并不明显
    - `x87` 的设计并非为并行计算而优化，其计算过程通常是串行的，因此难以充分利用现代处理器的并行能力。
    - `x87` 不支持一些现代硬件优化特性（如更高带宽的内存访问和矢量化运算），性能提升有限。
    - `x87` 使用堆栈结构（stack-based architecture）操作寄存器，指令灵活性较差，容易产生额外的寄存器切换开销。
- 使用`avx`指令能明显提升矩阵乘法的性能
    - `AVX` 指令可以一次处理 256 位（或更高位宽，如 AVX-512）数据，相当于并行计算 4 个或更多的双精度浮点数。这种并行处理显著提升了计算效率。
    - `AVX` 的指令能够同时进行加法和乘法（Fused Multiply-Add, FMA），这在矩阵乘法中非常高效。
    - `AVX` 的数据加载/存储是对齐的，减少了非对齐访问的开销。现代硬件中的内存控制器也为 `AVX` 指令集进行了优化。
    - `AVX` 支持更深的流水线，使得浮点运算和数据加载可以并行进行，减少了等待时间。