

Lab 1: FFT

实验目标

输入一个复数数组 `in`，对其做 FFT 后输出到 `out` 数组中，`n` 为数组长度。

```
void student_fft(int n, const double complex* in, double complex* out)
```

框架代码

实验文件可在 elearning 上下载, 其中包括 `fft.c`, `omp_sample.c` 和 `makefile` 三个文件

在 `fft.c` 中, 我们给出了一个框架, 其中包括测试代码以及 DFT 和 FFT 的简单串行实现, 你要做的是将 `student_fft` 中的 `naive_fft` 替换成自己实现的并行 FFT 代码。

在 `omp_sample.c` 中, 我们给出了一些 `openmp` 的使用样例, 之前没接触过的同学可以参考。

运行方式

1. 登陆服务器

我们为大家准备了一个24核的服务器, 进入校内网后 (校外需要VPN), 使用命令行或 SSH 相关工具登入服务器。命令行 SSH 指令:

```
ssh user[your-student-ID]@10.20.26.203 # e.g. ssh user20307130000@10.20.26.203
```

初始密码为学号。

由于我们的服务器资源有限且程序对于性能十分敏感, 因此禁止大家使用 `vscode` 或 `clion` 等软件远程连接服务器!!! 如果发现, 我们将会杀掉你的程序。

因此你应该在本地写好代码, 并确保能够编译运行且结果正确后再将代码上传至服务器运行。

2. 本地运行

编译:

本地编译代码需要安装 `gcc` 和 `make` 工具, 使用 windows 系统的同学可以在 elearning 上下载 `mingw64.zip` 压缩包, 解压后将 `bin` 文件夹路径加入到环境变量。比如将压缩包解压到 `C:\mingw64` 后, 可以在环境变量中新建一条 `C:\mingw64\bin`, 按下确定后在终端中运行 `gcc -v` 指令, 出现如下信息, 即为安装成功

```
PS C:\Users\Adam> gcc -v
Using built-in specs.
COLLECT_GCC=C:\mingw64\bin\gcc.exe
COLLECT_LTO_WRAPPER=C:/mingw64/bin/./libexec/gcc/x86_64-w64-mingw32/8.1.0/lto-wrapper.exe
Target: x86_64-w64-mingw32
Configured with: ../../src/gcc-8.1.0/configure --host=x86_64-w64-mingw32 --build=x86_64-w64-mingw32 --target=x86_64-w64-mingw32 --prefix=/mingw64 --with-sysroot=/c/mingw810/x86_64-810-posix-seh-rt_v6-rev0/mingw64 --enable-shared --enable-static --disable-multilib --enable-languages=c,c++,fortran,lto --enable-libstdcxx-time=yes --enable-threads=posix --enable-libgomp --enable-libatomic --enable-lto --enable-graphite --enable-checking=release --enable-fully-dynamic-string --enable-version-specific-runtime-libs --disable-libstdcxx-pch --disable-libstdcxx-debug --enable-bootstrap --disable-rpath --disable-win32-registry --disable-nls --disable-werror --disable-symvers --with-gnu-as --with-gnu-ld --with-arch=nocona --with-tune=core2 --with-libiconv --with-system-zlib --with-gmp=/c/mingw810/prerequisites/x86_64-w64-mingw32-static --with-mpfr=/c/mingw810/prerequisites/x86_64-w64-mingw32-static --with-mpc=/c/mingw810/prerequisites/x86_64-w64-mingw32-static --with-isl=/c/mingw810/prerequisites/x86_64-w64-mingw32-static --with-pkgversion='x86_64-posix-seh-rev0, Built by MinGW-W64 project' --with-bugurl=https://sourceforge.net/projects/mingw-w64 CFLAGS='-O2 -pipe -fno-ident -I/c/mingw810/x86_64-810-posix-seh-rt_v6-rev0/mingw64/opt/include -I/c/mingw810/prerequisites/x86_64-zlib-static/include -I/c/mingw810/prerequisites/x86_64-w64-mingw32-static/include' CXXFLAGS='-O2 -pipe -fno-ident -I/c/mingw810/x86_64-810-posix-seh-rt_v6-rev0/mingw64/opt/include -I/c/mingw810/prerequisites/x86_64-zlib-static/include -I/c/mingw810/prerequisites/x86_64-w64-mingw32-static/include' CPPFLAGS='-I/c/mingw810/x86_64-810-posix-seh-rt_v6-rev0/mingw64/opt/include -I/c/mingw810/prerequisites/x86_64-zlib-static/include -I/c/mingw810/prerequisites/x86_64-w64-mingw32-static/include' LDFLAGS='-pipe -fno-ident -L/c/mingw810/x86_64-810-posix-seh-rt_v6-rev0/mingw64/opt/lib -L/c/mingw810/prerequisites/x86_64-zlib-static/lib -L/c/mingw810/prerequisites/x86_64-w64-mingw32-static/lib'
Thread model: posix
gcc version 8.1.0 (x86_64-posix-seh-rev0, Built by MinGW-W64 project)
PS C:\Users\Adam>
```

安装 `make` 之后可以直接使用命令进行编译:

```
make local
```

windows 下安装 `make` 可能较为麻烦, 你可以直接使用命令进行编译:

```
gcc fft.c -O3 -fopenmp -lm -Wall -o fft
```

运行:

```
./fft [N] # N表示本次运行规模为2^N
```

在本地运行代码会输出程序的正确性以及运行时间, 如图所示:

```
PS D:\Fdu\并行\lab1> make local
gcc fft.c -O3 -fopenmp -lm -Wall -o fft
PS D:\Fdu\并行\lab1> ./fft 20
size: 1048576
minimal time spent: 1197.5843 ms
result: correct (err = 2.225074e-308)
```

PS: 如果你是 Linux 或 Mac 用户, 我相信你可以自行安装以上软件并完成编译.

3.上传至服务器

确保在本地能编译通过并且程序正确之后, 你可以将代码上传至服务器运行.

可以使用 `scp` 命令将整个 `lab1` 文件夹上传至服务器, 使用教程可以参考[Linux scp命令](https://www.runoob.com/linux/linux-command-scp.html) | [菜鸟教程](https://www.runoob.com/) ([runoob.com](https://www.runoob.com/)).

登陆服务器后, 你可以先使用 `top` 命令查看是否有其他同学正在运行程序, 如果有, 建议你等待他的程序运行完成后再进行编译并运行. 由于我们的输入规模并不大, 一般最多也就等个十来分钟. 如果你发现某个同学的程序运行太长时间了, 请联系他或者助教将它停下

在服务器上编译代码使用以下命令:

```
make server
```

运行方式同上, 在服务器上运行代码会同时输出正确性, 运行时间. 我们调用了 `fftw` (这应该是目前最快的 `fft` 程序)来验证你的程序正确性. 然后会同时输出 `fftw` 的运行时间, 你可以以这个时间为目标不断优化你的代码, 结果如图所示:

```
ubuntu01@ubuntu01:~/lab1$ make server
gcc fft.c -O3 -fopenmp -lfftw3_omp -lfftw3 -lm -DSERVER -Wall -o fft
ubuntu01@ubuntu01:~/lab1$ ./fft 20
size: 1048576
minimal time spent: 322.2600 ms
threads: 24, baseline time spent: 3.3426 ms
result: correct (err = 3.666290e-12)
```

PS: 你可以不使用以上方式进行编译, 如果你发现任何编译器或编译选项能提高程序性能, 欢迎使用, 但请在实验报告中说明, 让助教知道怎么运行你的代码

评分标准

本次实验成绩组成: FFT性能 (80%) + 实验报告 (20%)

首先, 你必须答案正确, 才能得分。

性能部分评分方式: 性能第一名80分, 第二名79分...以此类推

实验文档中需要包含:

- 1.在服务器上的运行结果截图
- 2.算法主要思路
- 3.列举你使用的所有优化, 并简要说明它起作用的原因
- 4.(可选)你对于本次实验的想法, 任何方面都可以

实验报告不宜过长, 请勿在其中粘贴大量代码和注释.

另外, 请勿修改框架代码, 同学们唯一的任务是完成 `student_fft` 函数. 如果为了提升性能必须修改, 请在实验报告中说明. 如果框架代码存在问题, 请及时反馈, 助教将会尽快修正。

!注意: 严禁抄袭, 如果发现抄袭现象, 抄袭与被抄袭者本次实验均作0分处理

作业提交

作业截止时间: 2023年10月22日23点59分

请在 `elearning` 上对应入口提交实验报告, 实验代码上传至服务器中你的用户目录下即可

请及时提交作业, 如有迟交本次实验最高只有60分。