快速傅里叶变换并行算法的研究及实现

软件1804：肖劲涛 指导教师：苏波

[摘要] 快速傅里叶变换在数字信号处理、微分方程求解、图像处理等诸多方面有着广泛的应用。随着计算机技术的不断发展、微电子技术的日益革新，FFT技术被越来越多地应用在实际的生产生活中。我们主要使用三种技术来实现FFT的并行化，即，多线程技术、MPI技术和GPGPU技术。其中多线程技术使用C++的<thread>库和Windows SDK的HANDLE实现；MPI技术在Linux下使用OpenMPI完成；GPGPU技术则是借助CUDA实现。针对多线程技术，我们使用主-从模型，并且使用全局共享内存来优化；对MPI技术的优化同样是从“共享内存窗口”出发，对单个计算机上的多个进程之间的消息传递进行优化；CUDA则是通过存储空间的改变来提高GPU与CPU之间的通信效率。

[关键词] 快速傅里叶变换；并行计算；串行计算；高性能计算

**Research and Implementation of**

**Fast Fourier Transform Parallel Algorithm**

**Abstract：**Fast Fourier transform has a wide range of applications in digital signal processing, differential equation solving, image processing and many other aspects. With the continuous development of computer technology and the increasing innovation of microelectronics technology, FFT technology is more and more applied in actual production and life. We mainly use three techniques to realize the parallelization of FFT, namely, multi-threading technique, MPI technique and GPGPU technique. Among them, the multi-threading technology is implemented using the <thread> library of C++ and the HANDLE of the Windows SDK; the MPI technology is implemented using OpenMPI under Linux; the GPGPU technology is implemented with the help of CUDA. For multi-threading technology, we use the master-slave model and use global shared memory to optimize; the optimization of MPI technology also starts from the "shared memory window" to optimize the message passing between multiple processes on a single computer. CUDA improves the communication efficiency between GPU and CPU by changing the storage space.

**Key Words：**FFT, parallel computing, serial computing, high performance computing

# 0.引言

傅里叶变换在信号处理、力学、数学、金融等领域都有着广泛的应用[1-3]。傅里叶变换的一个典型用途是，将时域信号分解为为频域内振幅分量和频率分量。直接计算长度为N的序列卷积的离散傅里叶变换（DFT）的复杂度为O(N2)。Cooley-Tukey快速傅里叶变换(FFT)将计算复杂度降低到了O(NlogN) [4]。

尽管FFT表现不凡，但还是有很多问题需要解决。这些方面包括：计算精度的提高、高效低耗的FFT处理器设计、计算复杂度在理论与实际的最小边界、FFT数据迁移量的减少、系数访问效率的改善等。本项目将对比不同的傅里叶变换算法在不同的程序设计方案（串行、并行等）下的性能，并给出指导性报告。

# 1.FFT并行化的分析

我们通过并行技术来试图对FFT算法进行优化，但不是所有的算法都可以并行化实现，也不是所有的可算法都适合GPGPU技术。下面，我们将从两个方面来分析FFT算法并行化的可行性与优化设计。

FFT是否能并行化：

通过在第2章中对FFT算法的介绍，我们可以确定出FFT算法主要分为两个部分，一是倒位序，而是蝶形运算。对于基-2FFT来说，其蝶形单元的输入和输出均有两个，对于基-3FFT来说，则是三个。这里我们以基-2FFT为例，我们可以确定，在同一个蝶形单元中，其输入输出与同级的其他蝶形单元无关，只与该单元的两个输入与输出有关。也就是说，对于同级的蝶形单元的计算是可以并行的部分。而设计数据交换的不同级之间的数据则不适合并行处理，因为其计算量少，主要是逻辑控制的特点，使用并行技术所带来的通信开销可能远大于收益。因此这部分数据交换的内容我们选择使用串行技术处理。

FFT是否适合并行编程：

通过上面的介绍，我们知道，FFT算法主要分为两个部分。我们假设串行部分的执行时间为，可并行部分的执行时间为，整个程序的总执行时间为。通过使用并行技术，我们将可并行部分的速度提高了n倍，即执行时间减少到原来的，那么，现在的总执行时间为。不难看出，一个算法中并行部分所占比重越高，并行化后的效率提升越大。对于FFT来说，其蝶形运算是可并行的，并且是费时的运算，因此FFT的并行化对其效率提高很有帮助。

# 2.算法实现的关键步骤处理思路

## 2.1 旋转因子的处理

通过第2章的介绍，我们可以发现在快速傅里叶变换中，没进行一次蝶形变换，旋转因子就会被调用一次，其属于频繁被调用的变量。但同时我们发现，如果我们知道了问题的规模，那么我们就可以确定旋转因子的值与个数。在一次FFT变换中，问题规模不发生变化，因此这些旋转因子也是不发生变化的。通过前一小节对FFT可并行化的分析，我们可以知道，在不同级之间的蝶形变换时串行的，而同一级中的蝶形变换是可以并行的。从图2-4中可以看出，同一级的同一线程的蝶形变换所使用的旋转因子是一样的，表2-2中列出了不同级所调用的旋转因子。

根据旋转因子的这些特点，我们可以预处理一部分旋转因子，在多次进行FFT变换的时候，直接调用旋转因子，而不必在每次调用时进行计算。

## 2.2 任务划分

通过前面的分析我们可以知道，在并行FFT的设计中主要由两个部分，一个是串行的、不同级之间的数据交换；一个是可并行的、同级的蝶形变换。因此，这里我们可以考虑之前提到过的主-从模型，即，在串行时由主线程来进行简单的、不同级之间的数据交换，而对于复杂的蝶形变换则交给多个子线程来处理。使用主-从模型的关键在于任务的划分与分配。首先，需要比较在FFT算法的每个步骤的效率和复杂度来确定是否需要对其进行并行处理，因为对于细粒度的任务来说，并行的通信开销往往大于串行的时间花费。在问题规模较小，即细粒度的任务，我们考虑使用串行处理；而对于问题规模较大，即粗粒度的任务，我们考虑并行化。FFT作为一个大规模的、具有分治特点的算法，我们在3.2.1节中对其分析了哪部分适合串行处理，哪一部分适合并行处理。

需要额外注意的是，因为存在多级的蝶形变换，所以我们的程序需要不断的从主线程分发任务到子线程，同时在完成一级的变换后回收数据到主线程。因此，在我们的程序中，会不断的使用开启线程、等待线程两个步骤，即不断的分发与同步。因此这里需要额外注意同步所带来的时间开销。这里我们简单的将上面的步骤理解为“并行-串行-并行”这一过程，如图2-1所示，如果两次并行计算之间的串行所需要的同步开销大于使用多个线程来完成的同步开销，那么我们应该让子线程们依次完成数据的传递，而避免使用主线程完成所有的数据交互工作。此外，我们还可以考虑，在子线程计算蝶形变换的时候，主线程进行某些计算来提高下次数据交互的速度。

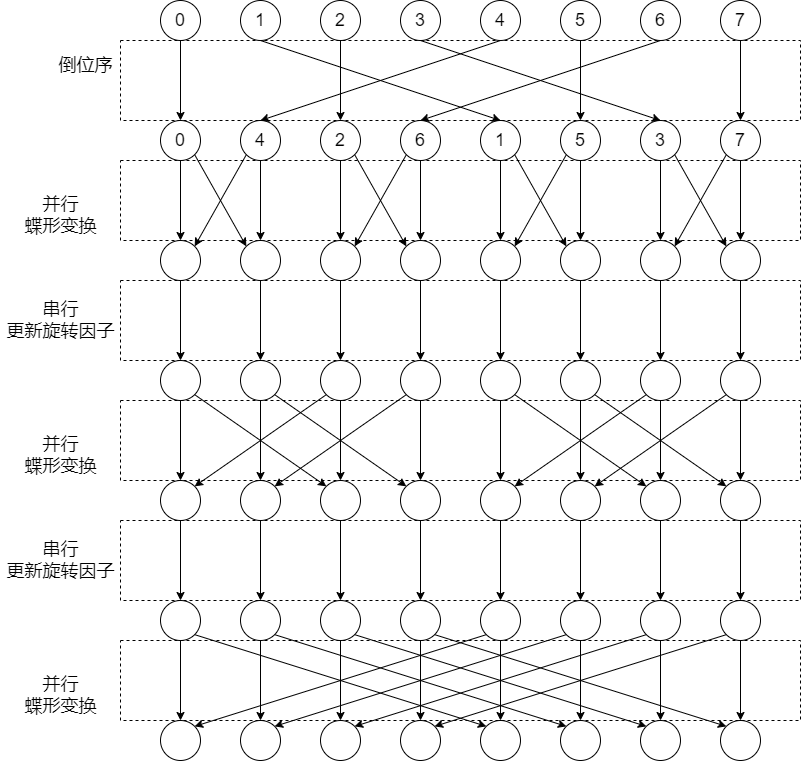


图2-1 并行FFT任务划分

下面我们对图3-1做一些补充说明。首先，我们这里所讨论的并行算法均要求序列长度为，其中。对于Cooley-Tukey算法来说，虽然其适用于任意长度的FFT计算，但是为了并行化，我们任然要求它的输入长度为。同时我们规定，任务规模远大于并行时使用的处理器数量。

对于长度为的序列，假设我们有*p+*1个处理器。对于第一阶段的倒位序来说，我们采用串行的方式来完成。对于该阶段来说，时间复杂度时线性的，一共需要进行次交换。在完成倒位序之后，我们开始第一次蝶形变换，这里使用*p*个处理器来并行处理，每个处理器负责组蝶形变换。完成一次蝶形变换后，主线程同步（等待）子线程，并计算在下一层所需要的旋转因子。在该阶段，我们忽略主线程计算旋转因子的时间，因为与计算蝶形变换相比，主线程的工作量并不高。考虑子线程的蝶形变换，我们知道，串行的FFT算法复杂度为，这里假设我们使用了*p*个子线程来做蝶形变换，那么此时的复杂度则为。以Cooley-Tukey算法为例的并行FFT算法如（算法2-1）所示：

|  |
| --- |
| **算法3-2：并行FFT算法** |
| **Input:**  需要计算的数据，*x(n);*  数据的规模*N*;  **Output:**  计算完成后的数据，*x(n)*;  1: 倒位序;  2: 当前线程（主线程）创建*p*个线程;  3: 计算*N*所对应的蝶形变换层级规模*M*;  4: 创建旋转因子存储数组*W[N]*;  5: 计算第1层所用的旋转因子; W[0] = ;  6: **for** *m* = 1; *m* <= *M*; *m*++ **do**  7: 主线程为子线程分配蝶形单元与旋转因子;  8: 主线程计算下一层的旋转因子，并按照表2-4的顺序放入*W*中;  9: 每个子线程计算当前的蝶形变换;  10: 主线程等待所有子线程完成;  11: **end for** |

# 3.FFT并行算法的实现

图3-1至图3-3展示了使用MPI技术实现的FFT及与串行FFT的对比。

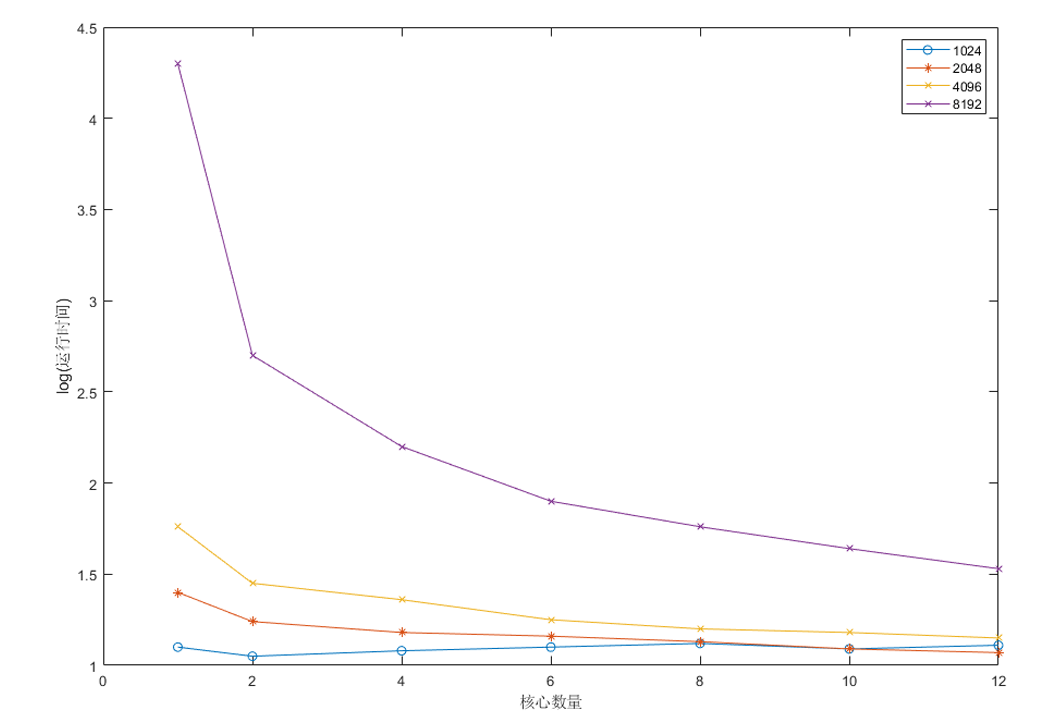


图3-1 使用MPI实现的FFT运行时间

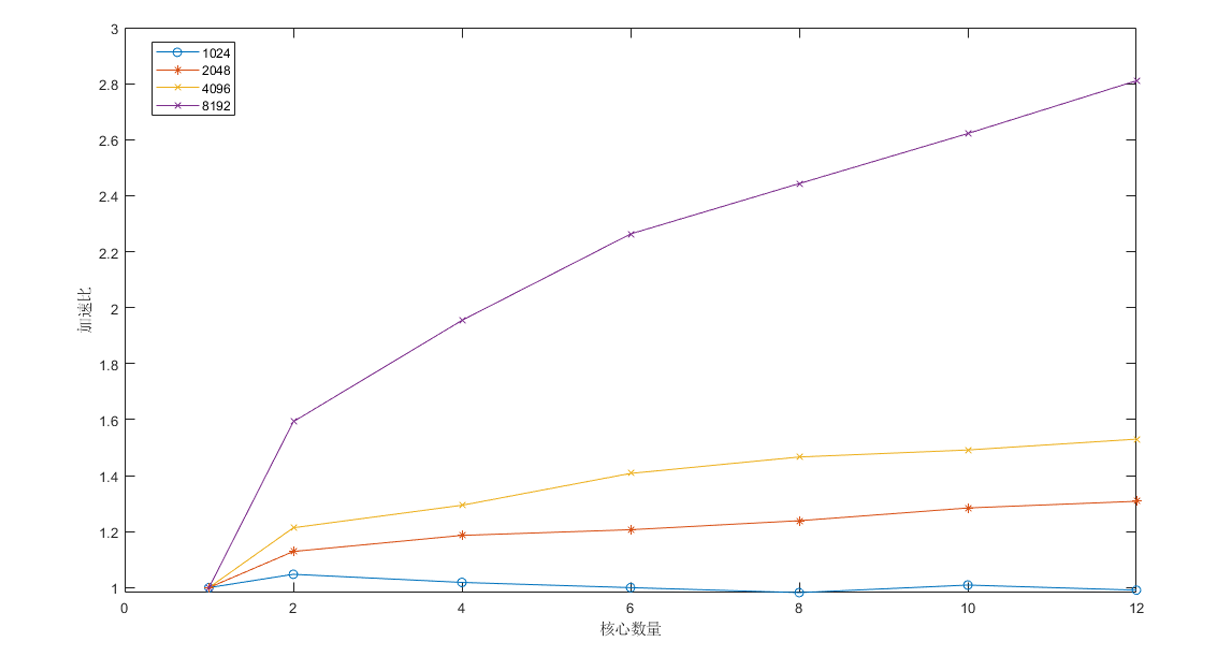


图3-2 MPI-FFT加速度

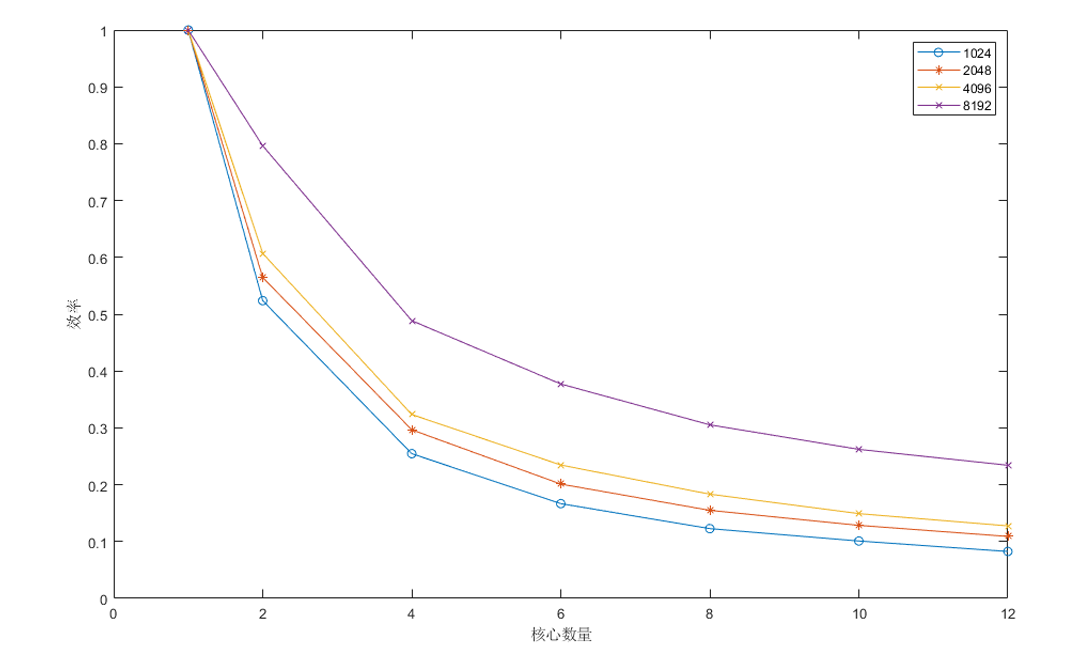


图3-3 MPI-FFT效率

# 4.结论

随着研究的深入，我们发现，并不是所有的并行设计都能够满足提高原本串行算法的效率。对于MPI技术来说，通信与同步是其运行时间的主要额外开销；而对于CUDA技术来说，主要的额外开销则来自消息在CPU与GPU之间的通信。于是我们提出了相应的优化方案。

1. 对于使用<thread>线程库的程序，之间开辟一个全局共享的内存空间即可，多线程技术使用主-从模型，主线程为子线程安排任务，同时在子线程被分配任务时进行下一轮的旋转因子计算。
2. 对于使用MPI技术的程序来说，我们采用预先计算旋转因子，并让每一个进程重复不同数据的相同计算。因为是在一台电脑上完成的计算，因此可以通过共享内存窗口来实现本地所有进程之间的数据共享，而不需要再依赖进程之间的点对点传递。
3. 对于CUDA技术来说，我们探讨了不同的显存配置与内主机之间交互的可能性，提出了相应的优化方案。

# 参考文献

[1] FRIGO M, JOHNSON S G J P O T I. The design and implementation of FFTW3 [J]. 2005, 93(2): 216-231.

[2] HSIAO C-F, CHEN Y, LEE C-Y J I T O C, et al. A generalized mixed-radix algorithm for memory-based FFT processors [J]. 2010, 57(1): 26-30.

[3] KATOH K, KUMA K-I, TOH H, et al. MAFFT version 5: improvement in accuracy of multiple sequence alignment [J]. 2005, 33(2): 511-518.

[4] COOLEY J W, TUKEY J W J M O C. An algorithm for the machine calculation of complex Fourier series [J]. 1965, 19(90): 297-301.