南京理工大学计算机科学与技术学院

高性能计算实验报告

班　　级 _____

学　　号 _____

姓　　名 _____

教　　师 李翔宇

目录

1 矩阵运算实验

1.1 实验环境说明

- 华为鲲鹏云主机、openEuler 20.03 操作系统；
- 安装 mpich-3.3.2.tar.gz；
- 安装 OpenBLAS-0.3.8.tar.gz；
- 每套实验环境可供 1 名学员上机操作。

1.2 实验目标和实验过程

1.2.1 实验目标

本实验指导书通过在华为鲲鹏云服务器上，编译运行矩阵乘法、池化、卷积等程序。完成实验操作后，读者会掌握简单的程序编写以及编译运行，集群 MPI 并行计算的配置以及加深对并行计算的了解。

1.2.2 实验过程

特别说明：

- 环境配置步骤略
- 以下步骤在四个主机上均要执行

1. 执行以下命令，创建 matrix 目录存放该程序的所有文件，并进入 matrix 目录

```
mkdir /home/ky4/matrix
cd /home/ky4/matrix
```

2. 执行以下命令，创建矩阵乘法源码 gemm.cpp

```
vim gemm.cpp
```

代码如下：

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <iostream>
#include "mpi.h"
// #include <cblas.h>

using namespace std;

// void CheckStatus(MPI_Status &status) {
//     if (status.MPI_ERROR != MPI_SUCCESS) {
```

```
//                cout << MPI::Get_error_class(status.MPI_ERROR);
//        }
// }

void randMat(int rows, int cols, float *&Mat) {
   Mat = new float[rows * cols];
   for (int i = 0; i < rows; i++)
      for (int j = 0; j < cols; j++)
         Mat[i * cols + j] = 1.0;
}

void openmp_sgemm(int m, int n, int k, float *&leftMat, float *&rightMat,
                           float *&resultMat) {
   // rightMat is transposed
#pragma omp parallel for
   for (int row = 0; row < m; row++) {
      for (int col = 0; col < k; col++) {
         resultMat[row * k + col] = 0.0;
         for (int i = 0; i < n; i++) {
            resultMat[row * k + col] +=
                  leftMat[row * n + i] * rightMat[col * n + i];
         }
      }
   }
   return;
}

void blas_sgemm(int m, int n, int k, float *&leftMat, float *&rightMat,
                         float *&resultMat) {
   // cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasTrans, m, k, n, 1.0, leftMat,
   // n, rightMat, n, 0.0, resultMat, k);
}

void mpi_sgemm(int m, int n, int k, float *&leftMat, float *&rightMat,
                        float *&resultMat, int rank, int worldsize, bool blas) {
   int rowBlock = sqrt(worldsize);
   if (rowBlock * rowBlock > worldsize)
      rowBlock -= 1;
   int colBlock = rowBlock;

   int rowStride = m / rowBlock;
   int colStride = k / colBlock;

   worldsize = rowBlock * colBlock; // we abandom some processes.
   // so best set process to a square number.

   float *res;

   if (rank == 0) {
```

```
    float *buf = new float[k * n];
    // transpose right Mat
    for (int r = 0; r < n; r++) {
      for (int c = 0; c < k; c++) {
        buf[c * n + r] = rightMat[r * k + c];
      }
    }

    for (int r = 0; r < k; r++) {
      for (int c = 0; c < n; c++) {
        rightMat[r * n + c] = buf[r * n + c];
      }
    }

    delete buf;

    MPI_Request sendRequest[2 * worldsize];
    MPI_Status status[2 * worldsize];
    for (int rowB = 0; rowB < rowBlock; rowB++) {
      for (int colB = 0; colB < colBlock; colB++) {
        rowStride = (rowB == rowBlock - 1) ? m - (rowBlock - 1) * (m / rowBlock)
                                           : m / rowBlock;
        colStride = (colB == colBlock - 1) ? k - (colBlock - 1) * (k / colBlock)
                                           : k / colBlock;
        int sendto = rowB * colBlock + colB;
        if (sendto == 0)
          continue;
        MPI_Isend(&leftMat[rowB * (m / rowBlock) * n], rowStride * n, MPI_FLOAT,
                  sendto, 0, MPI_COMM_WORLD, &sendRequest[sendto]);
        MPI_Isend(&rightMat[colB * (k / colBlock) * n], colStride * n,
                  MPI_FLOAT, sendto, 1, MPI_COMM_WORLD,
                  &sendRequest[sendto + worldsize]);
      }
    }
    for (int rowB = 0; rowB < rowBlock; rowB++) {
      for (int colB = 0; colB < colBlock; colB++) {
        int recvfrom = rowB * colBlock + colB;
        if (recvfrom == 0)
          continue;
        MPI_Wait(&sendRequest[recvfrom], &status[recvfrom]);
        MPI_Wait(&sendRequest[recvfrom + worldsize],
                 &status[recvfrom + worldsize]);
      }
    }
    res = new float[(m / rowBlock) * (k / colBlock)];
  } else {
    if (rank < worldsize) {
      MPI_Status status[2];
      rowStride = ((rank / colBlock) == rowBlock - 1)
```

```
                        ? m - (rowBlock - 1) * (m / rowBlock)
                        : m / rowBlock;
      colStride = ((rank % colBlock) == colBlock - 1)
                        ? k - (colBlock - 1) * (k / colBlock)
                        : k / colBlock;
      if (rank != 0) {
         leftMat = new float[rowStride * n];
         rightMat = new float[colStride * n];
      }
      if (rank != 0) {
         MPI_Recv(leftMat, rowStride * n, MPI_FLOAT, 0, 0, MPI_COMM_WORLD,
                  &status[0]);
         MPI_Recv(rightMat, colStride * n, MPI_FLOAT, 0, 1, MPI_COMM_WORLD,
                  &status[1]);
      }
      res = new float[rowStride * colStride];
   }
}
MPI_Barrier(MPI_COMM_WORLD);

if (rank < worldsize) {
   rowStride = ((rank / colBlock) == rowBlock - 1)
                     ? m - (rowBlock - 1) * (m / rowBlock)
                     : m / rowBlock;
   colStride = ((rank % colBlock) == colBlock - 1)
                     ? k - (colBlock - 1) * (k / colBlock)
                     : k / colBlock;
   if (!blas)
      openmp_sgemm(rowStride, n, colStride, leftMat, rightMat, res);
   else
      blas_sgemm(rowStride, n, colStride, leftMat, rightMat, res);
}
MPI_Barrier(MPI_COMM_WORLD);

if (rank == 0) {
   MPI_Status status;
   float *buf = new float[(m - (rowBlock - 1) * (m / rowBlock)) *
                          (k - (colBlock - 1) * (k / colBlock))];
   float *temp_res;
   for (int rowB = 0; rowB < rowBlock; rowB++) {
      for (int colB = 0; colB < colBlock; colB++) {
         rowStride = (rowB == rowBlock - 1) ? m - (rowBlock - 1) * (m / rowBlock)
                                            : m / rowBlock;
         colStride = (colB == colBlock - 1) ? k - (colBlock - 1) * (k / colBlock)
                                            : k / colBlock;
         int recvfrom = rowB * colBlock + colB;
         if (recvfrom != 0) {
            temp_res = buf;
            MPI_Recv(temp_res, rowStride * colStride, MPI_FLOAT, recvfrom, 0,
```

```
                        MPI_COMM_WORLD, &status);
            } else {
                temp_res = res;
            }
            for (int r = 0; r < rowStride; r++)
                for (int c = 0; c < colStride; c++)
                    resultMat[rowB * (m / rowBlock) * k + colB * (k / colBlock) +
                              r * k + c] = temp_res[r * colStride + c];
        }
    }
  } else {
    rowStride = ((rank / colBlock) == rowBlock - 1)
                        ? m - (rowBlock - 1) * (m / rowBlock)
                        : m / rowBlock;
    colStride = ((rank % colBlock) == colBlock - 1)
                        ? k - (colBlock - 1) * (k / colBlock)
                        : k / colBlock;
    if (rank < worldsize)
      MPI_Send(res, rowStride * colStride, MPI_FLOAT, 0, 0, MPI_COMM_WORLD);
  }
  MPI_Barrier(MPI_COMM_WORLD);

  return;
}

int main(int argc, char *argv[]) {
  if (argc != 5) {
    cout << "Usage: " << argv[0] << " M N K use-blas\n";
    exit(-1);
  }

  int rank;
  int worldSize;
  MPI_Init(&argc, &argv);

  MPI_Comm_size(MPI_COMM_WORLD, &worldSize);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  int m = atoi(argv[1]);
  int n = atoi(argv[2]);
  int k = atoi(argv[3]);
  int blas = atoi(argv[4]);

  float *leftMat, *rightMat, *resMat;

  struct timeval start, stop;
  if (rank == 0) {
    randMat(m, n, leftMat);
    randMat(n, k, rightMat);
```

```
    randMat(m, k, resMat);
  }
  gettimeofday(&start, NULL);
  mpi_sgemm(m, n, k, leftMat, rightMat, resMat, rank, worldSize, blas);
  gettimeofday(&stop, NULL);
  if (rank == 0) {
    cout << "mpi matmul: "
         << (stop.tv_sec - start.tv_sec) * 1000.0 +
                (stop.tv_usec - start.tv_usec) / 1000.0
         << " ms" << endl;

    for (int i = 0; i < m; i++) {
      for (int j = 0; j < k; j++)
        if (int(resMat[i * k + j]) != n) {
          cout << resMat[i * k + j] << "error\n";
          exit(-1);
        }
      // cout << resMat[i * k + j] << ' ';
      // cout << endl;
    }
  }
  MPI_Finalize();
}
```

3. 执行以下命令，创建卷积操作源码 conv.cpp

```
vim conv.cpp
```

代码如下：

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <string.h>

#include <iostream>

#include "mpi.h"

#include <cblas.h>
#include <assert.h>

using namespace std;

void randMat(int rows, int cols, float *&Mat) {
    Mat = new float[rows * cols];
    for (int i = 0; i < rows; i++)
        for (int j = 0; j < cols; j++) Mat[i * cols + j] = 1.0;
}

int get_steps(int kernel, int step, int len) {
```

```
    if (kernel > len) return 0;
    return (len - kernel) / step + 1;
}


inline void img2col_conv_kernel(int leftAnchorX, int leftAnchorY, int rightAnchorX,
                                int rightAnchorY, const int xKernel, const int yKernel, const int
xStep,
                                const int yStep, float *&img, float *&kernel, float *&conv) {

    int imgRows = rightAnchorX - leftAnchorX,
        imgCols = rightAnchorY - leftAnchorY;

    int convRows = get_steps(xKernel, xStep, imgRows);
    int convCols = get_steps(yKernel, yStep, imgCols);

    float *flattenImg = new float[convRows * convCols * xKernel * yKernel];

// #pragma omp parallel for
//      for (int i = leftAnchorX; i < rightAnchorX - xKernel; i += xStep) {
//          for (int r = 0; r < xKernel; r++) {
//              for (int j = leftAnchorY; j < rightAnchorY - yKernel; j += yStep) {
//                  int pos = (i - leftAnchorX)/xStep * convCols + (j - leftAnchorY) / yStep;
//                  memcpy(&flattenImg[pos * xKernel * yKernel + r * yKernel], &img[(i + r)
* imgCols + j], sizeof(float) * yKernel);
//              }
//          }
//      }

// #pragma omp parallel for
    // for (int i = 0; i < convRows * convCols; i++) {
    //     conv[i] = 0.0;
    //     for (int j = 0; j < xKernel * yKernel; j++) {
    //         conv[i] += flattenImg[i*xKernel*yKernel + j] * kernel[j];
    //     }
    // }

    cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, convRows *
convCols, 1, xKernel * yKernel, 1.0, flattenImg, xKernel * yKernel, kernel, 1, 0.0, conv, 1);
    // cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasTrans, convRows * convCols,
1, xKernel * yKernel, 1.0, flattenImg, xKernel * yKernel, kernel, convRows * convCols,
0.0, conv, 1);

    delete flattenImg;

}

inline void naive_conv_kernel(int leftAnchorX, int leftAnchorY, int rightAnchorX,
                              int rightAnchorY, const int xKernel, const int yKernel, const int
xStep,
```

```
                                const int yStep, float *&img, float *&kernel, float *&conv) {
    int imgRows = rightAnchorX - leftAnchorX,
        imgCols = rightAnchorY - leftAnchorY;

    int convRows = get_steps(xKernel, xStep, imgRows);
    int convCols = get_steps(yKernel, yStep, imgCols);
#pragma omp parallel for
    for (int i = leftAnchorX; i < rightAnchorX - xKernel; i += xStep) {
        for (int j = leftAnchorY; j < rightAnchorY - yKernel; j += yStep) {
            // to[i / xStride * to_n + j] = 0.0;
            int pos = (i - leftAnchorX)/xStep * convCols + (j - leftAnchorY) / yStep;
            conv[pos] = 0.0;
            for (int r = i; r < i + xKernel; r++)
                for (int c = j; c < j + yKernel; c++) {
                    conv[pos] += img[r * imgCols + c] *
                                 kernel[(r - i) * yKernel + (c - j)];
                }
        }
    }
}

void mpi_convolution(int m, int n, int xKernel, int yKernel, int xStep,
int yStep, float *&img, float *&kernel, float *&conv,
                     int rank, int worldsize, bool img2col) {
    const int total_xsteps = get_steps(xKernel, xStep, m);
    const int total_ysteps = get_steps(yKernel, yStep, n);
    const int xsteps_per_proc = total_xsteps / worldsize;
    const int last_xsteps = total_xsteps - xsteps_per_proc * (worldsize - 1);
    int steps;
    if (rank == 0) {
        MPI_Request *sendRequest = new MPI_Request[worldsize];
        MPI_Status *status = new MPI_Status[worldsize];
        for (int i = 1; i < worldsize; i++) {
            steps = (i == worldsize - 1) ? last_xsteps : xsteps_per_proc;
            MPI_Isend(&img[i * xsteps_per_proc * xStep * n],
                      (steps * xStep + xKernel - xStep) * n, MPI_FLOAT, i, 0,
                      MPI_COMM_WORLD, &sendRequest[i]);
        }
        for (int i = 1; i < worldsize; i++) {
            MPI_Wait(&sendRequest[i], &status[i]);
        }
        delete sendRequest;
        delete status;
    } else {
        MPI_Status status;
        steps = (rank == worldsize - 1) ? last_xsteps : xsteps_per_proc;
        img = new float[(steps * xStep + xKernel - xStep) * n];
        MPI_Recv(img, (steps * xStep + xKernel - xStep) * n, MPI_FLOAT, 0, 0,
                 MPI_COMM_WORLD, &status);
```

```
        conv = new float[steps * total_ysteps];
    }
    MPI_Barrier(MPI_COMM_WORLD);

    steps = (rank == worldsize - 1) ? last_xsteps : xsteps_per_proc;

    if (img2col)
        img2col_conv_kernel(0, 0, steps * xStep + xKernel - xStep, n, xKernel, yKernel,
                            xStep, yStep, img, kernel, conv);
    else
        naive_conv_kernel(0, 0, steps * xStep + xKernel - xStep, n, xKernel, yKernel,
                            xStep, yStep, img, kernel, conv);

    MPI_Barrier(MPI_COMM_WORLD);

    if (rank == 0) {
        MPI_Status status;
        for (int i = 1; i < worldsize; i++) {
            steps = (i == worldsize - 1) ? last_xsteps : xsteps_per_proc;
            MPI_Recv(&conv[i * xsteps_per_proc * total_ysteps],
                        steps * total_ysteps, MPI_FLOAT, i, 0, MPI_COMM_WORLD,
                        &status);
        }
    } else {
        MPI_Send(conv, steps * total_ysteps, MPI_FLOAT, 0, 0, MPI_COMM_WORLD);
    }
    MPI_Barrier(MPI_COMM_WORLD);

    return;
}

int main(int argc, char *argv[]) {
    if (argc != 4) {
        cout << "Usage: " << argv[0] << " M N enabled-img2col";
        exit(-1);
    }

    int rank;
    int worldSize;
    MPI_Init(&argc, &argv);

    MPI_Comm_size(MPI_COMM_WORLD, &worldSize);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    int m = atoi(argv[1]);
    int n = atoi(argv[2]);
    int img2col = atoi(argv[3]);

    int xKernel = 3, yKernel = 3;
```

```
    int xStep = 1, yStep = 1;

    float *Img, *Conv;

    struct timeval start, stop;
    if (rank == 0) {
        randMat(m, n, Img);
        randMat(get_steps(xKernel, xStep, m), get_steps(yKernel, yStep, n),
                Conv);
    }

    float *Kernel = new float[xKernel*yKernel];
    for (int i = 0; i < xKernel*yKernel; i++) Kernel[i] = 1.0;

    gettimeofday(&start, NULL);
    mpi_convolution(m, n, xKernel, yKernel, xStep, yStep, Img, Kernel, Conv,
                    rank, worldSize, img2col);
    gettimeofday(&stop, NULL);

    if (rank == 0) {
        cout << "mpi convolution: "
             << (stop.tv_sec - start.tv_sec) * 1000.0 +
                (stop.tv_usec - start.tv_usec) / 1000.0
             << " ms" << endl;

        for (int i = 0; i < min(10, m); i++) {
            for (int j = 0; j < min(10, n); j++)
                cout << Conv[i * n + j] << ' ';
            cout << endl;
        }
    }
    delete Img;
    delete Conv;
    MPI_Finalize();
}
```

4. 执行以下命令，创建卷积操作源码 pooling.cpp

```
vim pooling.cpp
```

代码如下：

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>

#include <iostream>

#include "mpi.h"

using namespace std;
```

```
void randMat(int rows, int cols, float *&Mat) {
	Mat = new float[rows * cols];
	for (int i = 0; i < rows; i++)
		for (int j = 0; j < cols; j++) Mat[i * cols + j] = 1.0;
}

inline void pooling_kernel(int leftAnchorX, int leftAnchorY, int rightAnchorX,
                           int rightAnchorY, int xStride, int yStride,
                           int from_m, int from_n, int to_m, int to_n,
                           float *&from, float *&to) {
	if ((rightAnchorX - leftAnchorX) % xStride != 0 ||
		(rightAnchorY - leftAnchorY) % yStride != 0)
		exit(-1);
	if (leftAnchorX % xStride != 0 || leftAnchorY % yStride != 0) exit(-2);

	for (int i = leftAnchorX; i < rightAnchorX; i += xStride) {
		for (int j = leftAnchorY; j < rightAnchorY; j += yStride) {
			// to[i / xStride * to_n + j] = 0.0;
			float temp = to[i / xStride * to_n + j / yStride];
			for (int r = i; r < i + xStride; r++)
				for (int c = j; c < j + yStride; c++) {
					temp = max(temp, from[r * from_n + c]);
				}
			to[i / xStride * to_n + j / yStride] = temp;
		}
	}
}

void mpi_pooling(int m, int n, int xstride, int ystride, float *&mat,
                 float *&res, int rank, int worldsize) {
	if (m % xstride || n % ystride) {
		cout << "matrix size and stride do not match \n";
		return;
	}
	const int xstrides_per_proc = (m / xstride) / worldsize;
	int strides;
	if (rank == 0) {
		MPI_Request *sendRequest = new MPI_Request[worldsize];
		MPI_Status *status = new MPI_Status[worldsize];
		for (int i = 1; i < worldsize; i++) {
			strides = (i < worldsize - 1)
			          ? xstrides_per_proc
			          : (m / xstride) - xstrides_per_proc * (worldsize - 1);
			MPI_Isend(&mat[i * xstrides_per_proc * xstride * n],
			          strides * xstride * n, MPI_FLOAT, i, 0, MPI_COMM_WORLD,
			          &sendRequest[i]);
		}
		for (int i = 1; i < worldsize; i++) {
```

```
                MPI_Wait(&sendRequest[i], &status[i]);
            }
            delete sendRequest;
            delete status;
        } else {
            MPI_Status status;
            strides = (rank < worldsize - 1)
                            ? xstrides_per_proc
                            : (m / xstride) - xstrides_per_proc * (worldsize - 1);
            mat = new float[strides * xstride * n];
            MPI_Recv(mat, strides * xstride * n, MPI_FLOAT, 0, 0, MPI_COMM_WORLD,
                    &status);
            res = new float[strides * (n / ystride)];
        }
        MPI_Barrier(MPI_COMM_WORLD);

        strides = (rank < worldsize - 1)
                        ? xstrides_per_proc
                        : (m / xstride) - xstrides_per_proc * (worldsize - 1);

        // pooling_kernel(rank * xstrides_per_proc * xstride, 0,
        //                 (rank * xstrides_per_proc + strides) * xstride, n,
        //                 xstride, ystride, m, n, strides, n / ystride, mat, res);

        pooling_kernel(0, 0, strides * xstride, n, xstride, ystride,
                        strides * xstride, n, strides, n / ystride, mat, res);

        MPI_Barrier(MPI_COMM_WORLD);

        if (rank == 0) {
            MPI_Status status;
            for (int i = 1; i < worldsize; i++) {
                strides = (i < worldsize - 1)
                                ? xstrides_per_proc
                                : (m / xstride) - xstrides_per_proc * (worldsize - 1);
                MPI_Recv(&res[i * xstrides_per_proc * (n / ystride)],
                            strides * (n / ystride), MPI_FLOAT, i, 0, MPI_COMM_WORLD,
                            &status);
            }
        } else {
            MPI_Send(res, strides * (n / ystride), MPI_FLOAT, 0, 0, MPI_COMM_WORLD);
        }
        MPI_Barrier(MPI_COMM_WORLD);

        return;
}

int main(int argc, char *argv[]) {
        if (argc != 3) {
```

```
        cout << "Usage: " << argv[0] << " M N";
        exit(-1);
    }

    int rank;
    int worldSize;
    MPI_Init(&argc, &argv);

    MPI_Comm_size(MPI_COMM_WORLD, &worldSize);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    int m = atoi(argv[1]);
    int n = atoi(argv[2]);

    int xstride = 4, ystride = 4;

    float *Mat, *resMat;

    struct timeval start, stop;
    if (rank == 0) {
        randMat(m, n, Mat);
        randMat(m / xstride, n / ystride, resMat);
    }
    gettimeofday(&start, NULL);
    mpi_pooling(m, n, xstride, ystride, Mat, resMat, rank, worldSize);
    gettimeofday(&stop, NULL);
    if (rank == 0) {
        cout << "mpi pooling: "
             << (stop.tv_sec - start.tv_sec) * 1000 * 1000L +
                    (stop.tv_usec - start.tv_usec)
             << endl;

        // for (int i = 0; i < min(10, m); i++) {
        //     for (int j = 0; j < min(10, n); j++)
        //         cout << Mat[i * k + j] << ' ';
        //     cout << endl;
        // }
    }
    delete Mat;
    delete resMat;
    MPI_Finalize();
}
```

5. 执行以下命令，创建 Makefile

```
vim Makefile
```

代码如下：

```
CC = mpic++
CCFLAGS = -O2 -fopenmp
LDFLAGS = -lopenblas
```

```
all:   gemm conv pooling
gemm: gemm.cpp
	${CC} ${CCFLAGS} gemm.cpp -o gemm ${LDFLAGS}
conv: conv.cpp
	${CC} ${CCFLAGS} conv.cpp -o conv ${LDFLAGS}
pooling: pooling.cpp
	${CC} ${CCFLAGS} pooling.cpp -o pooling ${LDFLAGS}
clean:
	rm gemm conv pooling
```

6. 执行以下命令，进行编译

```
make
```

7. 执行以下命令，建立主机配置文件

```
vim /home/ky4/matrix/hostfile
```

添加如下内容：

```
ecs-hpc-0001:2
ecs-hpc-0002:2
ecs-hpc-0003:2
ecs-hpc-0004:2
```

8. 执行以下命令，编写 run.sh 脚本

```
vim run.sh
```

内容如下：

```
app=${1}

if [ ${app} = "gemm" ]; then
	mpirun --hostfile hostfile -np ${2} ./gemm 4024 4024 4024 0
fi

if [ ${app} = "conv" ]; then
	mpirun --hostfile hostfile -np ${2} ./conv 4096 4096 ${3}
fi

if [ ${app} = "pooling" ]; then
	mpirun --hostfile hostfile -np ${2} ./pooling 1024 1024
fi
```

1.3 实验结果

特别说明:

- 以下命令在一台主机上执行即可

① 分别执行以下命令，查看矩阵乘法运行结果（1-8 数字表示启动处理的进程数量）

```
bash run.sh gemm 1
bash run.sh gemm 2
```

```
bash run.sh gemm 3
bash run.sh gemm 4
bash run.sh gemm 5
bash run.sh gemm 6
bash run.sh gemm 7
bash run.sh gemm 8
```

结果如下：

```
[ky4@ecs-hpc-0001 matrix]$ bash run.sh gemm 1
mpi matmul: 58144.3 ms
[ky4@ecs-hpc-0001 matrix]$ bash run.sh gemm 2
mpi matmul: 142127 ms
[ky4@ecs-hpc-0001 matrix]$ bash run.sh gemm 3

Authorized users only. All activities may be monitored and reported.
mpi matmul: 120500 ms
[ky4@ecs-hpc-0001 matrix]$ bash run.sh gemm 4

Authorized users only. All activities may be monitored and reported.
mpi matmul: 36278.5 ms
[ky4@ecs-hpc-0001 matrix]$ bash run.sh gemm 5

Authorized users only. All activities may be monitored and reported.

Authorized users only. All activities may be monitored and reported.
mpi matmul: 35602.8 ms
[ky4@ecs-hpc-0001 matrix]$ bash run.sh gemm 6

Authorized users only. All activities may be monitored and reported.

Authorized users only. All activities may be monitored and reported.
mpi matmul: 35497.2 ms
[ky4@ecs-hpc-0001 matrix]$ bash run.sh gemm 7

Authorized users only. All activities may be monitored and reported.

Authorized users only. All activities may be monitored and reported.

Authorized users only. All activities may be monitored and reported.
mpi matmul: 35723.9 ms
[ky4@ecs-hpc-0001 matrix]$ bash run.sh gemm 8

Authorized users only. All activities may be monitored and reported.

Authorized users only. All activities may be monitored and reported.

Authorized users only. All activities may be monitored and reported.
mpi matmul: 36344.8 ms
[ky4@ecs-hpc-0001 matrix]$
```

② 分别执行以下命令，查看卷积运行结果（conv 后面的 1-8 数字表示启动处理的进程数量。最后面的 0 表示 vanilla convolution kernel，1 表示 img2col kernel）

```
bash run.sh conv 1 0
bash run.sh conv 2 0
bash run.sh conv 3 0
bash run.sh conv 4 0
bash run.sh conv 5 0
bash run.sh conv 6 0
bash run.sh conv 7 0
bash run.sh conv 8 0

bash run.sh conv 1 1
bash run.sh conv 2 1
bash run.sh conv 3 1
bash run.sh conv 4 1
bash run.sh conv 5 1
```

```
bash run.sh conv 6 1
bash run.sh conv 7 1
bash run.sh conv 8 1
```

结果如下：

```
[ky4@ecs-hpc-0001 matrix]$ bash run.sh conv 1 0
mpi convolution: 399.316 ms
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
[ky4@ecs-hpc-0001 matrix]$ bash run.sh conv 2 0
mpi convolution: 4563.19 ms
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
[ky4@ecs-hpc-0001 matrix]$ bash run.sh conv 3 0

Authorized users only. All activities may be monitored and reported.
mpi convolution: 3188.59 ms
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
```

```
[ky4@ecs-hpc-0001 matrix]$ bash run.sh conv 4 0

Authorized users only. All activities may be monitored and reported.
mpi convolution: 2557.03 ms
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
[ky4@ecs-hpc-0001 matrix]$ bash run.sh conv 5 0

Authorized users only. All activities may be monitored and reported.

Authorized users only. All activities may be monitored and reported.
mpi convolution: 2312.15 ms
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
[ky4@ecs-hpc-0001 matrix]$ bash run.sh conv 6 0

Authorized users only. All activities may be monitored and reported.

Authorized users only. All activities may be monitored and reported.
mpi convolution: 1967.26 ms
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
```

```
[ky4@ecs-hpc-0001 matrix]$ bash run.sh conv 1 1
mpi convolution: 259.573 ms
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
[ky4@ecs-hpc-0001 matrix]$ bash run.sh conv 2 1
mpi convolution: 4399.01 ms
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
[ky4@ecs-hpc-0001 matrix]$ bash run.sh conv 3 1

Authorized users only. All activities may be monitored and reported.
mpi convolution: 3058.36 ms
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

```
[ky4@ecs-hpc-0001 matrix]$ bash run.sh conv 4 1

Authorized users only. All activities may be monitored and reported.
mpi convolution: 2601.16 ms
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
[ky4@ecs-hpc-0001 matrix]$ bash run.sh conv 5 1

Authorized users only. All activities may be monitored and reported.

Authorized users only. All activities may be monitored and reported.
mpi convolution: 2179.54 ms
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

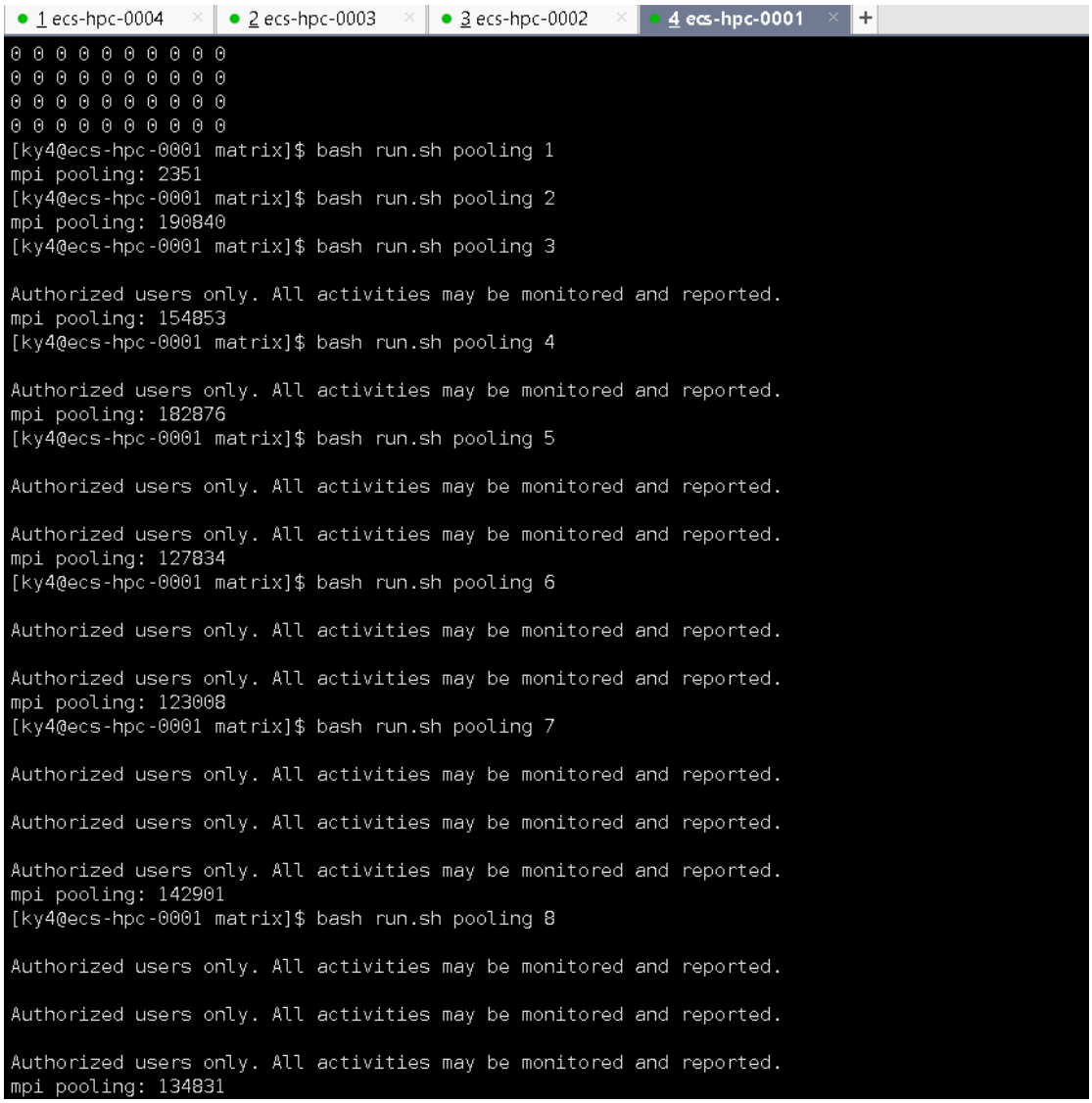
```
[ky4@ecs-hpc-0001 matrix]$ bash run.sh conv 6 1

Authorized users only. All activities may be monitored and reported.

Authorized users only. All activities may be monitored and reported.
mpi convolution: 2159.46 ms
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
[ky4@ecs-hpc-0001 matrix]$ bash run.sh conv 7 1

Authorized users only. All activities may be monitored and reported.

Authorized users only. All activities may be monitored and reported.

Authorized users only. All activities may be monitored and reported.
mpi convolution: 1647.9 ms
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

```
[ky4@ecs-hpc-0001 matrix]$ bash run.sh conv 8 1

Authorized users only. All activities may be monitored and reported.

Authorized users only. All activities may be monitored and reported.

Authorized users only. All activities may be monitored and reported.
mpi convolution: 1753.47 ms
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

③ 分别执行以下命令，查看池化运行结果（1-8 数字表示启动处理的进程数量）

```
bash run.sh pooling 1
bash run.sh pooling 2
bash run.sh pooling 3
bash run.sh pooling 4
bash run.sh pooling 5
bash run.sh pooling 6
bash run.sh pooling 7
bash run.sh pooling 8
```

结果如下：



综上：从整体大致可以看出，随着进程数量的增加，耗时减少。

2 蒙特卡罗算法实验

2.1 实验环境说明

- 华为鲲鹏云主机、openEuler 20.03 操作系统；
- 每套实验环境可供 1 名学员上机操作。

2.2 实验目标和实验过程

2.2.1 实验目标

本实验指导书通过在华为鲲鹏云服务器上，编译运行蒙特卡罗算法程序。完成实验操作后，读者会掌握蒙特卡罗算法程序的编写以及编译运行。

2.2.2 实验过程

特别说明：

- 环境配置步骤略
- 以下步骤在一个主机上执行

1. 执行以下命令，创建 MonteCarlo 目录存放该程序的所有文件，并进入 MonteCarlo 目录

```
mkdir /home/ky4/MonteCarlo
cd /home/ky4/MonteCarlo
```

2. 执行以下命令，创建 MonteCarlo 源码 MonteCarlo.cpp

```
vim MonteCarlo.cpp
```

代码如下：

```
#include "MonteCarlo.h"
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <time.h>
#include <iostream>
#include "omp.h"
using namespace std;
using namespace MC;

namespace MC {
namespace PI {
struct calPiArgs {
    int x, y;
};

struct calPiRets {
    bool hit;
};

void sampling(calPiArgs &arg, calPiRets &ret) {
    ret.hit = ((arg.x * arg.x + arg.y * arg.y) <= 1008 * 1008);
```

```
}

void yieldSamples(calPiArgs &args) {
	unsigned int seed = 1024;
	args.x = rand_r(&seed) % 1009;
	args.y = rand_r(&seed) % 1009;
}

void delSamples(calPiArgs &args) {}

}	// namespace PI

namespace Integral {

const int dim = 10;

struct variable {
	double *x;
	double y;
};

struct val {
	double y;
};

void f(variable &var, val &v) {
	v.y = 0.0;
	for (int i = 0; i < dim; i++) {
		v.y *= var.x[i];
	}
}

void sampling(variable &arg, val &ret) { f(arg, ret); }

void yieldSamples(variable &args) {
	args.x = new double[dim];
	unsigned int seed = 1024;
	for (int j = 0; j < dim; j++) {
		args.x[j] = (rand_r(&seed) % 1000) / 500.0;
	}
}

void delSamples(variable &args) { delete args.x; }

}	// namespace Integral

}	// namespace MC

int main(int argc, char *argv[]) {
```

```
	srand(time(NULL));
	if (argc != 3) {
		cout << "Usage: " << argv[0] << " thread-num sample-num";
		exit(-1);
	}

	int t = atoi(argv[1]);
	int n = atoi(argv[2]);
	double count = 0.0;
	omp_set_num_threads(t);

	/**
	 * Application 1:
	 * Calculate the value of \pi
	 */

	struct timeval start, stop;
	gettimeofday(&start, NULL);
	MC::PI::calPiRets *rets;	// = new calPiRets[n];
	rets = MC::RunMC<MC::PI::calPiArgs, MC::PI::calPiRets>(n);

#pragma omp parallel
	{
#pragma omp for reduction(+ : count)
		for (int i = 0; i < n; i++) {
			if (rets[i].hit) count += 1.0;
		}
	}
	// cout << "PI = " << double(count * 4.0 / n) << endl;
	gettimeofday(&stop, NULL);

	double elapse = (stop.tv_sec - start.tv_sec) * 1000 +
				(stop.tv_usec - start.tv_usec) / 1000;
	cout << elapse << " " << n << endl;

	/**
	 * Application 2:
	 * Integral
	 */
	gettimeofday(&start, NULL);
	MC::Integral::val *rets2;
	rets2 = MC::RunMC<MC::Integral::variable, MC::Integral::val>(n);

	count = 0;

#pragma omp parallel
	{
#pragma omp for reduction(+ : count)
		for (int i = 0; i < n; i++) {
```

```
                count += rets2[i].y;
            }
        }
        // cout << "Integral = " << double(count / n) << endl;
        gettimeofday(&stop, NULL);

        elapse = (stop.tv_sec - start.tv_sec) * 1000 +
                        (stop.tv_usec - start.tv_usec) / 1000;
        cout << elapse << " " << n << endl;

        return 0;
}
```

3. 执行以下命令，创建 MonteCarlo 头文件 MonteCarlo.h

```
vim MonteCarlo.cpp
```

代码如下：

```
#ifndef _MONTE_CARLO_HEADER_
#define _MONTE_CARLO_HEADER_
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <time.h>

#include <iostream>

#include "omp.h"

using namespace std;

namespace MC {

template <typename Args, typename Rets>
void sampling(Args &arg, Rets &ret);

template <typename Args>
void yieldSamples(Args &arg);

template <typename Args>
void delSamples(Args &arg);

template <typename Args, typename Rets>
Rets *RunMC(int sampleSize) {
        Args *args = new Args[sampleSize];
        Rets *rets = new Rets[sampleSize];

        // yieldSamples(sampleSize, args);

#pragma omp parallel for
```

```
    // {
    // #pragma omp for

    for (int i = 0; i < sampleSize; i++) {
        yieldSamples(args[i]);
        sampling(args[i], rets[i]);
        delSamples(args[i]);
    }
    // }
    delete args;
    return rets;
}

}  // namespace MC

#endif
```

4. 执行以下命令，创建 Makefile

```
vim Makefile
```

代码如下：

```
CC = g++
CCFLAGS = -I . -O2 -fopenmp
LDFLAGS = # -lopenblas

all: montecarlo

montecarlo: MonteCarlo.cpp
    ${CC} ${CCFLAGS} MonteCarlo.cpp -o montecarlo ${LDFLAGS}
clean:
    rm montecarlo
```

5. 执行以下命令，进行编译

```
make
```

6. 执行以下命令，建立主机配置文件

```
vim /home/ky4/MonteCarlo/hostfile
```

添加如下内容：

```
ecs-hpc-0001:2
ecs-hpc-0002:2
ecs-hpc-0003:2
ecs-hpc-0004:2
```

7. 执行以下命令，编写 run.sh 脚本

```
vim run.sh
```

内容如下：

```
app=${1}

if [ ${app} = "montecarlo" ]; then
```

```
    ./montecarlo ${2} 8000000
fi
```

2.3 实验结果

特别说明:

- 以下命令在一台主机上执行即可
- 主机 CPU 为 2 核

① 分别执行以下命令，查看蒙特卡洛算法运行结果（1-2 数字表示启动处理的进程数量）

```
bash run.sh montecarlo  1
bash run.sh montecarlo  2
```

结果如下：

```
[ky4@ecs-hpc-0001 MonteCarlo]$ g++ MonteCarlo.cpp -o montecarlo -fopenmp
[ky4@ecs-hpc-0001 MonteCarlo]$ bash run.sh montecarlo 1
196 8000000
1369 8000000
[ky4@ecs-hpc-0001 MonteCarlo]$ bash run.sh montecarlo 2
98 8000000
694 8000000
```

通过上述运行，可以看出蒙特卡罗算法程序已经在集群中并行运行起来。其中第一行输出代表的是蒙特卡罗算法统计耗时，第二行输出代表的是 Integral 统计耗时。可以得出结论，随着进程数量增加一倍，耗时减少一半左右。

3 快排算法实验

3.1 实验环境说明

- 华为鲲鹏云主机、openEuler 20.03 操作系统；
- 每套实验环境可供 1 名学员上机操作。

3.2 实验目标和实验过程

3.2.1 实验目标

本实验指导书通过在华为鲲鹏上，编译运行快排算法程序。完成实验操作后，读者会掌握快排算法程序的编写。

3.2.2 实验过程

特别说明：

- 环境配置步骤略
- 以下步骤在一个主机上执行

1. 执行以下命令，创建 quicksort 目录存放该程序的所有文件，并进入 quicksort 目录

```
mkdir /home/ky4/quicksort
cd /home/ky4/quicksort
```

2. 执行以下命令，创建 quicksort 源码 quicksort.cpp

```
vim quick_sort.cpp
```

代码如下：

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <time.h>

#include <iostream>

#include "omp.h"

using namespace std;

void QuickSort(int *&array, int len) {
    if (len <= 1) return;
    int pivot = array[len / 2];
    int left_ptr = 0;
    int right_ptr = len - 1;
    while (left_ptr <= right_ptr) {
        while (array[left_ptr] < pivot) left_ptr += 1;
        while (array[right_ptr] > pivot) right_ptr -= 1;
        if (left_ptr <= right_ptr) {
            swap(array[left_ptr], array[right_ptr]);
            left_ptr += 1;
            right_ptr -= 1;
        }
    }
    int *sub_array[] = {array, &(array[left_ptr])};
    int sub_len[] = {right_ptr + 1, len - left_ptr};

#pragma omp task default(none) firstprivate(sub_array, sub_len)
    { QuickSort(sub_array[0], sub_len[0]); }
#pragma omp task default(none) firstprivate(sub_array, sub_len)
    { QuickSort(sub_array[1], sub_len[1]); }
```

```
    // for (int i = 0; i < 2; i++) QuickSort(sub_array[i], sub_len[i]);
}

int main(int argc, char *argv[]) {
    srand(time(NULL));
    if (argc != 3) {
        cout << "Usage: " << argv[0] << " thread-num array-len\n";
        exit(-1);
    }
    int t = atoi(argv[1]);
    int n = atoi(argv[2]);
    int *array = new int[n];
    omp_set_num_threads(t);

    unsigned int seed = 1024;
#pragma omp parallel for
    for (int i = 0; i < n; i++) array[i] = rand_r(&seed);

    struct timeval start, stop;
    gettimeofday(&start, NULL);

#pragma omp parallel default(none) shared(array, n)
    {
#pragma omp single nowait
        { QuickSort(array, n); }
    }
    gettimeofday(&stop, NULL);

    double elapse = (stop.tv_sec - start.tv_sec) * 1000 +
                    (stop.tv_usec - start.tv_usec) / 1000;
    cout << elapse << " " << n << endl;

    for (int i = 0; i < n - 1; i++) {
        if (array[i] > array[i + 1]) {
            cerr << "quick sort fails! \n";
            break;
        }
    }
    return 0;
}
```

3. 执行以下命令，创建 Makefile

```
vim Makefile
```

代码如下：

```
CC = g++
CCFLAGS = -I . -O2 -fopenmp
LDFLAGS = # -lopenblas

all: quicksort
```

```
quicksort: quick_sort.cpp
	${CC} ${CCFLAGS} quick_sort.cpp -o quicksort ${LDFLAGS}
clean:
	rm quicksort
```

4. 执行以下命令，进行编译

```
make
```

5. 执行以下命令，建立主机配置文件

```
vim /home/ky4/quicksort /hostfile
```

添加如下内容：

```
ecs-hpc-0001:2
ecs-hpc-0002:2
ecs-hpc-0003:2
ecs-hpc-0004:2
```

6. 执行以下命令，编写 run.sh 脚本

```
vim run.sh
```

内容如下：

```
app=${1}

if [ ${app} = " quicksort " ]; then
	./ quicksort ${2} 8000000
fi
```

3.3 实验结果

特别说明：

- 以下命令在一台主机上执行即可
- 主机 CPU 为 2 核

② 分别执行以下命令，查看快排算法运行结果（1-2 数字表示启动处理的进程数量）

```
bash run.sh quicksort 1
bash run.sh quicksort 2
```

结果如下：

```
[ky4@ecs-hpc-0001 MonteCarlo]$ cd /home/ky4/
[ky4@ecs-hpc-0001 ~]$ ls
matrix  MonteCarlo  pagerank  quicksort
[ky4@ecs-hpc-0001 ~]$ cd quicksort
[ky4@ecs-hpc-0001 quicksort]$ g++ quick_sort.cpp -o quicksort -fopenmp
[ky4@ecs-hpc-0001 quicksort]$ bash run.sh quicksort 1
2667 8000000
[ky4@ecs-hpc-0001 quicksort]$ bash run.sh quicksort 2
1344 8000000
```

通过上述运行，可以看出快排算法程序已经在集群中并行运行起来。可以得出结论，随着进程数量增加一倍，耗时减少一半左右。

4 问题与解决办法

4.1 问题一

无法执行以下命令

```
sudo yum -y install gcc-gfortran
```

```
[ky4@ecs-hpc-0004 ~]$ sudo yum -y install gcc-gfortran
[sudo] password for ky4:
ky4 is not in the sudoers file.  This incident will be reported.
[ky4@ecs-hpc-0004 ~]$
```

解决：（四个主机均执行）

切换成 root 账号，修改/etc/sudoers 文件，添加如下内容：

```
ky4 ALL=(ALL) ALL
```

root@114.116.206.59 sudoers

```
26
27  ## Next comes the main part: which users can run what software on
28  ## which machines (the sudoers file can be shared between multiple
29  ## systems).
30  ## Syntax:
31  ##
32  ##  user    MACHINE=COMMANDS
33  ##
34  ## The COMMANDS section may have other options added to it.
35  ##
36  ## Allow root to run any commands anywhere
37  root    ALL=(ALL)   ALL
38
39  ## Allows people in group wheel to run all commands
40  %wheel  ALL=(ALL)   ALL
41  ky4 ALL=(ALL) ALL
```

就可以使用个人账户 ky4 执行此命令，如下图（之前已安装过了）：

```
[ky4@ecs-hpc-0004 ~]$ sudo yum -y install gcc-gfortran
[sudo] password for ky4:
Last metadata expiration check: 1:05:55 ago on Mon 06 Jun 2022 06:19:40 PM CST.
Package gcc-gfortran-7.3.0-20190804.h31.oe1.aarch64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
[ky4@ecs-hpc-0004 ~]$
```

4.2 问题二

无法执行以下命令

```
bash run.sh gemm 1
```

```
[ky4@ecs-hpc-0004 OpenBLAS-0.3.8]$ bash run.sh gemm 1
[mpiexec@ecs-hpc-0004] HYDU_parse_hostfile (utils/args/args.c:319): unable to open host file: hostfile
[mpiexec@ecs-hpc-0004] mfile_fn (ui/mpich/utils.c:336): error parsing hostfile
[mpiexec@ecs-hpc-0004] match_arg (utils/args/args.c:156): match handler returned error
[mpiexec@ecs-hpc-0004] HYDU_parse_array (utils/args/args.c:178): argument matching returned error
[mpiexec@ecs-hpc-0004] parse_args (ui/mpich/utils.c:1642): error parsing input array
[mpiexec@ecs-hpc-0004] HYD_uii_mpx_get_parameters (ui/mpich/utils.c:1694): unable to parse user arguments
[mpiexec@ecs-hpc-0004] main (ui/mpich/mpiexec.c:148): error parsing parameters
```

解决：(四个主机均执行)

当前路径错了（配置 OpenBLAS 环境后未切换回路径），把 run.sh 移动到 /home/ky4/matrix，并把当前路径转过去，重新执行

```
[ky4@ecs-hpc-0001 OpenBLAS-0.3.8]$ mv run.sh  /home/ky4/matrix/run.sh
[ky4@ecs-hpc-0001 OpenBLAS-0.3.8]$ cd /home/ky4/matrix
[ky4@ecs-hpc-0001 matrix]$ cd /home/ky4/matrix
[ky4@ecs-hpc-0001 matrix]$ bash run.sh gemm 1
```

4.3 问题三

继承问题二，依然无法执行

```
[ky4@ecs-hpc-0001 matrix]$ bash run.sh gemm 1
[proxy:0:0@ecs-hpc-0001] HYDU_create_process (utils/launch/launch.c:74): execvp error on file ./gemm (No such file or directory)
[ky4@ecs-hpc-0001 matrix]$ list
-bash: list: command not found
[ky4@ecs-hpc-0001 matrix]$ lm
-bash: lm: command not found
[ky4@ecs-hpc-0001 matrix]$ ls
conv.cpp  gemm.cpp  hostfile  Makefile  OpenBLAS-0.3.8  pooling.cpp  run.sh  v0.3.8.tar.gz
```

解决：(四个主机均执行)

重新编译，执行以下命令:

```
make
```

如下图（之前已 make 过了）：

```
[ky4@ecs-hpc-0001 matrix]$ make
make: Nothing to be done for 'all'.
[ky4@ecs-hpc-0001 matrix]$ ls
conv  conv.cpp  gemm  gemm.cpp  hostfile  Makefile  OpenBLAS-0.3.8  pooling  pooling.cpp  run.sh  v0.3.8.tar.gz
```

4.4 问题四

无法执行以下命令，即无法与其他主机连接

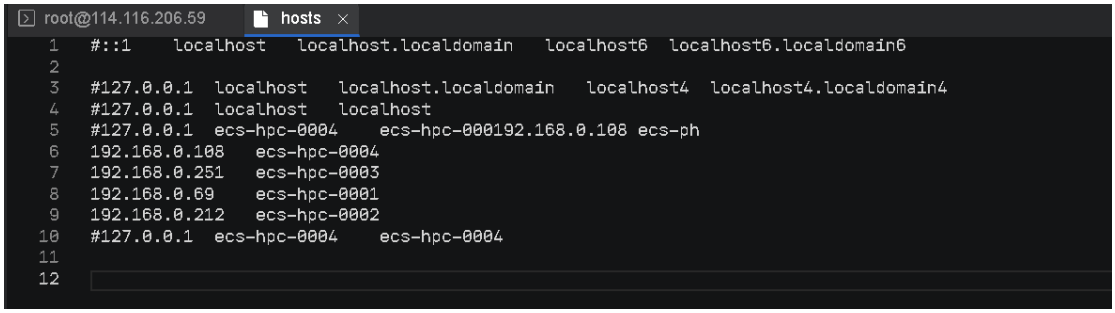
```
bash run.sh gemm 3
```

```
[ky4@ecs-hpc-0001 matrix]$ bash run.sh gemm 3

Authorized users only. All activities may be monitored and reported.
Fatal error in PMPI_Barrier: Unknown error class, error stack:
PMPI_Barrier(289)..................: MPI_Barrier(comm=MPI_COMM_WORLD) failed
PMPI_Barrier(275)..................:
MPIR_Barrier_impl(175).............:
MPIR_Barrier_intra_auto(110).......:
MPIR_Barrier_intra_smp(43).........:
MPIR_Barrier_impl(175).............:
MPIR_Barrier_intra_auto(110).......:
MPIR_Barrier_intra_dissemination(49):
MPIDU_Complete_posted_with_error(1137): Process failed
[ky4@ecs-hpc-0001 matrix]$
```

解决：(四个主机均执行)

发现四个主机的/etc/hosts 文件均有以下内容（不知道为什么它自己加上去了），将其注释掉

```
127.0.0.1       ecs-hpc-0004       ecs-hpc-0004
```


root@114.116.206.59
hosts
#::1 localhost localhost.localdomain localhost6 localhost6.localdomain6
#127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4
#127.0.0.1 localhost localhost
#127.0.0.1 ecs-hpc-0004 ecs-hpc-000192.168.0.108 ecs-ph
192.168.0.108 ecs-hpc-0004
192.168.0.251 ecs-hpc-0003
192.168.0.69 ecs-hpc-0001
192.168.0.212 ecs-hpc-0002
#127.0.0.1 ecs-hpc-0004 ecs-hpc-0004