Journal of Wuhan Polytechnic

基于 akka 分治竞争算法的参数优化

李　坤

（武汉职业技术学院 计算机技术与软件工程学院，湖北 武汉 430074）

摘　要：Akka 是一种被广泛使用的，具有高性能的并行分布式计算工具包，Akka 用 Scala 写成，运行于 Java 虚拟机 JVM 平台。尽管 Akka 优雅简洁，但在构建复杂并行项目的过程中，许多来自于用户的决策，都会影响系统性能并带来延迟。采用深度优化的方法，在源代码中优化参数。使用 CMA-LES 进化算法来优化，实现并行分治算法，使用在 Akka 工具包后，显著提升了它的性能。

关键词：Scala；JVM；AKKA；分治算法；参数优化

中图分类号：TP301.6　　**文献标识码：**A　　**文章编号：**1671-931X（2017）03-0084-03

一、引言

在现代多核系统软件的开发过程中，支持并发性是一个重要的要求，与此同时开发时间和并发调试的消耗也逐渐提高[1]。为应对这种难题、使用 AKKA 框架来提供强大的并发支持已经在业界内被广泛使用。在 Java 和 Scala 开发领域，Akka 的并发模型已经是事实上的标准。它具有很多优势，如：不变性，避免由于数据可变，造成竞争算法中竞争条件的缺失。轻量性，大量的进程可以共享一个线程。容错性，具有同 Erlang 一样“崩溃自愈”的特性[2]。

在本文中，我们定义一个基于 AKKA 分治算法的通用模型，并使用它来演示一些典型的并发算法，包括：快速傅里叶变换（FFT）[3]，快速排序[4]，矩阵乘法[5]。然后应用深度参数优化的方法（DPT）[6]，来优化这些算法执行效率。

二、算法简介

FFT（快速傅立叶变换）是一个用来计算离散傅里叶变换（DFT）或它的逆矩阵序列的复数的算法。FFT 无处不在，在信号和图像处理和分析中，用于重新调整图像，降噪，恢复模糊图像。本方法使用一系列的乘法和加法的正弦波，得到 $O(n^2)$ 复杂度的结果。相比之下，FFT 实现 $O(n \log n)$ 复杂性的 DFT，分解出奇偶组件，计算它们的变换，然后融合在一起。这种渐近的复杂性也有利于如 Karatsuba[7]的多项式算法并分别得到 $O(n^{\log 3})$ 和 $O(n^2)$ 复杂度的结果。FFT 有不同的变式，但都拥有相同属性的 DFT 周期性和复杂度的对称性。这里应用的是蝶形算法（Cooley-Tukey）。

Quicksort（快速排序）是一个分治排序算法，它的广泛使用得益于一般情况下，其时间复杂度平均为 $O(n \log(n))$。启发选择元素会被选中作为一个支点，序列被分割，这样在一个子序列中，所有的元素都比中心元素的值要小。然后排序过程中，递归地应用到子序列。顺序算法的性能取决于枢轴中点的选择，由于它的递归性质，使得其适合并行计算。我们在这里利用优化的快速排序使得遗传算法规划，可以获得一种更好的主函数。

Strassen 矩阵乘法是一个分治的方法，它降低了矩阵乘法的复杂性，从 $O(n^3)$ 降至 $O(n^{2807 4})$。渐近快速算法虽然存在，但很少使用，因为高常数因子的存在，使得它们难于实践。Strassen 算法通过减少递归调用的数

收稿日期：2017-03-09

作者简介：李坤（1984-），男，湖北武汉人，武汉职业技术学院计算机技术与软件工程学院助教，研究方向：大数据、并行分布式计算。

量,虽然这会导致存储花费提高,但也提高了性能。

三、算法实现

现在描述我们的算法实现关键细节，包括使用 Akka 工具包,Akka 派发器 Dispatcher 的背景介绍,优化框架的描述,CMA-LESDPT 的执行。

我们算法的实现依靠于 Akka 的并发支持。Akka 并发的基本构建模块之一是 Future,它是一个在函数式编程中广泛使用的概念,作为一种“容器”支持一些并发操作产生最终结果。例如，一个类型的对象 Future[Double]最终会产生双重结果。我们的实现使用 Future,来队列工作单元,把目标进行递归细分。

Akka 并发执行是可以延时的，并发策略是选择使用 Akka ExecutionContext 封装。ExecutionContext 派发器管理调度线程,用于执行 Future。我们使用默认的调度参数,称为“fork-join-executor”,在大多数情况下它性能非常优秀。fork-join 执行器有两个整数参数,我们可以进行深度参数优化。这样,我们既保证了并行性,使得线程数与机器物理核心数相匹配,同时协调了线程间的资源竞争,保障了数据的吞吐量。

分治算法模型在下面的图 1 中显示，我们定义了分治算法模型的调用抽象方法,包括连续,分裂和合并方法。这些方法在子类中被定义,以对应多个算法:FFT,快速排序和 Strassen 矩阵乘法。我们并发使用 Akka 的工具箱来实现,包括“控制反转”递归模式,同时通过参数评估来划分 Future。

```
trait DivideAndConquer[Args,Result] {
    // Implemented by subclasses:
    def shouldDivide(args:Args):Boolean
    def sequential(args:Args):Result
    def divide(args:Args):Sew[Args]
    def merge(results:Seq[Future[Result]])
    (implicit ec:ExecutionContext):Future[Result]
//////////////////////
final def concurrent(args:Args):Future[Result] = {
    if( ! shouldDivide(args))
        Future.successful(sequential(args))
    else {
        val futures = divide(args).map { Future( concurrent(_)) }
        Future.sequence(futures).flatMap {   merge(_)) }
        }
    }
}
```

图 1 AKKA 的并行分治

获得的结果通过 Future 合并，根据子类方法实现。图 2 给出了相应子类快速排序的实现,硬件编码阈值参数紧接其后,同时定义了使用 Strassen 和 FFT 的顺序算法,这也有利于调整阈值参数。分治的实现是直接对应于实现的顺序。图 3 为快速排序的实现,并给出了单元测试部分，包括正确的排序顺序和并行实现随机生成了测试数据。

Strassen 矩阵乘法算法的实现利用了额外的可调参数,参数决定了是否可以递归分割矩阵至(1×1)大小,或成倍增加。与阈值参数一样,这个是独立于 Akka 的。CMA-LES 使用 ApacheCommonsMath 实现(一个轻量自包含的数学和统计组件,解决了许多通用但没有及时出现在 Java 标准语言中的实践问题)。使用其默认参数 settings5,初始数量级大于默认参数。我们执行 100 步的搜索,每 10 个执行步骤都使用的参数设置分别为候选解。表 1 显示我们找到最终输出搜索结果,选出了最好的参数基准。

四、相关工作和结论

我们应用深度参数优化的方法提取参数并优化了三个最常用的算法:FFT，快速排序和 Strassen 矩阵乘法，利用了 Akka 并发工具包。我们发现基于 CMA-LESDPT 进化算法可以实现 FFT 的执行时间减

```
class Quicksort
extends DivideAndConquer[List[Int],List[Int]] {
    Val Threshold = 100
    Val Throughput = 10
    Val ParallelismFactor = 3
    Val threadDispatcher =
        configureAkka(Throughput,ParallelismFactor)
//////////////////////
override def shouldDivide(data:List[Int]):Boolean =
    data.length > Threshold
// well-known recursive implementation:
override def sequential(data:List[Int]):List[Int] = {
    if( data.isEmpty) {
        data
    } else {
        val pivot = data.head
        val (left,right) = data.tail partition (_ < pivot)
        sequential(left) ++ (pivot :: sequential(right))
    }
}
override def divide(data: List[Int]): Seq[List[Int]] = {
        val pivot = data.head
        val (left,right) = data.tail partition(_ < pivot)
        Seq( left,List(pivot),right)
        }
override def merge(data:Seq[Future[List[Int]]]):
        Future[List[Int]] = {
    Future.sequence( data ).map { l =>
        l. head ++ l.tail.head ++ l.tail.tail.head
    }
}
```

图 2 通过分治和并行框架快速排序

```
class TestQuicksort {
    @Tes
    def test: Unit = {
    implicit val executionContext: ExecutionContext =
        ActorSystem().dispatcher
    val ArraySize = 1600000
    val testData = List.fill(ArraySize)(randomInt)
    val result1 = Quicksort.sequential(testData)
    val result2 = Quicksort.concurrent(testData)
assertTrue(isSorted(result1))
assertTrue(isPermutation(testData,result1))
assertEquals(result1,result2)
}
}
```

图 3 快速排序单元测试

表 1 CMA-LES 和随机搜索各自优化后执行时间的对比

	Algorithm	Min(s)	Max(s)	Median(s)
CMA-LES	FFT	3.54	3.97	3.84
	Quicksort	1.79	2.19	1.91
	Strassen	0.44	0.53	0.48
Random Search	FFT	8.85	8.96	8.92
	Quicksort	3.08	3.57	3.29
	Strassen	3.70	3.75	3.74

半且能增加产生更高数量级的 Strassen 算法速度。

参考文献：

[1] J.M.Calder′ onTrilla,S.Poulding,C.Runciman.Weaving Parallel Threads [A].M á rcio Barros,Yvan Labiche.Search-Based Software Engineering [C].Bergamo,Italy:International Symposium on Search Based Software Engineering,2015.

[2] J.Armstrong.A History of Erlang [A].HOPL III:History of Programming Languages Conference [C].CA:the Third ACM SIGPLAN Conference on History of Programming Languages,2007.

[3] F.Wu.W.eimer,M.Harman,Y.Jia,J.Krinke.Deep Parameter Optimisation [A].Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation [C].NewYork:In GECCO Proceedings,2015.

[4] A.A.Karatsuba.The complexity of computations [J]. Proceedings Of the Steklov Institute of Mathematics-Interperiodica Translation,1995,211:169-183.

[5] N.Hansen and A.Ostermeier.Completely Derandomized Self-Adaptation in Evolution Strategies [J].Comput., 2001,9(2):159-195.

[责任编辑：胡大威]

Parameter Optimization Based on akka Partitioning Competition Algorithm

LI Kun
(Wuhan Polytechnic, Hubei, Wuhan 430074, China)

Abstract: Akka is a widely used, high-performance parallel distributed computing toolkit. Akka is written in Scala and runs on the Java virtual machine JVM platform. Although Akka is elegant and concise, many decisions from the user can affect system performance and cause delays when building complex concurrent projects. Depth optimization is used to optimize parameters in the source code. Using the CMA-LES evolutionary algorithm to optimize and implement the parallel divide and conquer algorithm, it has significantly improved its performance after being used in the Akka toolkit.

Key words: Scala; JVM; AKKA; divide and conquer algorithm; parameter optimization

（上接第 83 页）

海:华东理工大学,2014.

[7] 刘闯.嵌入式 CAN 总线与以太网冗余网关的设计与实现 [D].大连:大连海事大学,2016.

[8] 曹卫平，梁兴东. 应用于匹配网络的负阻抗变换器研究 [J].微波学报,2012,28(4):85-87.

[9] Daniel Davidek,Jan Klecka,Karel Horak,Petr Novacek. Odometer Module for Mobile Robot with Position Error Estimation [J].IFAC PapersOnLine,2016,49(25): 346-351.

[10]李勇.汽车单片机与车载网络技术[M].北京:电子工业出版社,2015:189-218.

[11]李悦城,野火. uC/OS-III 源码分析笔记[M].北京:机械工业出版社,2016:58-219.

[责任编辑：刘　骋]

Multi-channel Signal Acquisition System Based on ARM Coordinated Control

MA Rui-jun[1],YUAN Fei[2],ZHAO Xian-mei[1]
(1.Industrial Training Center,Guangdong Polytechnic Normal University,Guangzhou510665,China; 2.School of Automation,Guangdong Polytechnic Normal University,Guangzhou510665,China)

Abstract: Data acquisition is an indispensable part of modern industrial control field. The application of multiple processors' master-slave control mode can greatly improve the performance of signal acquisition and processing. Using 32 bit ARM processor for high speed acquisition and processing of signals, and the CAN bus for establishing topology network to effectively dock data and commands in multiple inter processors, then take advantage of uC/OS-III real-time system to realize task scheduling in signal acquisition and processing, which can not only improve the coordination control between multiple processors, and realize the collection and remote transmission of multi-channel signals, enhance the real-time and stability of the system, and at the same time, the transmission interference can be well suppressed.

Key words: data acquisition; ARM; coordination control; uC/OS-III operating system