浙江大學

课程综合实践
实验报告

实验名称 GEMM 通用矩阵乘法

姓名学号 陶泓宇 3200103929

实验日期 2021 年 8 月 6 日

目录

插图

1 实验目的和要求

本次实验需要你使用 OpenMP、MPI 等 API 手工完成一个支持分布式计算高性能 GEMM 实现。

2 实验内容和原理

2.1 Blocking

Blocking 可以增加访存局部性，提高元素的复用，加快矩阵索引速度。下面是 1x4 的 blocking 的示意图以及 demo 代码

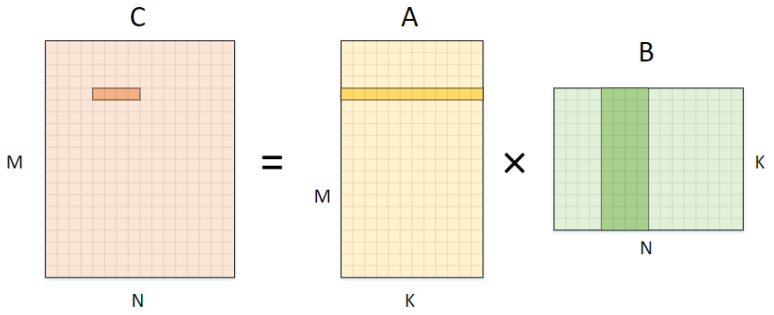

图 1: 1x4 的 blocking 示意图

使用 1x4 的 blocking 的 demo 代码：

```
for (int i=0;i<N;i++){
    for (int j=0;j<N;j+=4) {
        int sum[4];
        //clear sum, 使用sum寄存器减小索引时间
        for(int m=0;m<4;m++){
            sum[m] = 0;
       }
        for (int k=0;k<N;++k){
            int index = matC[i*N+k];
          //使用index寄存器减少索引时间
```

```
            for(int x_inner = 0;x_inner<4;x_inner++){
            sum[x_inner] += index * matA[(j+x_inner)*N+k];
            }
        }
        for(int fuzhi = 0;fuzhi<4;fuzhi++){
            matC2[i*N+j+fuzhi] = sum[fuzhi];
    }
  }
```

当采用 1x4 的 blocking 时，可以观察到 matC[i*N+k] 被复用了四次，由此采用 index 寄存器储存这个值，以减少索引的时间；nxn 时的情况同理。

2.2 向量化

施行向量化操作后，原本需要 64 条计算指令的计算过程所需指令减少到 16 条，访存也有类似效果。而向量化对处理器资源的高效使用，又带来了进一步优化空间，例如可以一次计算 8×8 个局部输出；gcc 可以在编译时自动向量化，所需代码如下：

```
$(CC) -fopenmp -O3 -march=core-avx2 -o
gemm hw_baseline.cpp $(CFLAGS)
```

该段代码在编译时采用了 avx2 指令集自动向量化。

2.3 openmp 优化

OpenMP 是一种用于共享内存并行系统的多线程程序设计方案，主要在线程级上优化（单机优化）；以下是一些常用语句：

```
#pragma omp parallel for
```

该语句告知编译器，紧接着的 for 循环可以被并行执行，并行线程数默认采用最大线程数，也可以通过以下语句指定并行线程数：

```
#pragma omp parallel for num_threads(n)
```

schedule 子句：

```
#pragma omp parallel for schedule(kind, chunksize)
```

当 kind 取 static 时，采用静态调度，0 - chunkSize-1 分配给第一个线程 chunkSize - 2chunkSize-1 分配给第二个线程，以此类推。

当 kind 取 dynamic 时，采用动态调度，线程空闲时会自动取领取一个 chunkSize 大小的任务，由于线程的启动时机和执行完的时间不确定，所以迭代器被分配到哪个线程时无法事先知道的。

当 kind 取 guided 时，采用启发式调度，chunkSize 表示每次分配的迭代次数的最小值每次分配给线程的迭代次数是不同的，开始可能比较大，以后逐渐减小，是一种更灵活的动态分配，当有大量任务时就会一次性分配较多任务，从而使得分配消耗更少，执行效率更高；当任务量较少时一次性分配较少线程，从而使得负载更加均衡。

2.4 MPI 优化

MPI 主要用于协调进程之间的并行，可以用于处理多台主机之间的通信。

头文件：

```
#include<mpi.h>
```

执行：

```
mpirun -np number name
```

生成 number 个进程来跑 name, 指定主机处理器时考虑使用 –host 和 –hostfile。

MPI 程序的开始与销毁：

```
int MPI_Init (int* argc ,char** argv[] )
int MPI_Finalize (void)
```

第一段程序表示 MPI 程序的开始，用于并行环境初始化，参数 argc_p 与 argv_p 是指向参数 argc 和 argv 的指针。当不需要的时候设置为 NULL。

调用 MPI_Finalize 是为了告知 MPI 系统 MPI 已经使用完毕。为 MPI 分配的任何资源都可以释放了。

```
int MPI_Comm_rank (MPI_Comm comm ,int* rank)
```

得到本进程在通信空间中的 rank 值, 即在组中的逻辑编号 (rank 值为 0 到 p-1 间的整数, 相当于进程的 ID。)

```
int MPI_Send( void *buff, int count,
MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)
```

void *buff：需要发送的变量，count: 发送消息的个数（如“hello”则 count 为 6）；MPI_Datatype datatype：发送的数据类型，dest：目的地进程号，tag：消息标签，接收方需要有相同的消息标签才能接收该消息，MPI_Comm comm：通讯域。表示你要向哪个组发送消息。

```
int MPI_Recv( void *buff, int count, MPI_Datatype datatype,
int source, int tag, MPI_Comm comm, MPI_Status *status)
```

前面几个变量的定义与 Send 函数中类似，MPI_Status *status 表示消息状态。接收函数返回时，将在这个参数指示的变量中存放实际接收消息的状态信息，包括消息的源进程标识，消息标签，包含的数据项个数等。

3 操作方法和实验步骤

3.1 自动向量化

修改 Makefile 文件如下：

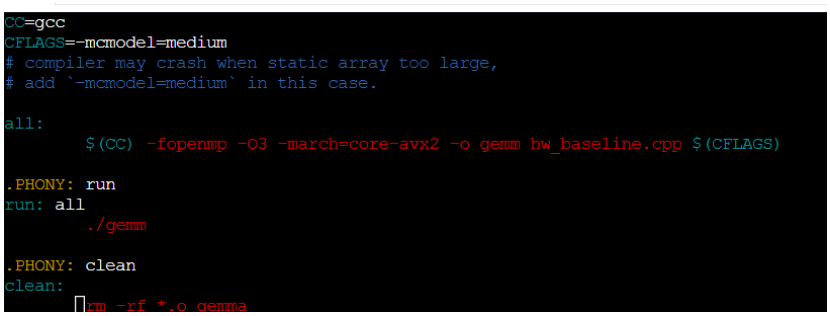

图 2: Makefile 文件截图

采用 O3 级优化，自动向量化的指令集取 avx2。

3.2 Blocking

分块后的局部代码截图如下：

图 3: Blocking 截图

此处采用的分块是 1x4 分块。

3.3 OpenMp 线程数修改

在 Makefile 文件中加入-fopenmp 以开启 openmp，由于集群中每个环境的实际线程数在 8 左右，修改并行的线程数为 8：

```
1 #pragma omp parallel for num_threads(8)
```

4 实验数据记录

因为最后也没有成功写出用 mpi 分布式优化的代码，下面的实验数据是在单机上进行测试得到的：

```
root@3dclbctkc9t8a-0:/tmp/chenjh08/lab4_test_thy/test3/test1# ./gemm
Performance : 16187.814353 Mops
Validation passed.
root@3dclbctkc9t8a-0:/tmp/chenjh08/lab4_test_thy/test3/test1#
root@3dclbctkc9t8a-0:/tmp/chenjh08/lab4_test_thy/test3/test1# ./gemm
Performance : 38024.123704 Mops
Validation passed.
root@3dclbctkc9t8a-0:/tmp/chenjh08/lab4_test_thy/test3/test1# ./gemm
Performance : 35856.120029 Mops
Validation passed.
root@3dclbctkc9t8a-0:/tmp/chenjh08/lab4_test_thy/test3/test1# ./gemm
Performance : 14843.499395 Mops
Validation passed.
root@3dclbctkc9t8a-0:/tmp/chenjh08/lab4_test_thy/test3/test1# ./gemm
Performance : 37543.466126 Mops
Validation passed.
root@3dclbctkc9t8a-0:/tmp/chenjh08/lab4_test_thy/test3/test1# ./gemm
Performance : 14580.089773 Mops
Validation passed.
root@3dclbctkc9t8a-0:/tmp/chenjh08/lab4_test_thy/test3/test1# ./gemm
Performance : 38341.779697 Mops
Validation passed.
root@3dclbctkc9t8a-0:/tmp/chenjh08/lab4_test_thy/test3/test1# ./gemm
Performance : 16946.670183 Mops
Validation passed.
root@3dclbctkc9t8a-0:/tmp/chenjh08/lab4_test_thy/test3/test1# ./gemm
Performance : 16380.759948 Mops
Validation passed.
root@3dclbctkc9t8a-0:/tmp/chenjh08/lab4_test_thy/test3/test1#
```

图 4: mops 结果截图

测试时取的 N 的值为 1000，Mpos 值在一万到三万之间波动，平均约两万。

取 N 的值为 10001 时，一般 2 分 40 秒左右跑出结果，Mops 值为一万左右。