文章编号:2096-1472(2024)02-0040-04
DOI:10.19644/j.cnki.issn2096-1472.2024.002.008

任务并行编程模型下排列熵算法的并行实现

李维权

(青岛港湾职业技术学院信息与电气工程学院,山东 青岛 266404)
✉ 2760794118@qq.com

摘 要:排列熵算法随着嵌入维数的增大,运算规模将会呈平方级数增大,计算时效性问题突出,亟待解决。为此,提出一种基于任务并行编程模型的线程级并行方法,通过任务并行运行系统(StarPU)将密集型计算划分为多个独立的任务,再由调度器将任务调度到不同的 CPU 上执行,实现排列熵算法的并行化。基于 StarPU 的排列熵并行算法与串行程序相比较,加速比为 23.79 倍,相较于 OpenMP(一种用于共享内存并行系统的并行计算方案),在分配 28 个线程时,加速比为 1.17 倍,结果表明该方法能够有效实现排列熵算法的加速执行。

关键词:排列熵算法;任务并行编程模型;OpenMP;StarPU
中图分类号:TP311 **文献标志码**:A

Parallel Implementation of Permutation Entropy Algorithm in Task Parallel Programming Model

LI Weiquan

(*College of Information and Electrical Engineering, Qingdao Harbour Vocational & Technical College, Qingdao* 266404, *China*)
✉ 2760794118@qq.com

Abstract: With the increase of embedding dimension, the operation scale of permutation entropy algorithm will increase in a square series, and the problem of computational efficiency is prominent and needs to be solved. Therefore, a thread-level parallel method based on task parallel programming model is proposed. The task parallel running system (StarPU) divides intensive computing into multiple independent tasks, and then the scheduler schedules the tasks to be executed on different CPUs, achieving the parallelization of the permutation entropy algorithm. The parallel algorithm based on StarPU achieves 23.79 times speedup over serial programs, and 1.17 times speedup over OpenMP (a parallel computing scheme for shared memory parallel systems) when allocating 28 threads. The results show that this method can effectively accelerate the execution of the permutation entropy algorithm.

Key words: permutation entropy algorithm; task parallel programming model; OpenMP; StarPU

0 引言(Introduction)

排列熵(Permutation Entropy, PE)是一种新的动力学突变检测方法[1]。与傅里叶变换、短时傅里叶变换[2]、奇异值分解[3]、小波分析[4]、Hilbert 变换[5]等方法相比,具有输出结果简单直观、易于分析、对突变信息的识别较好等优势,已在电子、机械、医学和生物等众多领域得到广泛应用[6-13]。PE 算法一般研究的问题规模比较大,并且随着嵌入维数的增大,运算规模会呈平方级数增大,计算时效性问题突出,亟待解决[14]。为了加速执行排列熵算法,REN 等[15]提出了一种基于 FPGA (Field Programmable Gate Array)平台可以重构并行 PE 算法,在一定程度上缓解了其计算实效性问题,但受运行平台的限制,此算法无法在各个领域中获得广泛应用。曹建等[16]提出了一种基于云计算平台 MaxCompute 的并行排列熵算法,但该方法仅适用于云计算环境。随着多核计算机的普及,利用多种并行编程方法(如 OpenMP、MPI、OpenMP+MPI 混合编程模式等)实现排列熵串行程序的并行化,减少算法的运行时间。然而,目前使用基于任务并行编程模型解决 PE 算法的时效性问题的研究较少。为此,本文提出一种共享内存模式下基于任务并行编程模型的并行 PE 算法,它能通过多任务划分、运行时调度提高通用计算平台上 CPU 的利用效率,解决 PE 算法的时效性问题。

收稿日期:2023-05-10
基金项目:西兴技术智能语音系统(02030061003)

1 背景知识(Background knowledge)

1.1 PE 算法描述

假设需要分析的一维时间序列为 $X=\{x(i),i=1,2,\cdots,N\}$，在给定嵌入维数 m 时，可以构造出一组 m 维向量，此过程也叫相空间重构，重构后的相空间向量矩阵 $\boldsymbol{A}$ 如公式(1)所示：

$$\boldsymbol{A}=\begin{bmatrix} x(1) & x(1+\tau) & \cdots & x(1+(m-1)\tau) \\ \vdots & \vdots & & \vdots \\ x(j) & x(j+\tau) & \cdots & x(j+(m-1)\tau) \\ \vdots & \vdots & & \vdots \\ x(N) & x(N+\tau) & \cdots & x(N+(m-1)\tau) \end{bmatrix} \tag{1}$$

其中：m 为嵌入维数，τ 为时间延迟，取值为正整数。

在完成上述相空间重构后，紧接着就是进行相空间向量矩阵 $\boldsymbol{A}$ 中每一重构分量的排序。如果分量中出现两个值相等的情况，将按照分量值所在的序号进行排序。

对于空间向量矩阵 $\boldsymbol{A}$ 中的任一分量，总共有 $m!$ 种排列方式。假设每一种排列方式出现的概率为 $P_1,P_2,\cdots,P_o$，其中 $O\leqslant m!$，时间序列 X 的排列熵可以通过公式(2)计算得到：

$$H_p(m)=-\sum_{j=1}^{k}P_j\ln P_j \tag{2}$$

当 $P_j=1/m!$ 时，$H_p(m)$达到最大值 $\ln(m!)$。

1.2 基于任务的并行编程范式

遵循基于任务的编程范式的程序结构由任务集合(一段顺序代码区域)组成，这些任务在数据区域集合上具有特定的目的。为了保证计算的一致性，任务间的交互主要通过在不同任务中使用相同的内存区域实现，因此任务间产生了隐含的依赖关系。通常，基于任务的应用程序由 DAG(Directed Acyclic Graph)组成，其中节点是任务，边是依赖项。隐式数据重用检测或程序员显式插入是描述依赖关系的方式。采用基于任务的编程模型的好处之一是使高性能计算应用程序的编程更加抽象，让程序员远离异构执行平台的细节。PaRSEC 和 StarPU 是两个典型的基于任务的编程模型的运行时系统。虽然它们之间有相似之处，但是本文只详细介绍 StarPU 运行时系统。

StarPU 是基于任务的应用程序和具有不同加速器(如 CUDA/OpenCL GPGPUs)的多核系统之间的接口层，是一个运行时系统。要使用 StarPU，程序员需要在任何需要的资源(CPU、CUDA、OpenCL)上提供每个任务的实现过程，并将使用的内存区域声明为数据句柄。StarPU API 允许声明每个任务使用的数据，然后提交它们。为了决定任务应该在资源中的哪个位置执行，StarPU 采用了不同的启发式调度方法。

2 PE 算法的并行实现(Parallel implementation of PE algorithm)

2.1 基于 OpenMP 的 PE 算法并行

针对 PE 算法多重复性 for 循环计算的特点，可以通过共享内存编程模型 OpenMP 进行多线程并行设计。以下重点介绍排列熵算法中相空间重构和矩阵重构分量排序两个核心计算的 OpenMP 实现过程。

(1)相空间重构：在相空间重构过程中，矩阵中各行重构过程无依赖关系，所以可以直接使用 OpenMP 指导语句将重构过程进行并行。算法 1 描述如下：

算法 1：相空间重构 OpenMP 实现

输入：时间序列 X，嵌入维数 m 和时间延迟 τ

输出：重构矩阵 $\boldsymbol{A}$

```
1. Begin
2.   利用长度、嵌入维数 m 和时间延迟 τ 计算相空间行数 max
3.   #pragma omp parallel for
4.   for i=0 to max do
5.     for j=0 to m do
6.       A[i][j] = X[i+j*τ]
7. End
```

(2)重构矩阵分量排序：在重构分量排序过程中，各行重构分量排序间也没有依赖关系，因此可以采用与矩阵重构部分相同的方法进行并行化。与之不同的是，统计排序产生 Hash 值时，共享内存模式下的多线程并行会带来统计结果写冲突。针对该问题，可以采取两种方式解决，一是通过原子操作语句解决问题，但会影响程序的性能；二是通过中间数值临时保存，在并行区外进行 Hash 值统计。第二种方法避免了并行区的原子操作，可以提高程序的并行性，因此本研究采用了第二种方法。算法 2 描述如下：

算法 2：重构矩阵分量排序 OpenMP 实现

输入：重构矩阵 $\boldsymbol{A}$，矩阵行数 max，嵌入维数 m

输出：Hash 表 $dict$

```
1. Begin
2.   long long tmp[max+1]  //负责存储重构分量排序产生的 Hash 值
3.   #pragma omp parallel for
4.   for i=1 to max+1 do
5.     for j=0 to m do
6.       M[j]=j+1
7.   for j=0 to m do
8.       for k=m-1 to 0
9.         if A[i][k-1] > A[i][k]
10.          {
11.            Swap( A[i][k-1], A[i][k])
12.            Swap( M[k-1], M[k])
13.          }
14.  for j=0 to m do
15.      由数组 M 计算出 i 行重构分量产生的 Hash 值存到 tmp[i]中
16.  for i=1 to max+1 do
17.   dict[tmp[i]]←dict[tmp[i]] + 1  //在并行区域外统计 Hash 表，避免原子操作
18. End
```

2.2 基于 StarPU 的 PE 算法并行

StarPU 是一个支持任务调度的运行时系统，提供了任务调度管理的统一视图，同时隐蔽了很多底层细节。编程者需要将密集型计算划分为一组任务，给出任务在计算单元上的代码实现过程，以及任务之间的数据依赖关系，在任务提交的过程中，系统会根据任务之间的依赖关系绘制 DAG 图，并将任务放入工作队列中等待处理单元执行。

StarPU 会检测集群系统中可用的计算单元，包括不访问主存的协处理器(如 GPU)，根据系统配置分配独立的任务计算单元 Worker，一个 CPU 类型的 Worker 需要占用一个 CPU 资源，一个 GPU 类型的 Worker 需要占用一个 CPU 资源和一

个 GPU 资源，任务被放到工作队列中后，调度器会根据任务的实现形式将其调度到对应的计算单元上。StarPU 任务的调度过程如图 1 所示。

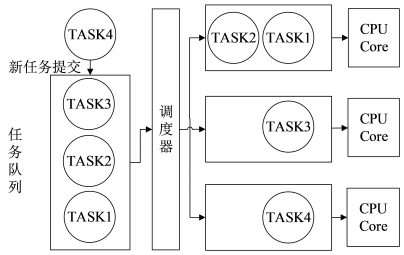

图 1 StarPU 任务的调度过程

Fig. 1 The scheduling process of StarPU tasks

在创建 StarPU 任务时，需要给出任务的执行环境，告诉调度器该任务可以在哪种计算单元上执行。被提交的任务按照时间先后顺序进入等待队列等待调度器调度。调度器负责将任务调度到适合的计算单元上执行。每个计算单元都有一个等待队列，如果采用 Eager 调度算法，计算单元会在自己的等待队列为空时，向调度器请求新的任务；如果采用 work_stealing 算法，为了使负载均衡，计算单元会优先考虑从其他负载较大的等待队列中取任务到自己的队列。

在 PE 算法生成重构矩阵后，需要将重构矩阵 $\boldsymbol{A}$ 每行的重构分量按照升序排序，它将成为该算法最耗时的部分。采用 StarPU 进行 PE 算法并行的核心是将重构矩阵 $\boldsymbol{A}$ 划分为多个小的任务，再将这些任务调度到多个处理器并行计算。通过研究排列熵算法发现，重构矩阵 $\boldsymbol{A}$ 各行的排序计算不存在依赖关系，可以直接按行划分。为每个任务分配其需要维护的矩阵分块，当 CPU 接收到任务时，按照任务给出的函数生成矩阵分块，并完成排序。

在使用基于 StarPU 进行排列熵算法并行过程中，在统计排序产生的 Hash 值时，同样会遇到多个 CPU 同时修改 Hash 表所带来的写冲突问题。为避免写冲突问题，把统计移到并行任务外，每个任务排序后的 Hash 值统计结果暂时存储到临时数组中，等所有任务完成后，再统一修改 Hash 表。基于 StarPU 的 PE 算法并行流程如图 2 所示。

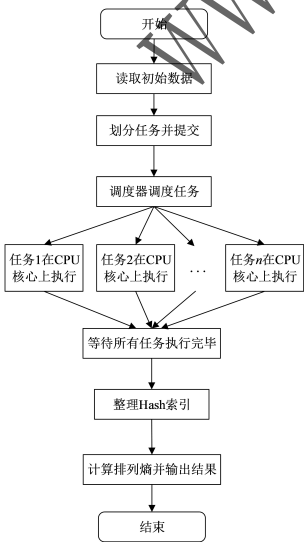

图 2 基于 StarPU 的 PE 算法并行流程

Fig. 2 The parallel process of PE algorithm based on StarPU

3 测试与测试结果(Testing and test result)

3.1 测试环境

Intel Xeon E5-2690 V4 处理器及软件配置参数如表 1 所示。

表 1 Intel Xeon E5-2690 V4 处理器及软件配置参数

Tab.1 Intel Xeon E5-2690 V4 processor and software configuration parameters

规格	数值
频率/GHz	2.6
CPU 数/个	2
核数/个	28
主存容量/GB	512
操作系统	Centos 7.4.1708
编译器	GCC 4.8
StarPU 运行时	StarPU-1.3.2

程序采用 C++语言编写，测试算例是在青岛某建筑震荡现场采集的数据，数据采用 xls 格式，数据量为 50 000 条，嵌入维数为 1 000，延迟时间为 10 s。

3.2 测试结果及分析

PE 算法计算过程使用集群中的两个节点，并且分别与串行 PE 算法和基于 OpenMP 的并行 PE 算法进行比较。因为服务器总共包含 28 个计算核心，StarPU 可以申请的最大 Worker 数为 28 个，所以 OpenMP 并行程序统计的最大线程数设置为 28 个线程。

StarPU 运行时需要先初始化环境，经测试得出初始化过程需要耗费 38 s，因为测试时使用的数据量小，无法忽视初始化带来的时间开销，所以在测试两个程序的运行时间时，都只测试计算过程耗费的时间。表 2 为基于 OpenMP 的 PE 算法的测试结果，表 3 为基于 StarPU 的 PE 算法的测试结果。

表 2 基于 OpenMP 的 PE 算法的测试结果

Tab.2 Test result of PE algorithm based on OpenMP

序号	线程数/个	运行时间/s	加速比
1	1	79.42	—
2	2	43.36	1.83
3	4	20.46	3.88
4	8	10.41	7.62
5	16	5.25	15.13
6	28	3.91	20.31

表 3 基于 StarPU 的 PE 算法的测试结果

Tab.3 Test result of PE algorithm based on StarPU

序号	线程数/个	运行时间/s	加速比
1	1	71.13	—
2	2	35.72	1.99
3	4	18.09	3.93
4	8	9.24	7.69
5	16	4.93	14.42
6	28	2.99	23.79

图 3 为基于 StarPU 与基于 OpenMP 两种算法的运行时间比较，图 4 为基于 StarPU 与基于 OpenMP 两种算法的加速比比较。

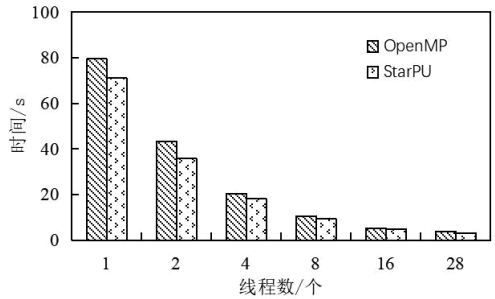

图 3　基于 StarPU 与基于 OpenMP 两种算法的运行时间比较

Fig. 3 Comparison of the running time between StarPU based and OpenMP based algorithms

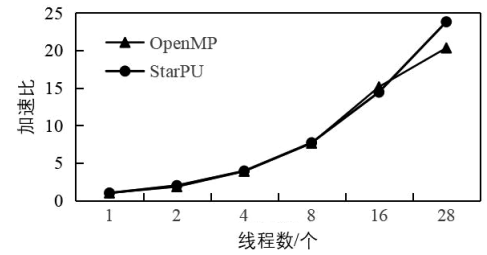

图 4　基于 StarPU 与基于 OpenMP 两种算法的加速比比较

Fig. 4 Comparison of the acceleration ratios between StarPU based and OpenMP based algorithms

从图 3 和图 4 可以看出，随着使用线程数的不断增加，程序的运行时间越来越短，加速比越来越大。在使用 28 个 CPU 核心时，OpenMP 获得了最大加速比，为 20.31 倍，StarPU 获得的加速比为 23.79 倍。

对比分析可以发现两种模型有以下几点不同。

(1)StarPU 开始运行时，需要检测集群配置信息并进行运行时系统的初始化，该过程有一定的时间开销，所以 StarPU 更适合优化计算规模比较大的程序。

(2)单独比较并行过程所用的时间，StarPU 更快，相比于 OpenMP，StarPU 的执行效率更高。

(3)随着使用的 CPU 核心个数的增加，两个模型取得的加速比都越来越大，但是 OpenMP 加速比的增长速度逐渐降低，而 StarPU 的并行加速比与 CPU 核心个数更接近线性关系。

4　结论(Conclusion)

为了解决排列熵算法的时效性问题，本文提出了基于任务并行编程模型的并行排列熵算法。该算法采用任务驱动，具有更细的并行粒度，负载均衡度更好。与 OpenMP 实现的并行排列熵算法相比，该算法的加速效果更好，加速比与线程个数的增加呈现线性关系。但是，目前本研究实现的只是共享内存模式下的多任务调度排列熵算法，如果想要将这种算法移植到异构平台上，就需要解决任务数据的传输问题。今后的工作中，将更深入地了解 StarPU 这类任务并行模型，解决异构平台的通信问题，进一步优化排列熵算法。

参考文献(References)

[1] 周倩，梁建国，傅游. 神威·太湖之光上排列熵算法异构并行加速[J]. 计算机工程与设计，2023，44(2)：400-406.

[2] AUBEL C，STOTZ D，BÖLCSKEI H. A theory of super-resolution from short-time Fourier transform measurements[J]. Journal of Fourier analysis and applications，2018，24(1)：45-107.

[3] BAI Y，LI Y，WANG X X，et al. Air pollutants concentrations forecasting using back propagation neural network based on wavelet decomposition with meteorological conditions[J]. Atmospheric pollution research，2016，7(3)：557-566.

[4] 刘宇，王迁，刘阔，等. 基于小波奇异性和支持向量机微铣刀破损检测[J]. 东北大学学报(自然科学版)，2017，38(10)：1426-1430.

[5] 张金豹. 基于全寿命数据的滚动轴承运行状态评估和剩余寿命预测[D]. 哈尔滨：哈尔滨工业大学，2020.

[6] 陈长征，李雪，孙自强. 基于多尺度排列熵的滚动轴承故障诊断[J]. 机械工程师，2019(9)：17-19，22.

[7] 伍济钢，文港. 基于优化多尺度排列熵和卷积神经网络的滚动轴承故障诊断方法[J]. 航天器环境工程，2023，40(1)：99-106.

[8] 张蒙，朱永利，张宁，等. 基于变分模态分解和多尺度排列熵的变压器局部放电信号特征提取[J]. 华北电力大学学报(自然科学版)，2016，43(6)：31-37.

[9] 石雨菲，王瑶，胡珊，等. 基于 MPE-ANN-SVM 的癫痫脑电检测分类研究[J]. 生物医学工程研究，2022，41(4)：353-358.

[10] 胡廷，孙婕，牛焱，等. 基因和排列熵在阿尔茨海默病早诊中的研究应用[J]. 计算机工程与应用，2020，56(15)：274-278.

[11] 郑有飞，尹继福，吴荣军. 我国大陆极端高温基于去趋势波动及排列熵法的时空分布特征研究[J]. 热带气象学报，2012，28(2)：251-257.

[12] 张林，余成波，谭拥，等. 基于 VMD-NLMS 的运动状态血氧监测算法[J]. 计算机仿真，2023，40(7)：342-347，500.

[13] 苗俊田，李卓军，刘冬冬，等. 基于 CEEMDAN 的压裂装备潜在性故障诊断模型[J]. 现代电子技术，2023，46(20)：17-20.

[14] 杨鹏，申洪涛，陶鹏，等. 云平台下时间序列数据并行化排列熵特征提取方法[J]. 电力自动化设备，2019，39(4)：217-223.

[15] REN X W，REN P J，CHEN B D，et al. A reconfigurable parallel acceleration platform for evaluation of permutation entropy[C]//IEEE. Proceedings of the IEEE：2014 36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society(EMBC). Piscataway：IEEE，2014：5735-5738.

[16] 曹建，李峥，杨璞，等. 云计算环境下基于 MapReduce 的并行化排列熵算法[J]. 电力信息与通信技术，2019，17(1)：1-6.

作者简介：

李维权(1965-)，男，本科，高级工程师。研究领域：并行计算，分布式计算。