

**课 程 项 目 报 告**

**项目名称： 斐波那契数列计算**

**专业班级： CS1605**

**学 号： U201614541**

**姓 名： 洪锦涛**

**指导教师： 陆枫**

**报告日期： 2019年7月29日**

**计算机科学与技术学院**

目录

[1 串行算法复杂度分析 1](#_Toc15327214)

[1.1 时间复杂度分析 1](#_Toc15327215)

[1.2 空间复杂度分析 1](#_Toc15327216)

[1.3 主要算法 1](#_Toc15327217)

[2 分析并行实现的正确性 2](#_Toc15327218)

[2.1 可行算法的描述 2](#_Toc15327219)

[2.2 工作分解——依赖性和同步与通信开销分析 2](#_Toc15327220)

[2.3 编程模型 2](#_Toc15327221)

[2.4 性能检查及优化 2](#_Toc15327222)

[3 大数场景分析 4](#_Toc15327223)

[3.1 设计大数计算场景下斐波那契数列的测试用例 4](#_Toc15327224)

[3.2 分析并行实现的加速比。 4](#_Toc15327225)

[3.3 分析并行实现加速比的正确性。 4](#_Toc15327226)

[4 并行优化方案设计 5](#_Toc15327227)

[4.1 设计大数场景并行实现的优化方案 5](#_Toc15327228)

[4.2 测试并分析优化方案的加速比与正确性 5](#_Toc15327229)

# 1 串行算法复杂度分析

时间复杂度分析

串行解决斐波那契数列计算根据数学公式采用一重循环的操作，故时间复杂度为O(n)。

空间复杂度分析

算法未使用二维数组等，故空间复杂度为O(n)

主要算法

if(a==1)

{

printf("1");

}

else

{

printf("1");

for(int i=1;i<a;i++)

{

printf(" %d",c);

sum=b+c;

b=c;

c=sum;

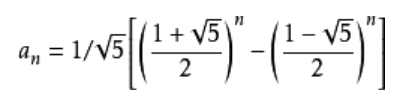
}

}

# 分析并行实现的正确性

可行算法的描述

斐波拉契数列的通项公式如下：



此时数列中每一项的计算只和项数有关，每一项的计算可以分割开来，于是可以将每一项的计算放在一个单独的线程里进行运算，运算结果放在数组中对应的位置，最后将数组按顺序输出即可。

工作分解——依赖性和同步与通信开销分析

利用通项公式来求解斐波那契数列，可以非常简单的并行对通项的求解，只需在for循环中将通项计算的结果赋值给数组即可并行的用数组保存计算的结果。

编程模型

使用OpenMP计算斐波那契数列；OpenMP是一种用于共享内存并行系统的多线程程序设计方案，OpenMP提供了对并行算法的高层抽象描述，编译器根据程序中添加的pragma指令，自动将程序并行处理。

OpenMP采用fork-join的执行模式。开始的时候只存在一个主线程，当需要进行并行计算的时候，派生出若干个分支线程来执行并行任务。当并行代码执行完成之后，分支线程会合，并把控制流程交给单独的主线程。

性能检查及优化



图2.1 串行运行时间



图2.2 OpenMP并行运行时间

由于数据集数量过小，加之使用了sqrt和pow函数等，导致并行的开销大于串行的开销，为负优化。

# 大数场景分析

设计大数计算场景下斐波那契数列的测试用例

大数场景基于OpenMP环境分析，串行和并行均使用通项公式来进行计算，两程序只要把#pragma omp parallel for一行去掉便相同，测试数据数量为1W

分析并行实现的加速比。



图3.1 串行下运行时间



图3.2 并行下运行时间

加速比约为5/4，数组数量为1W的情况下，并行稍微比串行快一点

分析并行实现加速比的正确性。

由于不知道OpenMP调用了多少个线程进行并行操作，故无法分析加速比的正确性。

# 并行优化方案设计

设计大数场景并行实现的优化方案

这次选择MPI来测试大数场景，数据数量仍为1W

测试并分析优化方案的加速比与正确性

在Linux环境下运行串行和MPI并行的结果：



图4.1 串行运行结果



图4.2 并行运行结果

加速比约为9/7，略微比OpenMP加速比高一点，可能是运行环境不同的影响

# 

# 