

# 一种基于优化存储格式的 DLB\_GaBP 算法

陈振武<sup>1,2</sup>, 黄 婧<sup>1</sup>, 兰添才<sup>1,2</sup>, 郑汉垣<sup>3</sup>

(1. 龙岩学院 数学与信息工程学院, 福建 龙岩 364012;

2. 龙岩学院 大数据挖掘与应用福建省重点实验室, 福建 龙岩 364012;

3. 龙岩学院 传播与设计学院, 福建 龙岩 364012)

**摘 要:** 基于动态运行的多线程或多进程环境中的多核并行处理机, 常存在数据存储格式及数据读取方法不合理, 并行运行任务的不均衡性状态, 直接影响到系统工作的整体效率, 因此, 优化数据存储格式、均衡优化任务调度算法是保证整个系统运行效率的关键。利用高斯置信传播算法 (GaBP) 在求解对称对角占优线性方程组中具有高并行度、低复杂度的特性, 结合对数据存储格式进行优化的基础上, 设计实现一种具有动态负载均衡特性的多核并行 GaBP 算法 (DLB\_GaBP 算法)。利用该算法并通过对稀疏矩阵集 (UFget) 的求解实验, 在千万量级以上的大规模计算环境中, DLB\_GaBP 算法具有更好的计算效率和更高的加速比, 这为充分发挥多核并行处理机的运算能力及大规模计算问题的快速求解提供了一种新算法思路。

**关键词:** 大规模并行运算; 稀疏矩阵存储格式; 动态负载均衡; GaBP 算法; DLB\_GaBP 算法

**中图分类号:** TP301.6

**文献标识码:** A

**文章编号:** 1673-629X(2020)06-0071-06

**doi:** 10.3969/j.issn.1673-629X.2020.06.014

## A DLB\_GaBP Algorithm Based on Optimized Storage Format

CHEN Zhen-wu<sup>1,2</sup>, HUANG Jing<sup>1</sup>, LAN Tian-cai<sup>1,2</sup>, ZHENG Han-yuan<sup>3</sup>

(1. School of Mathematics and Information Engineering, Longyan University, Longyan 364012, China;

2. Key Laboratory of Big Data Mining and Application of Fujian Province, Longyan University, Longyan 364012, China;

3. School of Communication and Design, Longyan University, Longyan 364012, China)

**Abstract:** The multi-core parallel processor based on dynamic multi-thread or multi-process environment often has unreasonable data storage format and data reading method, and the unbalanced state of parallel running task directly affects the overall efficiency of the system. Optimizing data storage format and balancing and optimizing task scheduling algorithm are the key to ensure the efficiency of the whole system. Based on the high parallelism and low complexity of Gaussian confidence propagation algorithm (GaBP) in solving symmetric diagonal dominant linear equations, a multi-core parallel GaBP algorithm (DLB\_GaBP algorithm) with dynamic load balancing characteristics is designed and implemented on the basis of optimizing the data storage format. In a large-scale computing environment with more than 10 million levels, the algorithm is used and the experiment of solving sparse matrix set (UFget) is carried out. DLB\_GaBP algorithm has better computational efficiency and higher acceleration ratio, which provides a new algorithm idea for giving full play to the computing power of multi-core parallel processors and fast solving large-scale computing problems.

**Key words:** large-scale parallel operation; sparse matrix storage format; dynamic load balancing; GaBP algorithm; DLB\_GaBP algorithm

## 1 概 述

现代多核处理机或多处理机系统速度能力已实现了每秒千万亿次浮点运算的量级, 因此, 要提高大规模的数值计算的速度与效率, 最终均归结于操作系统的任务流调度算法、存储空间存储格式的合理定义与

分隔。

对于多核处理机环境中的任务流调度算法可分为全局队列调度算法和局部队列调度算法两种。局部队列调度算法是为每个处理机核心维护一个局部任务等待队列, 处理机的利用率较低, 在大规模并行计算中一

收稿日期: 2019-07-29

修回日期: 2019-11-29

**基金项目:** 福建省自然科学基金项目 (2015J01587); 国家自然科学基金重大研究计划重点项目 (91630206); 龙岩学院博士基金项目 (2015)

**作者简介:** 陈振武 (1963-), 男, 高级实验师, CCF 会员 (76942M), 研究方向为计算机结构与性能评测; 通信作者: 郑汉垣 (1965-), 男, 博士, 教授, CCF 高级会员 (76941M), 研究方向为高性能计算及应用。

般不予使用;全局队列调度算法从处理机的任务均衡性、存储格式的合理性方面综合考虑充分发挥系统中各处理机核的综合性能,当某一处理机空闲时,系统立即从全局任务等待队列中选取就绪任务分配给该处理机执行,从而提高处理机的利用率,达到提高系统的运算速度与效率的目的。

吉林大学的耿晓中提出了一种动态负载均衡模型,并将影响多核处理器负载均衡的因素分为 5 类:多核系统的负载均衡环境、用户提交的任务属性、系统的负载评价、系统所采用的调度策略以及系统的调度评价指标<sup>[1]</sup>,为多核动态调度因素提供依据。

文献[2]针对动态调度算法的基本原理和执行过程,结合负载均衡策略,提出了一种基于异构多核处理器的 STDS 动态任务调度算法,并通过选择调度时间、负载均衡和任务等待时间三个方面证明了 STDS 算法在保证速度性能的同时有较理想的内核负载均衡效果。

贾燕成等提出了负载动态均衡规则<sup>[3]</sup>是在任务的调度过程中根据处理器的状态,将较大的任务分解并动态映射到其他节点上,并与该节点的原始任务组进行组合形成后继任务循环执行,使各个节点的任务达到动态平衡,可以避免节点的空闲等待,也缩短了各个节点通信开销,最大化提高了并行效率。

近年来,随着互联网+、物联网、大数据云计算应用领域的不断开展,计算机系统的运算量也在不断增加。如电力系统运行控制的潮流计算问题<sup>[4]</sup>、环境监控、实时评价问题、地震实时监测等领域,均涉及到大规模数值计算问题。而对这些大规模数值问题的求解,大多都是通过相应的数学建模,然后将问题的求解转换成为相应建立起来的稀疏线性方程组  $Ax = b$  的求解<sup>[5-7]</sup>。

迭代法是在稀疏线性方程组求解中常用的方法,尤其是近年来,基于 Krylov 子空间方法<sup>[8-9]</sup>的迭代法得到了广泛应用,Ori-Shental 等人基于递归更新的概率推理算法提出了 GaBP 迭代算法<sup>[10]</sup>。

文献[11-13]针对 GaBP 迭代算法具有计算复杂度低、并行度高的特性,在 GaBP 的迭代加速优化方法的基础上给出了对应的多种 GaBP 迭代加速优化算法,并从动态松弛因子的 GaBP 算法和 MannGaBP 迭代加速优化算法的实验,构造多核处理机动态因子。

文中针对多核处理机环境中的大规模稀疏矩阵线性方程组的存储格式、处理机间的任务均衡性、并行算法的优化等进行研究,在优化稀疏矩阵存储格式的基础上,依据文献[3]提出的负载动态均衡规则,对多核处理机环境的并行 GaBP 算法进行优化,设计实现具有动态负载均衡特性的多核并行 GaBP 算法(DLB\_

GaBP 算法)。经过实验验证,该算法在对大规模稀疏矩阵线性方程的求解过程中,具有更高的加速效果、更快的计算速度、更好的环境适应能力。

## 2 高斯置信传播(GaBP)算法

### 2.1 算法思想

GaBP 算法主要包括:初始化和迭代(包括消息累加、消息传播与更新、求解向量三个过程<sup>[13]</sup>)两个步骤。具体可描述如下:

步骤一:初始化。

(1) 设:  $P$ 、 $\mu$  与  $A$  是同阶的矩阵( $A$  是系数矩阵),  $\tilde{p}$  和  $\tilde{\mu}$  是与线性方程组右端向量同阶的列向量;  $i$ 、 $j$  是矩阵行与列节点编号。

(2) 初始化。

$$p_{ij} = 0; \mu_{ij} = 0 (i \neq j)$$

$$p_{ii} = A_{ii}; \mu_{ii} = b_i$$

步骤二:迭代计算。

(1) 消息累加。

$$\tilde{p}_i = p_{ii} + \sum_{k \in N(i)} p_{ki}$$

$$\tilde{\mu}_i = \mu_{ii} + \sum_{k \in N(i)} \mu_{ki}$$

//  $N(i)$  为第  $i$  行节点编号的集合(不含  $i$ );

(2) 消息传播与更新。

$$p_{ij} = \tilde{p}_i - p_{ji}; \mu_{ij} = \tilde{\mu}_i - \mu_{ji}$$

$$P_{ij} = -A_{ij}^2 P_{ij}^{-1}; \mu_{ij} = -P_{ij}^{-1} A_{ij}^2 \mu_{ij}$$

(3) 求解向量。

$$x_i = \tilde{\mu}_i / \tilde{p}_i$$

### 2.2 GaBP 算法的改进

分析 GaBP 算法可以发现,迭代过程非常类似 Jacobi 迭代算法,采取的是同步更新,即每一次迭代未使当前迭代步中的消息得到及时更新,只是对上一步的消息进行更新,这明显增加了迭代频数,降低了收敛速度。同时,GaBP 算法设计的前提是将  $Ax = b$  的求解转化成为对称对角占优线性方程组求解的等价性( $\max_x (e^{-\frac{1}{2}x^T Ax + b^T x})$ )问题的求解,J. Pearl 等人对于这一求解转换进行了分析,并证明了分别利用 GaBP 算法与 Gauss 消元法(直接法)的求解存在等价性,同时还证明了在收敛性方面,GaBP 算法具有更好的收敛效果<sup>[14-15]</sup>。

## 3 存储格式定义

文献[16]在充分分析了 ELL、CSR、COO、ELL+COO 混合的 HYB(Hybrid)等多种存储格式的优缺点的基础上,提出了 ELL+CSR 混合的 HEC 存储格式,并进行了相应的实验对比验证,也发现了 HEC 存储格

式比 CSR 格式多存储了 2 组分别用以存储格式本身所需设定的值和列数据,对读取带来一定的开销。

为了充分利用 GaBP 算法的特殊性,文中定义动态负载均衡多核并行 GaBP 的数据结构,在数据存储格式及数据读取方法上采用列压缩存储格式,格式本身分为两种形式:改进的列压缩存储格式(modified compressed spare column, MCSC)或优化的列压缩存储格式(optimized compressed spare column, OCSC)。

### 3.1 矩阵数据参数设定

设:  $n$  阶稀疏矩阵  $A$ , 有非零元素  $n_z$  个;

另设: 存储单元数组 ADN、ALN、AUN、NL、NJL、NU、NJU。

其中, ADN 用于存放按顺序排列的  $n$  个对角矩阵元素值  $a_i$ ;

ALN 用于按列存储  $A$  左下三角( $n_z - n$ )/2 个非零元素值  $a_{ij}$  ( $i > j$ ), NL 用于存储行号, NJL 用于存储非零元素对应列起始行号(NJL = 1 ~  $n$ );

AUN 用于按列存储  $A$  右上三角( $nn_z - n$ )/2 个非零元素值  $a_{ij}$  ( $i < j$ ), NU 用于存储行号, NJU 用于存储非零元素对应列起始行号(NJU = 1 ~  $n$ )。

### 3.2 存储格式定义

由于文中利用 UFgett 中的稀疏矩阵进行实验验证<sup>[17]</sup>,  $A$  按行号、列号、元素值三个元素组进行存储,可建立如下变换存储格式的左端矩阵数据结构:

$$A = \begin{Bmatrix} 4 & 2 & 7 & 1 \\ 2 & 2 & 0 & 9 \\ 7 & 0 & 1 & 0 \\ 1 & 9 & 0 & 5 \end{Bmatrix} = AL + AD + AU =$$

$$\begin{Bmatrix} 2 \\ 7 & 0 \\ 1 & 9 & 0 \end{Bmatrix} + \begin{Bmatrix} 4 \\ 2 \\ 1 \\ 5 \end{Bmatrix} + \begin{Bmatrix} 2 & 7 & 1 \\ 0 & 9 & 0 \end{Bmatrix}$$

$$\begin{matrix} \text{AND} = [4, 2, 1, 5] \\ \text{ALN} = [2, 7, 1, 9, *] \\ \text{AUN} = [* , 2, 7, 1, 9] \\ \text{NL} = [2, 3, 4, 4, *] \\ \text{NU} = [* , 1, 1, 1, 2] \\ \text{NJL} = [1, 2, 3, 5, 0] \\ \text{NJU} = [0, 1, 2, 3, 4] \end{matrix}$$

条件一: 若  $A$  不对称

则: 按 ADN, ALN, NL, NJL, AUN, NU, NJU 的顺序的 MCSC 格式存储。

条件二: 若  $A$  对称

则: 按 AD, AL, NBL, NBJL 顺序的 OCSC 格式存储。

条件三: 若  $A$  是对角稠密矩阵, 则非零元素的存取与查找, 采用 OCSC 格式存储结合顺序查找算法可以非常直接地进行。

条件四: 若  $A$  是非对角稠密矩阵, 非零元素, 一般可采用二分查找方法进行。依据 GaBP 算法稀疏性的特性, 形状对称矩阵  $P\mu$ 、 $A$  (其中  $A$  是左端数值对称矩阵), 利用 OCSC 存储它们, 可以将 NL、NJL 与 ALN、ADN 的存储格式进行一样的定义。但由于矩阵  $\mu_{ij}$  是形状对称的, 这需将其上三角元素形成矩阵存储, 可以借助 MCSC 格式进行存储, 因此, 矩阵  $\mu_{ij}$  可对应为:  $\mu AD, \mu AL, \mu NL, \mu NJL, \mu AU, \mu NU, \mu NJU$ 。

### 3.3 存储格式的比较

图 1 给出了稠密矩阵存储格式与 MCSC、OCSC 存储格式在存储空间的占比情况。

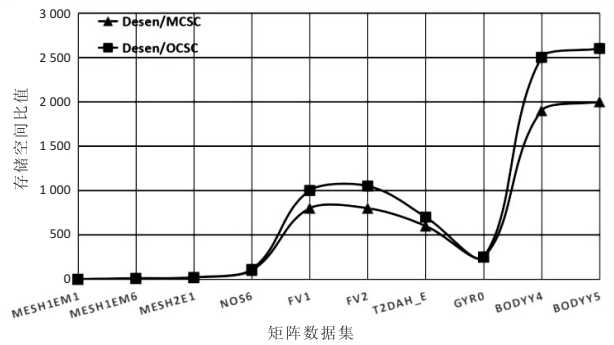


图 1 存储空间比值

从图 1 可以明显看出, MCSC 和 OCSC 与稠密存储空间的比值随着矩阵阶数、矩阵的稀疏性和非零元素数量规模的增大而增大, 说明 MCSC 和 OCSC 格式所需的存储空间远少于稠密存储空间, 且 OCSC 格式所需的存储空间比 ACSC 格式的存储空间更少。

## 4 基于列压缩存储格式的动态负载均衡

### 4.1 负载均衡存储格式划分

对稀疏线性方程组求解的并行算法首先需对矩阵进行数据划分, 划分的方式可以有列划分、行划分。文中采用按列数相同的列块顺序进行存储格式划分, 确保各线程进行计算时的列块中的列数相等。

由于  $A$  的每一列中所包含非零元个数不一定相同, 对应分配到各线程中的列块中包含的非零元数往往也不一定相同, 很明显, 各线程对列的计算量与该列中的包含非零元素的个数成正比, 这很容易引发线程负载的不均衡, 线程负载不均衡势必进一步引发部分处理机的资源空闲浪费。因此, 列划分应尽量使分配给各个线程计算列块中所包含的非零元素的个数差异性不大, 以确保形成均衡的负载。

### 4.2 动态列矩阵块的划分算法

系统线程调度、运行过程是动态的, 为确保建立动

态的负载均衡,而现代操作系统针对线程的调度都是建立在具体算法的基础上,下面是列矩阵块的动态划分算法步骤。

1: partition( seg[ ], NZ[ ], n, p ) //定义参数, seg[ ] 是存放矩阵列块划分方式的数组, NZ[ ] 是存放非零元素的数组, p 为线程数;

```
2: k = 0, q = 0, l = 0; seg[0] = 0 //初始化;
3: for all j ∈ 0, 1, ..., n do
4: l = l + NZ[ j ] //NZ[ j ] 为第 j 列非零元素个数;
5: end for
6: for all j ∈ 0, 1, ..., n do //迭代划分列块;
7: k = k + NZ[ j ]
8: if k > [ q + 1 ] * l / p then
9: seg[ q + 1 ] = j; q = q + 1
10: end if
11: seg[ p ] = n + 1
12: end for
```

由于 p 为线程数,其结果是对应于 q = 0, 1, ..., p - 1 时,将稀疏矩阵的第 seg<sub>q</sub> 到 seg<sub>q+1</sub> - 1 列分配到线程号 q 中,这就产生两种运行状态需要进行处理。

节点状态处理:每一轮迭代结束均判定节点状态,如果处于消息动态均衡,则不作计算处理,以减少列块中待计算节点的数量。

动态均衡处理:为防止因减少计算节点引起新的负载不均衡,新一轮的迭代需重新对列块进行划分,并重新分配给相应线程,让每一线程重新达到相同的计算量。以此类推,达到动态负载均衡。

### 4.3 DLB\_GaBP 算法

DLB\_GaBP 算法的实现实质上就是将 GaBP 算法中的消息累加、消息传播与更新、求解向量分别加入到 OpenMP 的并行制导控制段中,构成三段多核并行控制段,即基于 OpenMP 的多核并行 GaBP 算法。该算法可表述如下:

```
Initialization: //初始化
1: init();
   Iteration
2: repeat
3: if( iters mod D_LOOP = 0 ) then //列块划分迭代步数条件判断
4: partition( seg, NZ, n, Nthreads )
5: end if
6: # pragma omp parallel private ( tid, p, j ) threads ( NT-
   HREADS )
7: { tid = omp_get_thread_num();
8: for( i = seg[ tid ]; j < seg[ tid + 1 ]
   && Ti == 0; i ++ ) do
9: if( Ti == 0 ) then
10: pi = Aii + ∑k ∈ N(i) Pki; μi = bi + ∑k ∈ N(i) μki
```

```
13: for all j ∈ N(i) \ j do
14: Pij += - Aij2 / ( Pi - Pji ); μij = - Aij ( μi - μji ) / ( Pi - Pji )
15: end for
16: end if
17: end for
18: }
19: # pragma omp parallel private ( tid, p, j ) threads ( NT-
   HREADS )
20: { tid = omp_get_thread_num();
21: for( i = seg[ tid ]; j < seg[ tid + 1 ] && Ti == 0; i ++ ) do
22: pi = Aii + ∑k ∈ N(i) Pki; μi = bi + ∑k ∈ N(i) μki
23: xi = μi / Pi // xi 为收敛精度
24: if( Ti == 0 ) and ( xi converged ) then // xi 满足设定的
   收敛精度,该节点的 Ti 值设定为 1
25: Ti = 1; NZj = 0
26: end if
27: end for
28: }
29: until convergence; all xi converged
30: output: x* = [ xi ]
```

为能够尽量好地提高计算效率,文中通过迭代步来控制上述的有效组合,具体是通过将迭代的次数限制在一定数量范围内,超过即重新进行列块划分。即在 Partition( ) 函数前面添加了迭代步控制,每隔 D\_LOOP 迭代步重新进行列矩阵块的划分。D\_LOOP 的取值一般是依据划分列块中的列数,当然也可以是定量或变量,这只要在算法中进行相应的定义即可。

当收敛精度 x<sub>i</sub> 满足设定的精度要求时,返回该节点 T<sub>i</sub> = 1, NZ[ j ] = 0, 下一轮迭代计算时跳过该节点,同时,负载均衡处理中也不统计该节点的邻居节点数,这可以大大减少节点的计算负载量,但是需要进行列块的重新划分以实现新的负载均衡。

新一轮迭代计算前进行新的列块均衡划分,总体上减少了计算量,也保持了各线程的计算量相对均衡,进而解决了因线程同步造成的大量线程的长时间等待问题,同时加速了迭代计算。

## 5 数值试验

### 5.1 实验环境

实验在高性能集群系统中进行,系统环境配置如下:

CPU: Intel Xeon E5-2650; 内存: 96 GB;

众核加速卡: Intel MIC 卡 (61 核, 8 G 共享存储);

操作系统: Redhat Linux 6.3 X64;

编译器: 支持 Intel MIC 应用开发的 Intel 编译器。

### 5.2 测试结果及分析

测试数据采用稀疏矩阵集 (UFget)<sup>[17]</sup>, 文中所提



算法是基于对角占优线性方程组的求解,选取的用于测试算法的稀疏矩阵集见表 1。

表 1 用于测试算法的稀疏矩阵集列表

No	id	Group	Name	Rows	nonzerros
1	873	Pothen	Mesh1em1	48	306
2	874	Pothen	mesh1em6	48	306
3	875	Pothen	mesh2e1	306	2 018
4	222	HB	nos6	675	3 255
5	887	Norris	fv1	306	85 264
6	888	Norris	fv2	9 801	87 025
7	868	Pothen	bodyy4	17 546	121 938
8	869	Pothen	bodyy5	18 589	129 281
9	1 205	Oberwolfach	t2dah_e	11 445	176 117
10	1 435	Oberwolfach	gyro	17 361	1 021 159

图 2 给出了 DLB\_GaBP 算法与并行 GaBP 算法在相同的计算环境中的计算时间对比,可以看出 DLB\_GaBP 算法具有更好的计算效率。

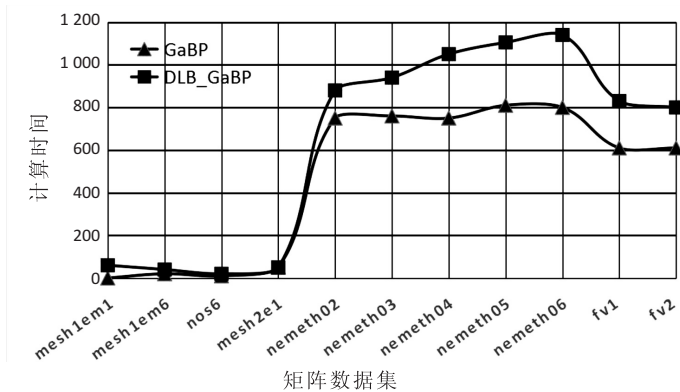


图 2 DLB\_GaBP 与 GaBP 算法计算时间对比

图 3 给出了 DLB\_GaBP 算法在相同运算规模而不同的运算线程或核心数环境中的加速比对比,从图中可看出随着算例计算规模的加大加速比也明显增大,表明算法具有良好的线程数或核数变化的加速效果。

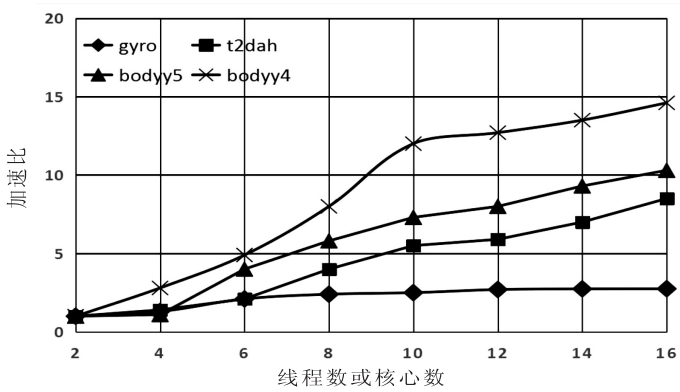


图 3 DLB\_GaBP 在不同线程数或核数中的加速比

图 4 是 DLB\_GaBP 与 GaBP 算法前后加速比对比图,算例矩阵名称是按其非零元素规模大小排列。

从图中可清楚看出,两种算法的加速比随着非零元素数量的增大而提高,尤其是非零元素规模不大的环境中,DLB\_GaBP 的加速优势并未呈现出来,随着非零元素个数的增加,即计算问题数据规模的增大,DLB\_GaBP 算法明显比 GaBP 算法有更好的加速比。

结合用于测试算法的稀疏矩阵集列表 1,观察图 4 的 DLB\_GaBP 算法的加速比对比曲线,稀疏矩阵的行数小于  $10^4$ ,非零元小于  $10^5$  的小规模计算问题,算法并没有加速效果,这是由于多核并行算法在启动多核并行占用的时间远比小规模算例实际求解计算的时间长。这说明问题求解的规模越大,算法的优越性越明显。

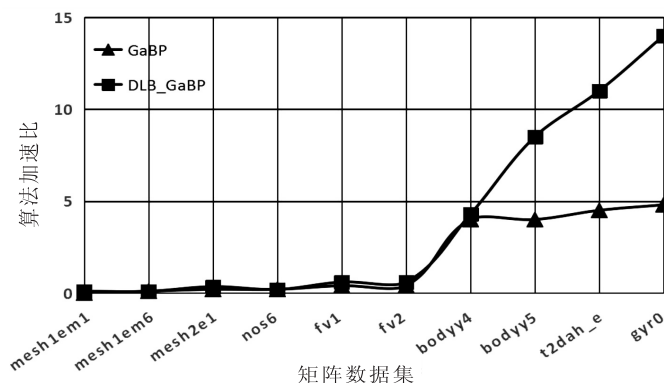


图 4 DLB\_GaBP 与 GaBP 算法加速比对比

## 6 结束语

通过建立列数相同的列块顺序进行存储格式定义划分,以确保各线程进行计算时的列块中的列数相等的基础上,实现在有限的内存空间对大规模稀疏线性方程组的左端矩阵存储,可降低算法的空间复杂度。同时通过挖掘基于 GaBP 的迭代加速优化方法本身的特性,建立具有负载均衡特性的 DLB\_GaBP 算法,具有适应性更好、迭代收敛速度更快的优势,更适用于大规模数值问题的计算,具有一定的推广应用价值。

### 参考文献:

- [1] 耿晓中. 基于多核分布式环境下的任务调度关键技术研究[D]. 长春: 吉林大学, 2013.
- [2] 刘正. 基于动态任务调度的 STDS 算法设计研究[J]. 智能系统学报, 2015, 10(2): 324-332.
- [3] 贾燕成, 黎英. 实时仿真并行调度算法研究[J]. 计算机工程, 2013, 39(1): 303-308.
- [4] 扈诗扬, 汪芳宗. 基于 GaBP 算法的快速潮流计算方法[J]. 计算技术与自动化, 2016, 35(4): 76-80.
- [5] MOËS N, DOLBOW J, BELYTSCHEV T. A finite element method for crack growth without remeshing[J]. International Journal for Numerical Methods in Engineering, 1999, 46(1): 131-150.
- [6] CHUNG T J. Computational fluid dynamics[M]. Cambridge: Cambridge University Press, 2002.
- [7] DOHA E H, BHRAWY A H, EZZELDIEN S S. A Chebyshev spectral method based on operational matrix for initial and boundary value problems of fractional order[J]. Computers & Mathematics with Applications, 2011, 62(5): 2364-2373.
- [8] ASHBY T J, GHYSELS P, HEIRMAN W, et al. The impact of global communication latency at extreme scales on Krylov methods[C]//12th international conference on algorithms and architectures for parallel processing (ICA3PP-12). Fukuoka: ACM, 2012: 428-442.
- [9] EL-KURDI Y, GROSS W J, GIANNACOPOULOS D. Efficient implementation of Gaussian belief propagation solver for large sparse diagonally dominant linear systems[J]. IEEE Transactions on Magnetics, 2012, 48(2): 471-474.
- [10] SHENTAL O, SIEGEL P H, WOLF J K, et al. Gaussian belief propagation solver for systems of linear equations[C]//2008 IEEE international symposium on information theory (ISIT 2008). Toronto: IEEE, 2008: 1863-1867.
- [11] ZHENG H, SONG A, LIU Z, et al. A GaBP-GPU algorithm of solving large-scale sparse linear systems[J]. Journal of Information & Computational Science, 2014, 11(3): 911-921.
- [12] 郑汉垣, 宋安平, 张武. 基于 GaBP 的迭代加速优化算法[J]. 航空计算技术, 2019, 49(3): 1-5.
- [13] 陈振武, 郑汉垣, 兰添才, 等. 求解大规模三对角线性方程组的 GaBP 并行算法[J]. 计算机工程, 2016, 42(10): 96-100.
- [14] PEARL J. Probabilistic reasoning in intelligent systems: networks of plausible inference[M]. San Francisco: Morgan Kaufmann, 1988.
- [15] WEISS Y, FREEMAN W T. Correctness of belief propagation in Gaussian graphical models of arbitrary topology[J]. Neural Computation, 2001, 13(10): 2173-2200.
- [16] 程凯, 田瑾, 马瑞琳. 基于 GPU 的高效稀疏矩阵存储格式研究[J]. 计算机工程, 2018, 44(8): 54-60.
- [17] DAVIS T, HU Y. The university of Florida sparse matrix collection[J]. ACM Transactions on Mathematical Software, 2011, 38(1): 1-25.