计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2019,28(10):138−144 [doi: 10.15888/j.cnki.csa.007119]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于 PRAM 并行模型最大值查找的方法与改进①

贺　成, 施华君

(中国电子科技集团公司第三十二研究所, 上海 201808)
通讯作者: 贺　成, E-mail: ephe@mail.ustc.edu.cn

摘　要: 随着多处理器的出现, 并行技术受到了广泛的关注, 成为了加速处理问题速度的重要技术. 但是使用并行技术在加速计算的同时也带来了对处理器数量需求的急剧提升, 并行成本的显著增加. 针对这一问题, 通过研究基于 PRAM (Parallel Random Access Machine) 下的 3 种最大值查找并行算法中的不足, 提出了一种比平衡树算法, 快速查找法, 双对数深度树方法并行成本 (cost) 更优的基于数据划分方法的最大值查找并行算法. 基于数据划分方法的最大值查找算法有效的解决了现有并行方法中处理器工作量分配不均, 对处理器需求过大, 实现条件苛刻等问题. 为此后类似并行算法降低并行成本提供一个方向.

关键词: 并行算法; 最大值; 并行成本; 并行时间; 并行机

引用格式: 贺成,施华君.基于 PRAM 并行模型最大值查找的方法与改进.计算机系统应用,2019,28(10):138–144. http://www.c-s-a.org.cn/1003-3254/7119.html

Method and Improvement of Maximum Search Based on PRAM Parallel Model

HE Cheng, SHI Hua-Jun

(The 32nd Research Institute of China Electronic Technology Group Corporation, Shanghai 201808, China)

Abstract: With the emergence of multiprocessors, parallel technology has attracted widespread attention and become an important technology to speed up the processing of problems. Nevertheless, the use of parallel technology to speed up computing has also led to a sharp increase in the number of processors and a significant increase in parallel costs. To solve this problem, by studying the shortcomings of three parallel maximum search algorithms based on PRAM (Parallel Random Access Machine), a parallel maximum search algorithm based on data partitioning method is proposed, which is better than the balanced tree algorithm, fast search method and double logarithmic depth tree method. The maximum searching algorithm based on data partitioning method effectively solves the problems of uneven workload allocation, excessive demand for processors and harsh implementation conditions in existing parallel methods. This provides a direction for similar parallel algorithms to reduce parallel costs.

Key words: parallel algorithms; maximum; parallel cost; parallel time; parallel computer

当今许多尖端技术的解决都离不开并行处理, 许多国家都将其列入关键技术. 并行机的发展和智能计算等实际应用的需要共同促进了并行算法的研究[1,2]. 算法是求解问题的步骤和方法[3]. 简单的讲, 并行算法是用多台处理器联合求解问题的方法和步骤[4,5]. 并行算法不能仅通过时间复杂度[6]来进行评价, 因为人们在对计算速度的渴求的同时, 对于并行处理所需要的处理器数量急剧增长的问题也开始关心. 所以, 评判并行算法综合性能的指标“成本”显得格外的重要. 并行算法的成本是指并行算法在并行机各处理器上运行时间总和$Cost = Tp * P$[7], Tp 代表每个处理器处理问题的运行时间, P 代表处理器数量. 并行算法相比串行算法而

① 收稿时间: 2019-03-20; 修改时间: 2019-04-17, 2019-04-23; 采用时间: 2019-04-26; csa 在线出版时间: 2019-10-15

言, 可以在更短的时间内给出问题的答案, 满足了应用对于实时性的要求. 但是并行算法所带来的额外开销也是不容忽视的, 其中并行算法对于处理器数量的要求导致并行算法的实现是昂贵的. 这无疑给有些并行算法的实现带来一定的限制. 其中并行算法中最大值查找就存在成本过高的问题.

本文以最大值查找算法为基础, 探讨了 3 种并行化查找最大值的方法 (平衡树方法、双对数深度树方法、快速查找法), 进而提出提出了一种比平衡树算法, 快速查找法, 双对数深度树方法并行成本 (Cost) 更优的基于数据划分方法的最大值查找并行算法. 该方法通过减少处理器使用的数量, 维持处理问题时间与平衡树方法一致, 以此取得相比平衡树方法, 双对数深度树方法, 快速查找法在并行成本上表现更优的算法. 为之后的相关并行算法的优化提供一个改进的方向, 为大数据处理中查找最大值提供了一个可行的办法.

本文并行算法研究是在 PRAM[8,9]模型下进行的, 但微型计算机 CPU 架构中的 SIMD 并行机制, 由于实现指令的复杂性, 几乎没有编译程序将高级语言设计的程序优化到 SIMD 指令级别[10]. 因此本文通过模拟仿真实验对几种并行查找算法的实际运行时间进行对比.

1 并行查找最大值的算法

1.1 平衡树查找方法

1.1.1 算法原理

利用平衡二叉树[11−13](Balanced Binary Tree) 抽象出并行求解最大值的方法, 树的叶节点是输入元素集合, 树的非叶节点用于比较操作, 树的高度为 H. 在树的同一深度上内节点并行计算. 树中的节点都包含两个子节点, 所以每个节点只需要一个处理器就可以在常数时间内做出比较操作, 同一深度并行计算需要的处理器数目与内节点数相同, 这样仅需要 O(H) 的时间就可以完成最大值查找. 需要处理器数目为 $2H$–1. 如图 1.

下面是平衡树方法的伪代码[14]:

输入: $n=2^m$ 个数存于数组 $A(n{:}2n{-}1)$ 中.
输出: 最大值, 位于 $A(1)$.

```
Balance(Element A[])
{
  for k=m–1 to 0 do
    for j=2^k to 2^(k+1)–1 par-do
      A[j]=max{A[2j], A[2j+1]}
}
```

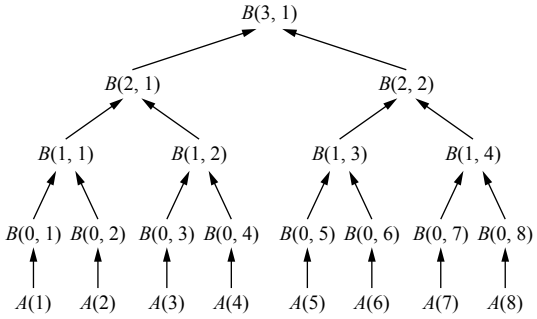

图 1 平衡树查找最大值

1.1.2 算法分析

平衡二叉树算法的运行时间为$Tp=O(\log N)$, 处理器数目$P(n)=O(N/2)$, 并行成本$Cost=Tp*P=O(N\log N)$. 从时间复杂度上来看,平衡树算法运行速度较快. 该方法能快速缩减待比较的数据量, 因此速度得到了极大的提升. 但是由于每次操作之后待比较数据都减少一半, 所以每轮操作之后都有一半当前参与工作的处理器无事可做. 直到树根时, 仅仅有一个处理器在工作. 平衡树方法的优点是可以快速缩减待比较数据量, 从而提高查找速度. 但是由于其自身结构的原因, 导致了处理器负载不均衡[15,16], 处理器利用率不高.

1.2 快速查找方法

1.2.1 算法原理

通过比较手段快速获取集合中的最大值[17], 每次操作就需要确定更多元素之间的大小关系, 也就是每次操作需要更多元素进行比较. 快速并行算法利用一个 $n*n$ 的逻辑二维表来存储某一个元素与集合中其余元素的大小关系 (0 表示小于, 1 表示不小于), 如表 1. 当某一行或者某一列元素关系都为“大于”时, 该行 (列) 代表的元素为最大值.

表 1 逻辑关系表

	a1	a2	…	an
a1	1	0	…	0
a2	1	1	…	1
…	0	0	…	0
An	1	0	…	1

下面是快速并行算法的伪代码[14]:

输入: 存有 n 个不同元素的数组 A.
输出: 布尔数组 M, 当且仅当 $A(i)$=max{A}时, $M(i)$=1.

```
Quick(Element A[])
{
(1) for 1<=i, j<=n par-do
```

```
  if A[i] >= A[j]
     B[i][j] = 1
  else
     B[i][j] = 0
(2) for 1<=i<=n par-do
  M[i]=B[i][1]∩B[i][2]∩… ∩B[i][n]
     }
```

该算法要求步骤 (1) 同时读, 步骤 (2) 要求同时写.

1.2.2 算法分析

算法执行时间为$Tp(n)=O(1)$, 使用了$P(n)=O(N^2)$个处理器, 成本为$Cost=Tp*P=O(N^2)$. 从时间复杂度上看, 该算法是目前已知运行速度最快的最大值查找并行算法. 与平衡树方法和双对数方法每轮操作只能确定部分元素之间大小关系, 该算法每轮操作都可以确定一个元素与其余所有元素的大小关系. 利用这一并行思想可以快速确定所有元素之间的关系, 但是该方法缺点就是所需要的处理器数目过多, 处理器数目将会以元素个数的指数增长, 并且对于处理器之间的通信速度要求极高, 算法实现要求条件苛刻.

1.3 双对数深度树查找方法

1.3.1 算法原理

所有输入数据作为叶节点, 节点 u 的子节点数为$\sqrt{N_u}$, 其中 N_u 是节点 u 所包含的叶节点的数目. 规定节点 u 的层为 u 到根路径的边数, 显然根节点的层为 0[14]. 假定共有元素$N=2^{2^k}$, 则第 i 层有$2^{2^k-2^{k-i}}$个节点, 每个节点有$2^{2^{k-i-1}}$个子节点. 由此可以计算出树的高度为$k+1=loglogN$. 图 2 为当 N=16 时的双对数深度树[18].

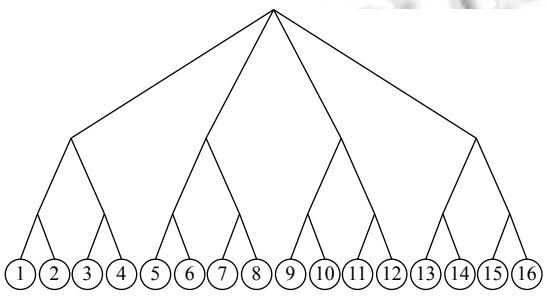

图 2 双对数深度树

下面是双对数深度树方法的伪代码:

```
输入: n =2^(2^k)个数存于数组 A(1:n) 中.
输出: 布尔数组 M, 当且仅当 A(i)=max{A}时, M(i)=1.
DoubleTree(Element A[])
{
(1) for j=1 to n/2 do
   A[j]=max{A[2j–1], A[2j]}
(2) for i=k–1 to 0 do
   for m=0 to m=2^(2^k−2^(k−i)) par-do
      Quick(A[m*2^(2^(k−i−1)):(m+1)*2^(2^(k−i−1))])
}
```

1.3.2 算法分析

算法执行时间$Tp(n)=O(\log\log N)$, 使用了$P(n)=O(N)$个处理器, 成本为$Cost=Tp*P=O(N\log\log N)$. 双对数深度树每次比较操作之后, 待比较数据量减少的速度比平衡树方法更快, 每个处理器的工作量相比平衡树方法都有更好的提升, 所以双对数深度树的查找时间比平衡树更快. 但是双对数深度树的每个节点的最大值都是通过调用快速查找算法而实现的, 所以双对数深度树的方法的实现的难度仍然很大.

2 基于数据划分的最大值查找算法

基于数据划分的最大值查找算法 (以下简称为“数据划分方法”) 针对平衡树算法处理器工作分配不均导致工作效率低下, 快速查找算法对于处理器数量需求过大, 以及双对数深度树方法难以实现的问题进行了改进. 汲取了双对数深度树中处理器工作饱满, 查找时间快, 快速查找算法逻辑表比较数据法的优点. 提出了一种比平衡树算法, 快速查找法, 双对数深度树方法并行成本 $Cost$ 更优的基于数据划分方法的最大值查找并行算法, 通过降低处理器 (P) 数量, 保证处理时间 (Tp) 达到 O(log$n$) 量级, 来达到降低并行成本的目的.

数据划分方法使用比第一节中 3 种并行方法更少的处理器, 保证处理速度达到 O(log$n$) 级, 就必须充分利用每个处理器的工作效率, 从而保证数据划分方法不会因为处理器数目的减少, 而使得问题处理速度远远慢于平衡树方法. 研究过程中发现利用逻辑表将数据进行合理的划分为多个组, 可以提升每个处理器的效率, 且能快速降低待查找数据集.

最终实现了一种比平衡树算法, 快速查找法, 双对数深度树方法并行成本额 $Cost$ 更优的高度可行的基于数据划分查找最大值的并行算法.

2.1 算法原理

从逻辑上将集合中的数据进行分组, 假设共 N 个

元素, 分为 log$N$ 组, 每组 N/log$N$ 个元素, 共 N/log$N$ 个处理器.

Step1. 每组都采取两两元素比较的方式, 使每组元素个数都减少为 N/(2log$N$) 个元素. 由于每组需要 N/(2log$N$) 个处理器, 所以每两组可以并行计算, 共需要 (log$N$)/2 步完成所有操作.

Step2. 此时每组所剩元素为 N/(2log$N$) 个, 继续采用两两比较的方式, 使每组元素个数都减少为 N/(4log$N$). 由于每组需要 N/(4log$N$) 个处理器, 所以每四组可以并行计算, 共需要 (log$N$)/4 步完成所有操作.

Step3. 此时每组所剩元素为 N/(4log$N$) 个, 继续采用两两比较的方式, 使每组元素个数都减少为 N/(8log$N$). 由于每组需要 N/(8log$N$) 个处理器, 所以每八组可以并行计算, 共需要 (log$N$)/8 步完成所有操作.

依次类推, 最终的目的是使得每组剩余元素个数为 1. 而此时需要 log(N/2log$N$) 步完成. 当每组只剩 1 个元素的时候, 就可以采用平衡树或双对数深度树进行求解最终的最大值.

下面是基于数据划分算法的伪代码:

输入: n 个数存放于 A[log$n$][log$n$]数组中
输出: 最大值 A[1][1]

```
Data_Partition(Element A[][])
{
  for i = 1 to log(n/(2logn))
    for m=1 to logn par-do
      blance(A[m][1:(logn)/i])
  Blance(A[1:logn][1])
}
```

2.2 算法分析

求解组内最大值共需要 $\frac{(N-1)}{N}\log N=\log\frac{N}{2}+\log\frac{N}{4}+\cdots+\log\frac{N}{2\log N}$, 若采取平衡二叉树方法求解组内最大值的最大值, 则算法最终的运行时间为 $Tp(n)=\mathrm{O}\left(\frac{N-1}{N}\log N+\log N\right)=\mathrm{O}(\log N)$, 处理器数目为 $P(n)=\mathrm{O}\left(\frac{N}{\log N}\right)$, 运行成本为 $Cost=Tp*P=\mathrm{O}(N)$, 数据划分方法的时间复杂度与平衡树方法相同, 所需处理器数目比第一节中三种并行方法需要处理器的数量更少. 因此使得该算法的综合成本相比平衡树方法, 快速查找方法, 双对数深度树方法更优, 由于不需要类似于快速查找方法中对于处理器之间通信的苛刻要求, 所以数据划分方法相较于快速查找法与双对数深度树方法而言, 其可行性更高.

3 实验方式

Pan[19]和 Pavel[20]提出了并行计算模型具备可重配置流水线总线的线性阵列模型 (Linear Arrays with a Reconfigurable Pipelined Bus System, LARPBS), 并且在该模型上实现了平衡树与双对数深度树查找的并行算法[21]. 由于实验条件所限, 无法实现真正搭建并行机和 LARPBS. 鉴于此种情况, 本文采用多核并行计算进行仿真模拟实验. 深度学习[22]中神经网络利用 Tensorflow 学习库来构建计算图, 通过 GPU 进行计算图中部分节点并行计算, 从而实现并行加速效果. 因此本文也使用基于 TensorFlow 的 GPU[23,24]处理框架来进行仿真模拟实验.

3.1 实验原理

利用 TensorFlow 构建各个并行查找算法的计算图, 如图 3–图 6(为了显示方便, 图中只显示一定数据量下的计算图), 将计算图中的每个节点视为并行机中的一个处理器, 计算图中每个节点的连线 (数据传递) 代表处理器之间的通信. 通过 TensorFlow 自带的 GPU 加速来实现同计算层次中节点的并行计算, 以此来仿真并行机运行查找算法的过程. 实验测试数据共分为 5 组, 每组数据通过随机数生成, 每组数据个数依次为 512, 1024, 2048, 4096, 8192. (为了简化程序将数据量都设置为 2^n). 每组数据测试 1000 次取运行时间的平均值, 作为最后实验分析数据.

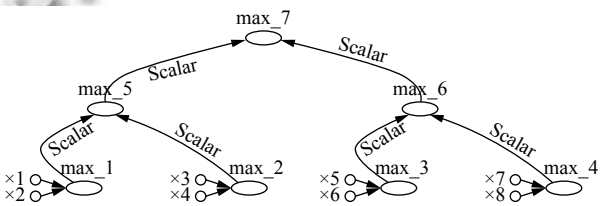

图 3 数据量为 8 的平衡二叉树方法 tensorflow 计算图

3.2 实验环境

操作系统: Windows7
编程语言: Python3.6
机器学习库: Tensorflow-GPU 1.12.0
处理器: CPU i7-4790 3.6 GHz
加速器: GPU GTX1070
编辑器: Pycharm Community Edition 2017.1.2

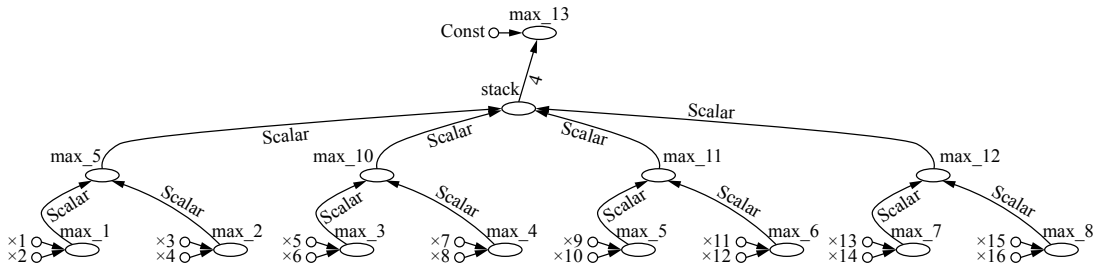

图 4 数据量为 16 的双对数深度树 tensorflow 计算图

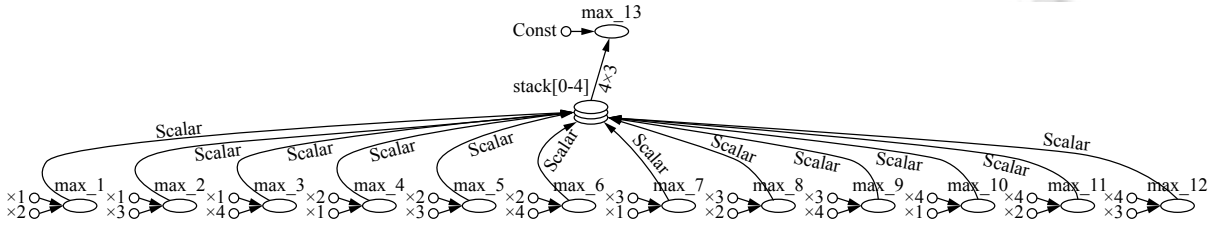

图 5 数据量为 4 的快速查找算法计算图

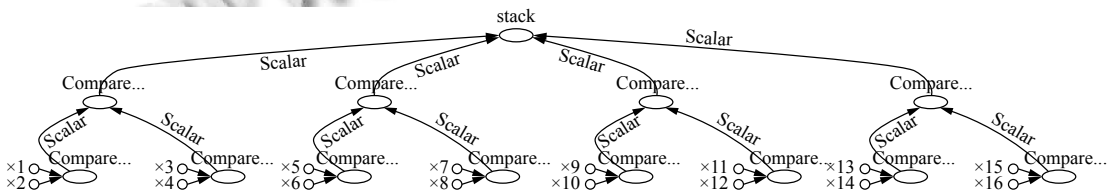

图 6 数据量为 16 的数据划分方法计算图

3.3 实验结果与分析

从数据划分方法运行时间 (见表 2, 图 7) 来看, 数据划分方法随着测试数据量的增加其实际运行时间的增长符合理论时间复杂度的增长规律, 说明仿真模拟实验从运行时间结果上来看是满足实验需要的, 实验数据具有一定的可参考性. 表 2 中有一个现象, 快速查找算法理论时间复杂度是O(1), 但是实际运行时间却跟双对数深度树方法几乎一样, 甚者运行时间表现的更糟糕. 这是由于快速查找算法的理想模型机是 MIMD (Multiple Instruction stream Multiple Data stream), tensorflow 搭建的计算图并不能模拟 MIMD 所具有的性质, 因此运行时间表现糟糕. 而改进算法的实际运行时间与平衡树算法运行时间几乎一致, 较其他算法略慢, 但是达到了改进算法最初对其运行时间的要求.

从运行所需的处理器数量来看 (见表 3, 图 8), 随着数据量的增长, 快速查找方法所需要的处理器数量爆炸增长, 因而在当前坐标轴下已经无法看到快速查找方法的曲线, 而数据划分方法所需要的处理器数量远低于平衡树查找法和双对数深度树查找法, 并且随着数据量的增多, 数据划分方法对处理器数量的需求相较其他查找算法优势越大. 达到了数据划分方法最初对其所需处理器数量的要求.

表 2 实验结果时间对比表

	平衡树	对数树	快速查找	改进算法
时间复杂度	O(log*N*)	O(log log*N*)	O(1)	O(log*N*)
实际运行平均时间 (s) (测试数据量 *N*=512)	0.0058	0.0022	0.0009	0.0061
实际运行平均时间 (s) (测试数据量 *N*=1024)	0.0071	0.0022	0.0011	0.0074
实际运行平均时间 (s) (测试数据量 *N*=2048)	0.0079	0.0023	0.0015	0.0083
实际运行平均时间 (s) (测试数据量 *N*=4096)	0.0085	0.0023	0.0018	0.0090
实际运行平均时间 (s) (测试数据量 *N*=8192)	0.0091	0.0024	0.0026	0.0098

从运行成本来看 (见表 4, 图 9), 由于快速查找方法并行成本过高, 因而在当前坐标轴下已经无法看到快速查找方法的曲线. 数据划分方法的综合成本相比于其他最大值查找的并行算法是更优的, 说明数据划分方法是有效的, 达到了研究目的.

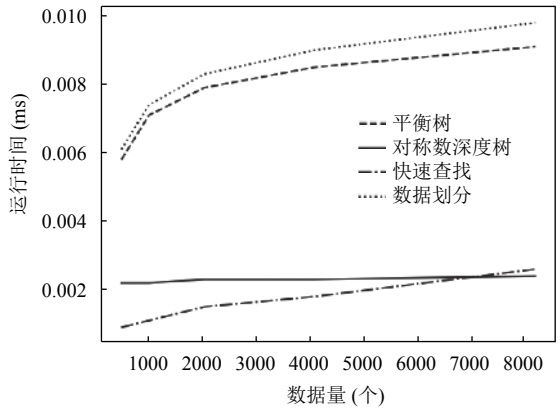

图 7 算法运行时间对比图

表 3 模拟处理器数量对比表

	平衡树	对数树	快速查找	改进算法
处理器数量复杂度	$O\left(\frac{N}{2}\right)$	$O(N)$	$O(N^2)$	$O\left(\frac{N}{\log N}\right)$
处理器数量 (测试数据量为 512)	256	512	262 144	57
处理器数量 (测试数据量为 1024)	512	1024	1 048 576	103
处理器数量 (测试数据量为 2048)	1024	2048	4 194 304	187
处理器数量 (测试数据量为 4096)	2048	4096	16 777 216	342
处理器数量 (测试数据量为 8192)	4096	8192	67 108 864	631

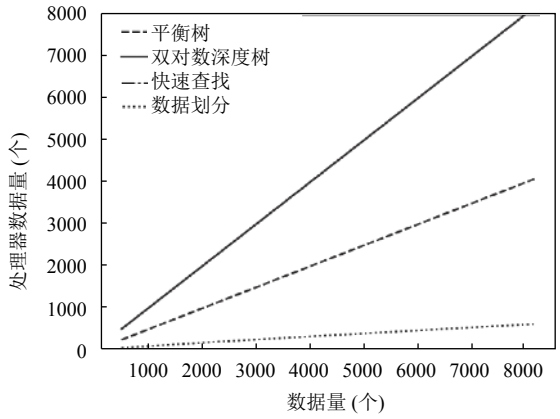

图 8 算法使用处理器数目对比图

4 结论与展望

通过对比实验, 数据划分方法的并行成本 (Cost) 相较平衡树方法, 双对数深度树方法, 快速查找法是更优的, 说明数据划分方法到达了最初的研究目的. 数据划分方法使用更少处理器, 有效降低了所需处理器数量, 保证了问题处理时间达到并行处理的目的, 使其执行时间与平衡树方法一致的改进算法, 略低于双对数深度树方法. 最终减少并行算法的成本, 节约了资源.

表 4 运行成本对比表

	平衡树	对数树	快速查找	改进算法
成本复杂度	$O(N\log N)$	$O(N\log\log N)$	$O(N^2)$	$O(N)$
成本 (测试数据量为 512)	1.4848	1.1264	235.9296	0.3477
成本 (测试数据量为 1024)	3.3652	2.2528	1153.4366	0.7622
成本 (测试数据量为 2048)	8.8096	4.7104	6291.456	1.5521
成本 (测试数据量为 4096)	17.408	9.4208	30 198.9888	3.078
成本 (测试数据量为 8192)	32.2736	19.6608	174 483.0464	6.1838

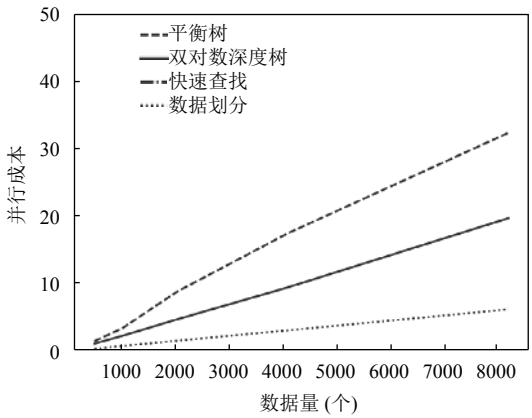

图 9 算法成本对比图

随着并行技术越来越多的应用到学术研究, 生产生活等方面, 人们对于算法运行产生的并行成本的要求会更高. 希望可以保证处理速度的同时, 降低并行算法的成本. 本文最大值查找算法的改进, 就是在此背景下进行的. 并行算法众多, 文章通过比较具有代表性的最大值查找算法进行改进. 为此后类似并行算法降低并行成本提供一个方向. 随着更多学者和专家的投入, 相信会有更多具有高额成本的并行算法在快速解决问题的同时, 有效减低并行技术产生巨额的成本.

参考文献

1 王青云, 罗泽. 基于 LOD 的海量地形数据并行渲染技术. 计算机系统应用, 2017, 26(12): 200–206. [doi: 10.15888/j.cnki.csa.006113]
2 李晓梅. 并行算法的发展及其前沿研究课题. 中国科学基金, 1995, (3): 13–18.
3 严蔚敏, 李冬梅, 吴伟民. 数据结构 (C 语言版). 计算机教

育, 2012, (12): 62.
4 罗贵章, 陈忠伟. 并行算法综述. 计算机光盘软件与应用, 2013, (15): 51–52.
5 陈国良, 孙广中, 徐云, 等. 并行算法研究方法学. 计算机学报, 2008, 31(9): 1493–1502. [doi: 10.3321/j.issn:0254-4164.2008.09.002]
6 Shiloach Y, Vishkin U. Finding the maximum, merging, and sorting in a parallel computation model. Journal of Algorithms, 1981, 2(1): 88–102. [doi: 10.1016/0196-6774(81)90010-9]
7 胡峰, 胡保生. 并行计算技术与并行算法综述. 电脑与信息技术, 1999, (5): 47–59.
8 Acharya HB. Parallel processing of packets with a PRAM. Proceedings of the 19th International Conference on Distributed Computing and Networking. Varanasi, India. 2018. 45.
9 Forsell M, Leppänen V. An extended PRAM-NUMA model of computation for TCF programming. Proceedings of the IEEE 26th International Parallel and Distributed Processing Symposium Workshops & Phd Forum. Shanghai, China. 2012. 786–793.
10 明玉瑞, 李思泽. 基于 SIMD 机制的并行排序算法. 计算机系统应用, 2009, 18(11): 87–90. [doi: 10.3969/j.issn.1003-3254.2009.11.022]
11 赵军, 张东梅. 平衡二叉树. 电脑学习, 2007, (2): 33–34. [doi: 10.3969/j.issn.2095-2163.2007.02.018]
12 陈海涛, 李宗惠. 平衡二叉树的失衡调整方法探讨. 中国科教创新导刊, 2010, (34): 146, 148.
13 Ha PH, Anshus OJ, Umar I. Efficient concurrent search trees using portable fine-grained locality. IEEE Transactions on Parallel and Distributed Systems, 2019, 30(7): 1580–1595. [doi: 10.1109/TPDS.71]
14 陈国良. 并行算法的设计与分析. 3 版. 北京: 高等教育出版社, 2009. 63–64, 78–80.
15 Huang D, Han DZ, Wang J, *et al*. Achieving load balance for parallel data access on distributed file systems. IEEE Transactions on Computers, 2018, 67(3): 388–402. [doi: 10.1109/TC.2017.2749229]
16 Jackson A, Campobasso MS, Drofelnik J. Load balance and parallel I/O: Optimising COSA for large simulations. Computers & Fluids, 2018, 173: 206–215.
17 Berkman O, Matias Y, Vishkin U. Randomized range-maxima in nearly-constant parallel time. Computational Complexity, 1992, 2(4): 350–373. [doi: 10.1007/BF01200429]
18 Olariu S, Overstreet M, Wen ZF. Reconstructing a binary tree from its traversals in doubly logarithmic CREW time. Journal of Parallel and Distributed Computing, 1995, 27(1): 100–105. [doi: 10.1006/jpdc.1995.1075]
19 Pan Y, Hamdi M. Quicksort on a linear array with a reconfigurable pipelined bus system. Proceedings of the 2nd International Symposium on Parallel Architectures, Algorithms, and Networks. Beijing, China. 1996. 313–319.
20 Pavel S, Akl SG. Efficient algorithms for the hough transform on arrays with reconfigurable optical buses. Proceedings of 1996 International Conference on Parallel Processing. Honolulu, Hawaii, USA. 1996. 697–701.
21 李庆华, 蒋廷耀. 基于 LARPBS 模型的最大值查找算法. 计算机科学, 2004, 31(3): 183–185. [doi: 10.3969/j.issn.1002-137X.2004.03.052]
22 陈旭, 孟朝晖. 基于深度学习的目标视频跟踪算法综述. 计算机系统应用, 2019, 28(1): 1–9. [doi: 10.15888/j.cnki.csa.006720]
23 梁娟娟, 任开新, 郭利财, 等. GPU 上的矩阵乘法的设计与实现. 计算机系统应用, 2011, 20(1): 178–181, 149. [doi: 10.3969/j.issn.1003-3254.2011.01.038]
24 Campos V, Sastre F, Yagües M, *et al*. Scaling a convolutional neural network for classification of adjective noun pairs with TensorFlow on GPU clusters. Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. Madrid, Spain. 2017. 677–682.