软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2019,30(12):3622−3636 [doi: 10.13328/j.cnki.jos.005599]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

多步前进同步并行模型*

张尉东[1], 崔 唱[2]

[1](北京大学 信息科学技术学院,北京 100871)

[2](北京大学 元培学院,北京 100871)

通讯作者: 张尉东, E-mail: zhangwd@pku.edu.cn

摘 要: 提出一种并行计算模型——多步前进同步并行(delta-stepping synchronous parallel,简称 DSP)模型和一种形式化表示方法.针对大同步并行(bulk synchronous parallel,简称 BSP)模型同步次数多、收敛速度慢的特点,该模型能够有效地减少同步次数和通信开销,进而加速算法的收敛.通过形式化表示和迭代过程推导,发现 DSP 是一种比 BSP 更一般的并行计算模型.在 BSP 的基础上,DSP 将 BSP 中执行 1 次的局部计算变为执行多次.理论分析和验证实验表明,新增加的局部计算步可以进一步挖掘和利用隐藏在数据分区中的局部性.同时,通过"计算换通信"原理增加的局部计算并非越多越好.最后的实验结果显示,DSP 模型能够有效地效减少算法的迭代轮数及收敛时间,对 BSP 的加速可高达到数倍乃至数十倍.

关键词: 并行计算模型;图并行算法;单源最短路算法;PageRank;雅各比迭代算法;随机梯度下降

中图法分类号: TP311

中文引用格式: 张尉东,崔唱.多步前进同步并行模型.软件学报,2019,30(12):3622−3636. http://www.jos.org.cn/1000-9825/5599.htm

英文引用格式: Zhang WD, Cui C. Delta-stepping synchronous parallel model. Ruan Jian Xue Bao/Journal of Software, 2019, 30(12):3622−3636 (in Chinese). http://www.jos.org.cn/1000-9825/5599.htm

Delta-Stepping Synchronous Parallel Model

ZHANG Wei-Dong[1], CUI Chang[2]

[1](School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China)

[2](Yuanpei College, Peking University, Beijing 100871, China)

Abstract: In this study, a parallel computation model named delta-stepping synchronous parallel (DSP) is introduced, which is a more general form than BSP (bulk synchronous parallel). Compared with BSP, delta steps of local computation substitute the single local computation in each superstep. The added local computation is named as speculative computation step (SCStep). SCStep could further explore the locality hidden in data and accelerate value diffusion. It turns out to be dramatically effective on reducing the number of iterations and shortening the convergence time. Meanwhile, it is found that excessively using the SCStep is not appropriate considering the increased computation overhead. To identify applicable algorithms and also prove the correctness, the iterative process is formalized and the convergence condition is deduced. Finally, case studies and evaluations show that DSP model could significantly reduce the number of iterations and shorten the convergence time by dozens of times.

Key words: parallel computing model; graph parallel algorithm; single source shortest path algorithm; PageRank; Jacobi iterative method; stochastic gradient descent

在过去的十几年中,各种用途的处理框架相继被开发出来,如为通用数据处理而开发的 Hadoop/Spark/

* 基金项目: 国家重点研发计划(2017YFB0202001); 国家自然科学基金(61432018, 61672208)

Foundation item: National Key Research and Development Program of China (2017YFB0202001); National Natural Science Foundation of China (61432018, 61672208)

收稿时间: 2017-12-01; 修改时间: 2018-04-21; 采用时间: 2018-05-11

Dryad[1−3],为蛋白质折叠而开发的 GROMACS[4],为海洋、气象气候科学而开发的 CORSIKA[5]以及为材料工程科学、天文天体物理和社会科学开发的 NAMD[6]、FLASH[7]和 Swarm[8]等.

无论是并行计算框架还是并行算法,并行化都需要一种或多种并行计算模型来指导其实现.常见的并行计算模型包括大同步并行(bulk synchronous parallel,简称 BSP)[9]、异步并行(asynchronous iterative algorithm,简称 AiA)[10]、时间并行(parallel in time,简称 PiT)[11]、logP[12]和轮回并行(samsara parallel,简称 SP)[13]等.然而,所有这些并行计算模型都采用无差别的方式来处理稀疏和稠密数据集,这就忽略了数据集内部所蕴涵的独特性(如局部性分布、依赖关系密度等).这不仅导致了大量的无用计算和迭代步,还导致计算时间和收敛结果不受控制.一个经典的例子是使用通用计算机集群(commodity machine cluster)在两个稀疏程度不同但直径相同的图上执行 BSP 模型并行化的单源最短路算法,其所耗时间几乎相等.原因是它们使用了相同的迭代轮数,而每轮迭代的时间基本都消耗在全局同步上.

为了追求更快的收敛速度,各种应用的异步算法相继被设计出来.异步算法可以在时间上重叠计算和通信,从而实现更灵活地利用数据内部的局部性,达到加速算法收敛的目的.异步算法的缺点也比较明显.除了需要针对单个问题单独设计相应的算法和终止条件外,还会比同步算法耗费更多的通信和计算.SP 与 BSP 和 AiA 相比,其每次向前投机地计算多步,再将多步计算结果统一打包并交换.在将交换的数据校验后,若发现某步结果不对,则将所有进程跳回最早出错的一步重新计算.尽管 SP 可能进行更多的无用计算,但每次大同步却可以投机地前进多步,且每步都能和 BSP 的迭代步一一对应.

就我们所知,目前还没有一个模型可以在不修改算法的前提下,以不同的方式处理稀疏程度不同的数据集.要进行有区别的处理,首先需要挖掘数据中存在依赖关系和局部性分布,即使先不讨论依赖关系和局部性,如何构造出根据不同的数据集产生不同并行行为的并行程序也缺乏相应的理论指导.在日常数据处理工作中,我们发现大量并行算法仅在满足弱一致性条件时即可收敛;同时,大多数数据都是相当稀疏的.结合这两方面,我们认为可以通过进一步挖掘和利用数据分区中的局部性,首先加速局部收敛进而促进全局的收敛.

通过数学建模和形式化推导,我们发现本文提出的多步前进同步并行(delta-stepping synchronous parallel,简称 DSP)模型是一种比 BSP 更一般的同步并行模型.它可以更充分地挖掘和利用隐藏在数据中的局部性来加速算法的收敛.进一步的实验同样验证了这个结论.DSP 与 BSP 唯一的不同点在于:在每个超级计算步内,DSP 执行Δ步局部计算($\Delta\geqslant 1$);而 BSP 仅执行 1 步局部计算.引入更多局部计算的目的在于进一步挖掘和利用蕴含在数据分区内的局部性,进而加速局部收敛、减少迭代轮数和通信开销.在通用计算集群中,对于计算量少但迭代步数多的算法,迭代轮数的减少直接意味着收敛时间的减少.原因是这种类型的算法将大部分执行时间都消耗在全局同步上.在表 1 中,我们将 DSP 与几种常见的并行计算模型进行了比较:(i) 同步方式——同步、异步;(ii) 迭代次数;(iii) 与 BSP 每步结果是否能对应上,结果是否收敛.

Table 1 Comparison between common parallel models and DSP

表 1 常见的并行计算模型和 DSP 的比较

名字	同步方式	迭代	轨迹
BSP	同步	1 遍	精确
PiT	同步	多遍	收敛不精确
AiA	异步	1 遍	不精确
SP	异步	1 遍	精确且收敛
DSP	同步	1 遍	收敛不精确

本文有如下贡献.

1) DSP 并行计算模型:本文提出了一种新的并行计算模型,可用于加速一大类并行迭代算法.
2) 并行计算模型的形式化表示方法:本文提出了一种形式化表示方法,可以很好地表示各种并行计算模型及其迭代过程.
3) 正确性和适用性证明:利用贡献 2)中给出的形式化方法,推导并证明了 DSP 的适用性及收敛条件.
4) 编程指导:为指导用户将 BSP 程序改写或直接构造 DSP 程序,给出了具体的实施步骤.

本文首先介绍若干相关工作,包括 BSP、参数服务器等模型.为了对 DSP 建立一个初步映像,第 2 节列举 DSP 的几个应用场景.第 3 节描述并行模型的形式化表示方法,并使用该种方法表示 BSP 和 DSP 的迭代过程,推导和证明 BSP 和 DSP 的等价收敛条件.第 4 节使用真实数据集和实际应用验证并评估 DSP 模型的加速性能.第 5 节是总结和工作展望.

1 相关工作

1.1 大同步并行模型(BSP)

Leslie Valiant 于 1990 年在牛津大学提出了著名的 BSP 并行计算模型.该模型主要用于指导同步并行算法或程序的设计.一个典型的 BSP 算法或程序由如下 3 部分组成:局部计算(local computation);通信(communication);阻塞同步(barrier synchronization).

BSP 算法的一个超级计算步(superstep)可以描述为图 1 所示的过程.

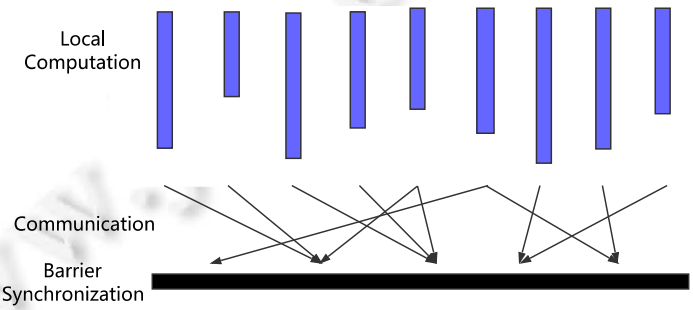


Fig.1 BSP pattern[14]

图 1 BSP 算法执行流程[14]

与 BSP 相比,DSP 简单地将单步局部计算变为Δ步局部计算($\Delta \geq 1$),新增加的局部计算步只是简单地重复执行之前的计算.DSP 算法的一个超级计算步可以描述为图 2 所示的过程(DSP 将局部计算步数变为Δ步.局部计算时,只更新本地变量,不进行节点间通信).同时,我们将额外增加的($\Delta \geq 1$)步局部计算称为投机计算步(speculative computation step,简称 SCStep),其定义如下.

投机计算步(SCStep). BSP 局部计算的简单重复,期间只做局部数据更新,计算节点间无数据通信发生.

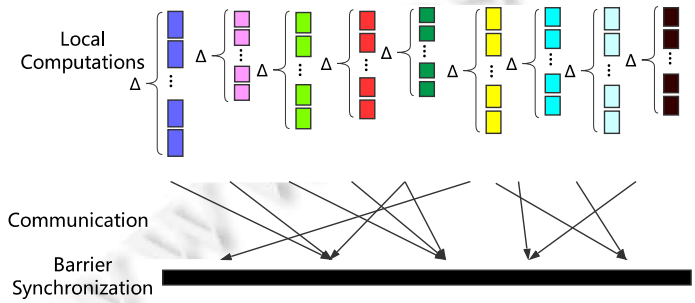


Fig.2 DSP executing pattern

图 2 DSP 算法执行模型

1.2 参数服务器

为了解决大规模分布式机器学习中数以百万计参数频繁更新的问题,Google 在 2012 年提出了参数服务器的方案[15].参数服务器允许各个模型副本在一个很小的时间间隔内,异步地更新和下载中心参数服务器中最新

的参数.这样一来,各个模型副本就可以减少使用参数服务器时互斥等待的时间.虽然在允许的时间间隔内,各个模型副本从参数服务器上获取的数据并不一致,但算法最后却能收敛.Google 的论文[15]中展现了参数服务器惊人的加速潜能,同时也表达了对加速原理的疑惑.进一步工作[15−19]分别将参数服务器用于加速不同的应用,并分别给出了收敛性证明.然而到目前为止,还没有一个正确性的一般性证明.与参数服务器不同的是,DSP 算法除了支持一大类机器学习的优化算法以外,从根本上是针对并行迭代计算提出的,因而几乎可以适用于所有的并行迭代计算.针对其适用性和正确性,本文给出了一个一般性证明及适用约束条件.

另外,参数服务器的相关工作并没有解释其加速原理,本文中对 DSP 加速原理的解释同样适用于解释参数服务器加速的原理.

1.3 其他工作

(1) KLA:*K* 层异步算法(*K*-level asynchronous,简称 KLA)[20].DSP 与 KLA 的不同主要体现在以下几点.

- DSP 仅仅在大同步时才进行节点间通信,而 KLA 在局部计算时也会进行节点间通信.因而,DSP 比 KLA 更简洁,同时也具备比 KLA 更好的扩展性和更广的适用范围.实验中我们发现,通信的开销不仅仅与通信的次数相关,而且还受到消息长度的极大影响.
- DSP 的应用不限于 KLA 所局限的图计算领域.
- DSP 不是 KLA 的特例,KLA 也不是 DSP 的特例.

(2) 多步并行最短路算法(*Δ*-stepping: A parallelizable shortest path algorithm)[21,22]:通过维护一个待选顶点的列表,每次向前尝试性地进行*Δ*步迭代.DSP 并不需要显示地维护这样一个列表和待选数据集,只需在数据分区中简单重复地执行相同的操作并更新局部数据,就可以达到与该算法相同的效果.DSP 用于加速单源最短路算法的示意图如图 3 所示.虚线界定的网格代表一个计算节点,每个计算节点负责计算图中部分顶点的最短路.由于 DSP 每个超级计算步可以执行*Δ*次局部计算和局部更新,而 BSP 却只能执行 1 次,所以着相同颜色的顶点可以在同一轮大同步求得最短路.求得图 3(a)中所有顶点的最短路仅需 5 轮全局大同步,求得图 3(b)中所有顶点的最短路则需要 13 轮全局大同步.

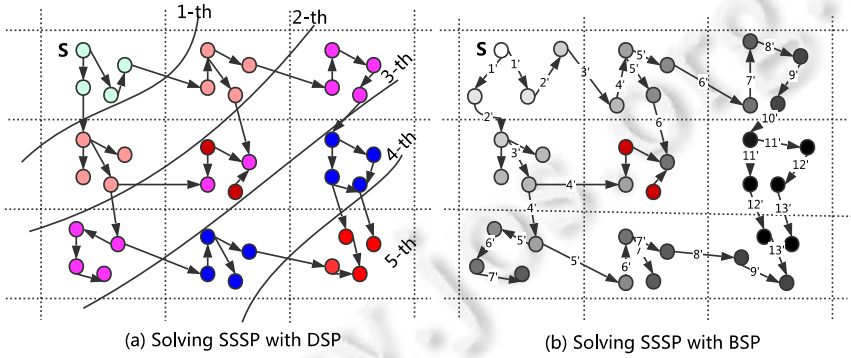


Fig.3 Solving procedures of single source shortest path (SSSP) from all other vertices to vertex *S* with DSP and BSP model respectively

图 3 分别使用 DSP 和 BSP 模型求解图中所有顶点到 *S* 点的单源最短路

2 应用举例

正式描述模型之前,为对 DSP 建立一个初步的映像,首先介绍几个简单的例子.

- 图并行算法

将 DSP 用于加速图并行算法时,可以很好地解释 DSP 的工作流程及原理.如图 3 所示,将 DSP 用于加速单源最短路(SSSP)算法,图中每个虚线界定的网格表示一个计算节点,每个计算节点负责图中一部分图顶点最短

路的计算.BSP 模式下,每个超级计算步每次计算仅向前推进 1 步,而 DSP 每次却可以前进多步,甚至将最短路在一个超级计算步就传递给相邻的下一个网格.对其他图算法如 PageRank 等,多步局部计算同样可以通过更充分利用数据分区内的局部性加速子图内的局部收敛,进而促进算法的全局收敛.

- 求解线性方程组

在某些工程和科研应用中,部分线性方程组的系数矩阵如此稀疏,以致于可以将未知数向量 X 分为多段,分别进行独立求解.并行求解这类型方程组实际上只需很少的通信,即可达到较高精度的收敛.因此,DSP 模型十分适合用于加速稀疏线性方程组的求解.

- 机器学习

机器学习的训练数据通常由数 10 亿级的高维向量组成,同时,学习模型的参数也由数百甚至上千维的向量组成.然而,由于数据和模型的稀疏特性,分布于不同计算节点上的参数之间的严格同步并非必备条件,甚至部分节点的数据之间并不存在相互依赖,因此,分布式优化算法十分适合采用 DSP 进行加速.

3 模型及其收敛性保证

DSP 模型的目标是在不影响算法收敛的前提下,通过减少算法的迭代轮数来减少算法在通信上的开销,进而加速算法的收敛.为了实现这个目标,我们发现,不同于稠密数据集上的迭代计算,稀疏数据集上的迭代计算拥有更小的计算通信比:$T_{computation}/T_{communication}$,即迭代时间主要耗费在通信上,而非计算.因此,通信开销的减少可以有效地加速并行算法的收敛.此外我们还发现,仅仅 1 步局部计算很难充分挖掘和利用数据分区内的局部性.对于一大类图并行算法,1 次局部计算对应着 1 次局部值传递,那么多步局部计算意味着多步局部值传递.然而,投机计算步并非越多越好,因为投机计算步是以增加计算量为代价,当集群节点间初始负载不均时,太多的投机计算步甚至会加剧这种不均衡.

分析表明,投机计算步具备如下两个特征:(i) 尝试在数据分区内通过更充分地挖掘和利用空间局部性,实现空间局部性最优;(ii) 尝试在一个超级计算步内通过更多的执行投机计算步实现超级计算步内的时间局部性最优.简而言之,DSP 通过执行合理数量的投机计算步实现空间和时间上的局部最优.如果这里的空间和时间局部最优正好发生在某些适合的算法或作用于稀疏数据集上,那么它极有可能转化为最终和全局的最优.

此外,DSP 还具备一些其他优点.如:(i) 当应用于值传递算法时,可同时适用于稠密和稀疏数据集;(ii) 当用于加速雅各比迭代时,展现出了类似于超松弛(successive over-relaxation,简称 SOR)[21]的加速效果,即同时减少了局部计算步和全局同步轮数.

3.1 形式化表示

为了统一表示和比较不同的并行计算模型,本文提出了一种类似于“矩阵乘法”的转换:$X_{k+1}=X_k\otimes F_{n\times m}$,其中,$X_k,X_{k+1}$ 分别为第 k 轮迭代的输入和输出变量,$F_{n\times m}$ 为迭代计算进行的操作.不同于代数中的矩阵乘法,$F_{n\times m}$ 中的每个元素 $F_{i,j}$ 表示输入向量 X_k 的分量 x_i 与 x_j 之间进行的计算,其结果返回给 x_j.得到 x_j 所有依赖的分量与 x_j 运算后的结果,再将这些结果进行聚合,聚合的结果将作为 x_j 本轮计算的输出.直观上理解,x_j 下一轮的新值实质上是将它依赖的每个分量对它产生的影响或改变进行聚合的结果.类似的表示方法也在文献[23,24]中出现,不同的是,我们进一步推导出了输入变量 X 随迭代进行的演变形式.

为使下文更简洁地表达和推导,首先定义如下变量和符号.

1) X_0:迭代计算的初始输入变量.

其向量表示为$(x_0,x_1,...,x_n)$.x_i 可以表示各种类型的数据,如图计算中顶点信息、线性方程组中的未知数等.

2) X_k:迭代计算的第 k 轮输出.

并行化时,变量 X_k 被切分为多段并分布在不同的计算节点上进行计算,$X_k^{\{p,q\}}$ 表示某个计算节点仅负责计算从 x_p 到 x_q 一段的分量.

3) F:关系矩阵.

$F_{i,j}$ 定义 x_i 与 x_j 之间的运算,函数形式为 $F_{i,j}(x_i,x_j)$,简写为 $F_{i,j}(x_i)$.其运算结果返回给 x_j 供其进一步与其他所依赖的变量计算产生的结果进行聚合(具体实现时,$F_{i,j}$ 通常被表达为一个数学公式、函数、过程或方法).

$F^{(p,q)}$表示对输入变量进行一次部分转换.$F^{(p,q)}$仅计算和更新 x_p 到 x_q 之间的变量.其定义如下.

$$\begin{bmatrix}
1 & 0 & \cdots & 0 & F_{0,p} & \cdots & F_{0,q} & 0 & \cdots & 0 \\
0 & 1 & \cdots & 0 & F_{1,p} & \cdots & F_{1,q} & 0 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 1 & F_{p-1,p} & \cdots & F_{p-1,q} & 0 & \cdots & 0 \\
0 & 0 & \cdots & 0 & F_{p,p} & \cdots & F_{p,q} & 0 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 0 & F_{q,p} & \cdots & F_{q,q} & 0 & \cdots & 0 \\
0 & 0 & \cdots & 0 & F_{q+1,p} & \cdots & F_{q+1,q} & 1 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 0 & F_{n,p} & \cdots & F_{n,q} & 0 & \cdots & 1
\end{bmatrix}.$$

使用向量与矩阵相乘的规则,X 的分量被投射到 $F_{i,j}$ 上进行运算,最终只有 x_p 到 x_q 之间的变量进行了运算和更新,其余分量保持不变.

4) ⊎:聚合操作符.

此运算符将$\{F_{i,j}(x_i)|i=0,1,2,...,n\}$中所有运算结果进行聚合,聚合得到的值将作为 x_j 本轮迭代计算的输出.公式为

$$\begin{aligned}
x_i' &= (x_0,x_1,...,x_n)\cdot(F_{i,0},F_{i,1},...,F_{i,n})^T \\
&= \uplus[F_{i,0}(x_i,x_0),F_{i,1}(x_i,x_1),...,F_{i,n}(x_i,x_n)] \\
&= \uplus_{j=0}^{n} F_{j,i}(x_j,x_i), \text{简称} \uplus_{j=0}^{n} F_{j,i}(x_j).
\end{aligned}$$

常见的聚合操作有 min(·),max(·),average(·),Σ,Π等.

5) ⊗:转换运算符.

该运算符将关系矩阵在输入变量上作用 1 次.

表 2 中列举了几种常见算法的关系函数和聚合函数.

Table 2 Common algoriths and corresponding relation/aggregation functions

表 2 常见算法及其对应的关系函数和聚合函数

算法	$F_{i,j}$	⊎	注解
PageRank	$\frac{x_i}{L(x_i)}$	$\sum_{x_i\in N(x_j)}\frac{x_i}{L(x_i)}$	$L(x_i)$:x_i 所指向的网页数目; $N(x_j)$:所有指 x_j 的页面的集合
SSSP	$X_i+w_{i,j}$	$\min_{x_i\in N(x_j)}(x_i+w_{i,j})$	$w_{i,j}$:x_i 指向 x_j 的边的权重; $N(x_i)$:所有指向 x_j 的页面的集合
Jacobi	$\frac{a_{i,j}}{a_{i,i}}x_i$	$\frac{b_i}{a_{ii}}-\frac{1}{a_{ii}}\sum_{i\neq j}a_{ij}x_i$	$N(x_i)$:所有指 x_j 的页面的集合

使用如上定义的变量和符号,BSP 模型的迭代过程可推导如下.

$$
\begin{aligned}
&X_0=(x_0,x_1,...,x_n)\\
&X_1=X_0\otimes F\\
&\quad=(x_0,x_1,...,x_n)\otimes\begin{pmatrix}F_{0,0} & F_{0,1} & \cdots & F_{0,m}\\ F_{1,0} & F_{1,1} & \cdots & F_{1,m}\\ \vdots & \vdots & \ddots & \vdots\\ F_{n,0} & F_{n,1} & \ddots & F_{n,m}\end{pmatrix}\\
&\quad=\left(\biguplus_{i=0}^{n}F_{i,0}(x_i),\biguplus_{i=0}^{n}F_{i,1}(x_i),...,\biguplus_{i=0}^{n}F_{i,m}(x_i)\right)\\
&\quad=(h(X_0),h(X_0),...,h(X_0)),\text{其中},h(X)=\biguplus_{j=0}^{n}F_{j,i}(x_j)\text{表示一轮向量乘和聚合操作}\\
&X_2=X_1\otimes F\\
&\quad=(h(X_1),h(X_1),...,h(X_1))\\
&\quad=(h(h(X_0)),h(h(X_0)),...,h(h(X_0)))\\
&\quad=(h^2(X_0),h^2(X_0),...,h^2(X_0))\\
&X_2=(h(X_2),h(X_2),...,h(X_2))\\
&\quad=(h(h(X_1)),h(h(X_1)),...,h(h(X_1)))\\
&\quad=(h^3(X_0),h^3(X_0),...,h^3(X_0))\\
&\quad\vdots\\
&X_k=(h^k(X_0),h^k(X_0),...,h^k(X_0))
\end{aligned}
\tag{1}
$$

通过如上推导,我们发现,k 轮 BSP 迭代可由 k 次关系矩阵的转换来表示.

类似地,对 X_0 进行 l 轮 DSP 迭代,可得到对应的 $X_{l\Delta}$(每轮 DSP 大同步对应进行Δ步局部计算,详细推导过程见 https://github.com/wdfnst/DSP_Proof/blob/master/dsp_proof.pdf):

$$
X_{l\Delta}=(h^l(g^{\Delta-1}(\alpha_0,\beta_0)),...,h^l(g^{\Delta-1}(\alpha_n,\beta_n))) \tag{2}
$$

其中,

$$
g(\alpha_p,\beta_p)=\biguplus\left(\biguplus_{i\in(p,q)}F_{i,j}(\alpha_p),\beta_p\right),
$$

$$
\alpha_p=\biguplus_{i=0}^{n}F_{i,p}(x_{t_0,i}),
$$

$$
\beta_p=\biguplus_{i\notin(p,q)}F_{i,p}(x_{t_0,i}),
$$

$$
p=0,1,2,...,n.
$$

直观上理解,公式(2)中的参数解释如下.

- $g(\alpha_p,\beta_p)$:将 BSP 中一次局部计算变为两次局部计算后产生的新算法.
- α_p:聚合 x_p 所有依赖的变量对其作用的结果.
- β_p:聚合 x_p 所依赖的且位于不同计算节点上的变量对其作用的结果.β_p 也可视为 x_p 对其他分区数据的依赖程度.
- $\gamma_p(=\alpha_p-\beta_p)$:聚合 x_p 所依赖的且位于相同计算节点上的变量对其作用的结果.γ_p 也可视为 x_p 对分区内数据的依赖程度.

要使用 DSP 加速 BSP,即在迭代中用公式(2)表示的计算替换公式(1)表示的计算,首先需要证明它们可以收敛到相同的最终状态.然而公式(2)并不能保证收敛到与公式(1)相同的最终状态.但对于凸优化问题或只要求收敛到局部最优点的算法,达到收敛就足够了.第 3.2 节将使用数学归纳法证明 DSP 的收敛条件.

3.2 DSP模型的收敛性保证

包括参数服务器和 DSP 模型在内的许多模型或算法[22−25]都使用了投机计算的思想,然而,其中仅部分模型或算法给出了正确性解释或适用条件说明.在没有正确性保证的前提下,用户在选用这些方法时始终保持着谨慎的态度.本节将给出 BSP 和 DSP 的关系,并推导出 DSP 的收敛条件.

定理 1. 如果算法在 BSP 模式下收敛,那么,当且仅当满足如下条件时,算法在 DSP 模式下也收敛:

$$\text{算法 } g(\alpha_p,\beta_p)\text{在 BSP 模型下收敛} \tag{3}$$

直观上理解,定理 1 描述了在 BSP 模式下收敛的算法,若增加一次局部计算后得到的新算法仍然收敛,那么增加任意数量的局部计算得到的新算法都将收敛.

证明:当Δ=1 时,DSP 模型退化为 BSP 模型.故Δ=1 时,结论成立.

假设当$\Delta=k(k\geqslant 1)$时,结论仍然成立,即

$$h^l(g^{k-1}(\alpha_p,\beta_p)) \tag{4}$$

收敛.

那么当$\Delta=k+1$ 时,得到如下左式,并变形为右式:

$$h^l(g^k(\alpha_p,\beta_p))=h^l(g^{k-1}(g(\alpha_p,\beta_p),\beta_p)) \tag{5}$$

对比公式(4)、公式(5),在 BSP 模式下,因为α_p 收敛,所以公式(4)收敛.从而,公式(5)收敛的条件是:

$$\text{算法 } g(\alpha_p,\beta_p)\text{在 BSP 模型下收敛} \tag{6}$$

在满足条件(6)时,定理 1 对$\Delta=k+1$ 成立.

由数学归纳法原理可知:在满足条件(6)时,定理 1 对所有$\Delta\in\mathbf{N}^+$成立. □

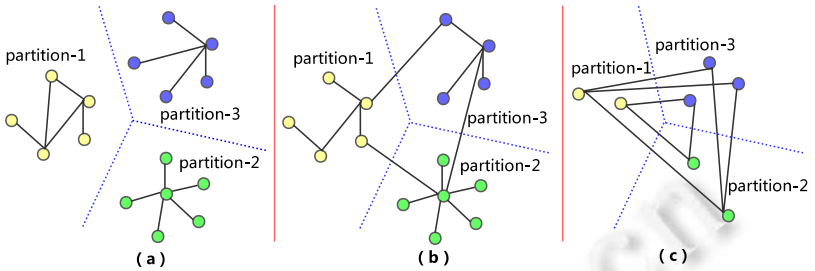
3.3 DSP加速因素分析

由公式 $g^{\Delta 1}(\alpha_p,\beta_p)$可知,DSP 算法的收敛速度主要依赖于 3 个因素:Δ,α_p 和β_p(其中,p=0,1,2,…,n),并存在如下关系.

- 当β_p=0 时(可理解为 x_p 的收敛不依赖任何位于其他计算节点上的变量),x_p 在不需要任何全局数据同步的前提下即可收敛.这时,额外的(Δ–1)步投机计算可以获得非常显著的加速,以至于当Δ充分大时,x_p 可以在不需要任何全局同步的情况下直接收敛.图 4(a)示例了这种情况,每个数据分区被分配到不同的计算节点,彼此之间不存在任何依赖关系,这个图上的并行迭代计算可以在不需要任何全局数据同步的情况下收敛.
- 当γ_p=0 时(可理解为 x_p 的收敛完全依赖于位于其他节点上的变量),因为 x_p 所依赖的变量在Δ步局部计算中并不会更新,所以Δ次局部计算产生的新值也不会有任何变化.图 4(c)示例了这种情况,图中每个分区中的顶点所依赖的变量全部位于其他计算节点内,这种情况下,额外的(Δ–1)次局部计算不会产生任何加速效果.
- 当γ_p>0 时(即 x_p 所依赖的变量部分位于和自己相同的节点内),额外的(Δ–1)步计算会促进 x_p 更快地收敛,且加速效果与γ_p 成正比.
- 当β_p>0 时(即 x_p 所依赖的变量部分位于其他节点),第 1 次局部计算之后,β_p 并没有及时从其他节点获取最新值进行更新,剩下的(Δ–1)步局部计算仍然使用上次全局同步的变量计算产生的β_p.过期的β_p 对 x_p 的收敛其起副作用,DSP 的加速效果与β_p 成反比.图 4(b)示例了这种情况,每个分区内的顶点所依赖的变量既有来自本节点的,也有来自其他节点上的.

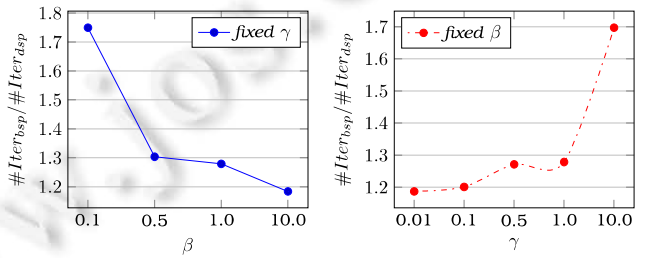
某种意义上,γ_p 和β_p 可分别用数据分区内和数据分区间的依赖关系密度来解释.若能通过适当的数据划分增加分区内依赖关系密度(即增大γ_p),同时减小分区间依赖关系密度(即减小β_p),那么算法的收敛极有可能得到加速.然而,完美的数据切分(如图划分)通常都是 NP 难问题.尽管增大γ_p 比较困难,足够小的β_p 却十分普遍.因为稀疏数据集划分之后得到的β_p 通常都会非常小,这就为我们使用 DSP 加速迭代计算创造了条件.

为了验证β_p,γ_p 与加速效果之间的关系,我们使用 PageRank 算法在随机图上进行了两组实验:(i) 固定子图内的连接度,变化子图之间的连接度;(ii) 固定子图间的连接度,变化子图内部的连接度.实验结果如图 5 所示,图 5(a)显示,加速比$\#Iteration_{bsp}/\#Iteration_{dsp}$ 随着增加的β而下降,即子图间连接增加后,DSP 的加速性能下降了.图 5(b)显示,加速比$\#Iteration_{bsp}/\#Iteration_{dsp}$ 随着增加的γ而上升,即子图内连接增加后,DSP 的加速性能上升了.这一结果进一步验证了我们上面的分析.



(a) γ_p=0:图中每个顶点所依赖的全部变量和它位于同一个计算节点上
(b) β_p>0:图中某些顶点依赖的变量部分位于和它不同的计算节点上
(c) β_p=0:图中每个顶点所依赖的全部变量都和它位于不同的计算节点上

Fig.4
图 4



(a) 固定γ,变化β (b) 固定β,变化γ

Fig.5 Acceleration of DSP on PageRank
图 5 DSP 对 PageRank 的加速

3.4 DSP并行算法的构造

对比算法 1、算法 2,将 BSP 算法改造为 DSP 算法或直接构造 DSP 算法只需在进行全局通信前增加一个条件测试(测试是否执行了指定步数的局部计算).构造 DSP 程序时,可按如下几步进行.

1) 若 BSP 算法收敛,则检验拥有两步局部计算的新算法是否收敛,若收敛则转至第 2)步.
2) 筛选合适的参数Δ.
3) 调整代码.增加条件测试,使得每执行 1 次 *DataExchange*(·)对应执行Δ次 *Computing*(·).

算法 1. BSP 并行算法模板.

```
1:  procedure BSP_Algo(G,v0)
2:      iter_counter←0
3:      while True do
4:          computing(·)
5:          DataExchange(·)
6:          if is_convergent(·) then
7:              break
8:          iter_counter++
```

算法 2. DSP 并行算法模板.

```
1:  procedure BSP_Algo(G,v0)
2:      iter_counter←0
3:      while True do
```

```
4:      computing(·)
5:      if iter_count%Delta==0 then
6:          DataExchange(·)
7:          if is_convergent(·) then
8:              break
9:      iter_counter++
```

对于各种算法,并没有普遍适用的Δ值.根据第 3.3 节中的分析可知:当数据集比较稀疏或采用较好的数据划分算法时,可以选用适当大的Δ;反之,适当小的Δ加速效果可能更好.除此之外,非线性是导致变量变化快慢的主要因素之一.当算法非线性较强时,重用剧烈变化的变量值会引入较大的误差,致使收敛速度减慢.所以,非线性强的算法适合选用较小的Δ,反之适于选用较大的Δ.

4 实验与评估

实验中用到的设备主要包括 20 台高性能服务器及连接它们的高速以太网(40GB/s 或换算为 3.2GB/s)和 InfiniBand 网络(6.8GB/s).每台服务器由两片 Intel Xeon E5520 处理器(4 核×2.27GHz)和 48G 内存以及其他外设组成.除 GMRES 和 SGD 外,其他算法都采用 C++和 MPI 通信接口实现.默认情况下,MPI 优先使用 InfiniBand 进行通信,所以这里实现的算法实际上都优先使用 InfiniBand 进行通信.因为 DSP 加速的原理是通过减少迭代轮进而减少通信开销实现的,而在高速网络中,由于通信延迟低,DSP 的加速性能并没有得到充分的展示.在由 TCP/IP 网络连接的通用计算集群中,DSP 的加速性能会比下面列出的结果更加卓越.

4.1 图算法应用

1) PageRank.PageRank 算法通过赋予 web 网络中每个页面一个权重,并以此来衡量每个页面的相对重要程度.其计算既可通过代数方法求解也可通过迭代法求解,迭代公式为

$$PR(p_i)=\frac{1-c}{N}+\sum_{p_j\in N(P_i)}\frac{PR(P_j)}{L(p_j)},$$

其中,$p_1,p_2,...,p_n$表示需要计算 PageRank 值的网页,$N(p_i)$表示所有连接指向 p_i 的网页集合,$L(p_j)$表示网页 p_j 的出度,N 表示所有网页的数目.常数$(1-c)/N$表示网页浏览者随机打开一个网页的概率,这个机制用来解决和防止终止点问题和陷阱问题.

我们收集了 3 份真实 Web 图数据,并对其基本信息进行了统计.如表 3 所示,指标包括顶点数、边数、平均出度、最大出度和最小出度.从其平均出度来看,这些 Web 图并不稀疏.并且 Web 图的入度分布一般都高度畸形,即其顶点的入度悬殊非常大.

Table 3 Statistics of real world Web graphs

表 3 真实 Web 图及其基本统计信息

图名称	顶点数	边数	平均出度	最大出度	最小出度
BerkStan	685 230	7 600 595	11.092	249	0
NotreDame	325 729	1 497 134	4.596 23	3 445	0
Stanford	281 903	2 312 497	8.203 11	255	0

将表 3 中的 Web 图按顶点近似等分划分,并分发到不同的计算节点.在每个计算节点上每执行Δ步局部计算再进行一次全局大同步.图 5(a)~图 5(c)展示了 DSP 的加速效果.数字显示,DSP 可以显著减少 BSP 的迭代轮数,进而减少算法的收敛时间.在这组实验中,我们采用了 Metis 工具包[26]进行图划分(Metis 工具包中实现的算法可以在保证子图间顶点数量均衡的前提下,最大限度地减少子图间割边的数量,从而降低子图间的依赖性;而随机图划分算法仅能保证子图间顶点数量的均衡,不能减少子图间割边的数量,也不能降低子图间依赖的程度).与此同时,我们还在随机划分的子图上进行了相同的测试,结果显示:除Δ=2 有少许加速外,DSP 算法的收敛速度甚至还不如 BSP.这个结果也验证了我们在第 3.3 节中的分析,即 DSP 的加速效果十分依赖于β和γ.

图 6 使用真实 Web 图和路图,测试 DSP 对 PageRank 和 SSSP 算法的加速性能.图 6(a)~图 6(c)展示在使用 Metis 工具包切分的子图上,DSP 对 PageRank 加速性能;图 6(d)~图 6(f)展示在使用 Metis 工具包切分的子图上,DSP 对 SSSP 的加速性能;图 6(h)~图 6(g)展示在使用随机划分的子图上,DSP 对 SSSP 的加速性能.PageRank 算法的收敛精度设为 10^{-10}.

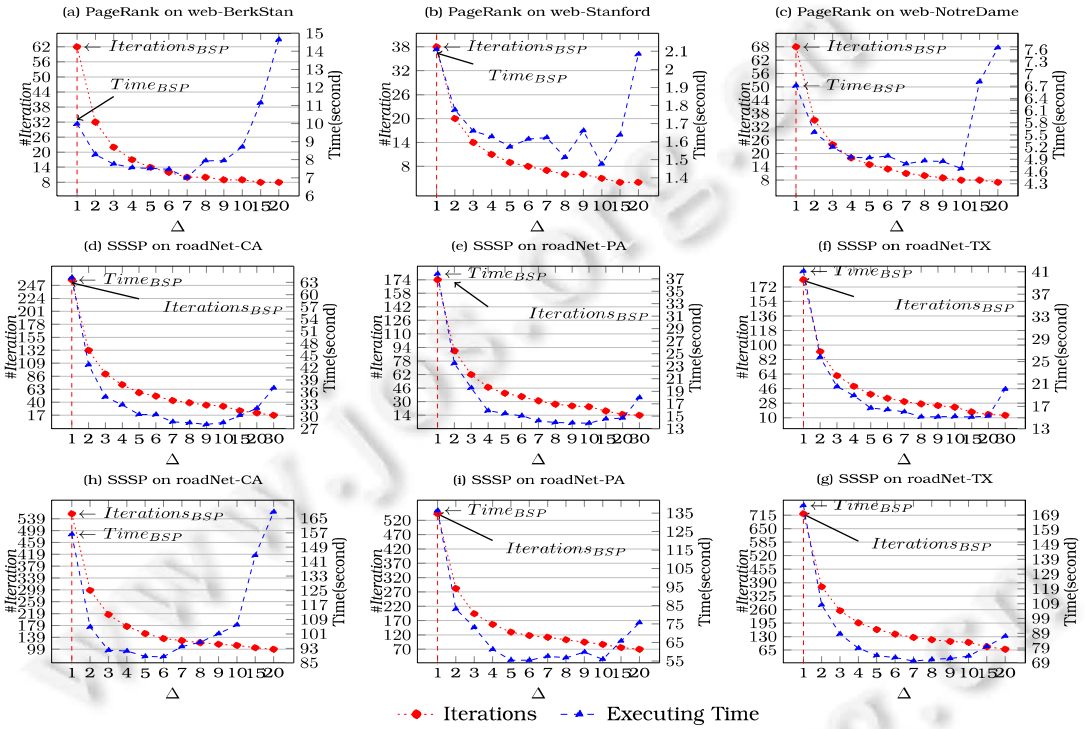


Fig.6 Performance comparison between DSP and BSP working on PageRank and SSSP with the real Web graph

图 6 使用真实 Web 图和路图,测试 DSP 对 PageRank 和 SSSP 算法的加速性能

2) SSSP.给定一个有向赋权图 $G(V,E)$,SSSP 算法用来寻找图中所有点到指定源点的最短距离.

为了方便说明,图 3 中示例了一个简单的单源最短路求解的例子,以说明 DSP 加速单源最短路这类图算法的原理.图中每个虚线界定的网格表示一个计算节点,分别负责计算图中部分顶点的最短路.图 3(a)、图 3(b)中随着着色的变化(或加深),分别展示了 DSP 和 BSP 迭代的过程.由于 DSP 在一个超级计算步中可执行多次局部计算,从而可以将最短路向前传递多步,而 BSP 每次只能前进 1 步.图 3(a)中使用 DSP 加速,整个计算过程仅需 5 次全局大同步即可收敛,而图 3(b)使用 BSP 迭代,则需要 13 次大同步.在这个例子中,DSP 的加速比为 2.6 倍.

为了验证 DSP 在实际路图中的加速性能,采用美国若干州的路图进行了实验.同样,为了验证第 3.3 节中的分析,如下两种图划方法被用于图划分:(i) 随机划分;(ii) Metis 划分.路图的的基本信息及统计信息见表 4,可以看出,路图是比 Web 图稀疏得多的自然图.

Table 4 Statistics of real world road networks

表 4 真实路图及统计信息信息

图名称	顶点数	边数	平均出度	最大出度	最小出度
roadNet-CA	1 965 206	2 766 607	2.82	12	1
roadNet-PA	1 088 092	1 541 898	2.83	9	1
roadNet-TX	1 379 917	1 921 660	2.79	12	1

如图 6 所示,图 6(d)~图 6(f)、图 6(g)~图 6(i)分别展示了在采用不同图划分方法得到的子图上进行实验的

结果.数字显示,算法在 Metis 划分的子图上的性能是在随机划分的子图上的性能的数倍.这一结果进一步验证了我们在第 3.3 节中的分析.同时,仅采用随机分图,DSP 对 BSP 的加速也可高达 10 倍.

4.2 线性方程组求解

设 $AX=b$ 表示一个由 n 个线性方程组成的线性方程组,其中,

$$A=\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix},\ x=\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix},\ b=\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}.$$

当系数矩阵 A 为低阶非稠密矩阵时,使用高斯主元消元法可以进行非常高效的求解.但当系数矩阵为稀疏矩阵时,迭代法则更加高效,它能更充分地利用系数矩阵中出现的大量零元,进而避免大量不必要的计算.Jacobi 迭代法求线性方程组的迭代公式为

$$x_i^{(k+1)}=\frac{1}{a_{ii}}\left(b_i-\sum_{j\neq i}a_{ij}x_j^{(k)}\right),\ i=1,2,...,n.$$

超松弛法(successive over-relaxation,简称 SOR)[21]是 Jacobi 迭代法的一种改进算法,可以实现比 Jacobi 迭代更快的收敛.其迭代公式为

$$x_i^{(k+1)}=(1-\omega)x_i^{(k)}-\frac{\omega}{a_{ii}}\left(b_i-\sum_{j<i}a_{ij}x_j^{(k+1)}-\sum_{j>i}a_{ij}x_j^{(k)}\right),\ i=1,2,...,n \tag{7}$$

其中,常数 $\omega>1$ 称为松弛因子.

为了比较 DSP 和 SOR 加速 Jacobi 迭代的原理,我们将执行一步局部计算对 x_l 所做的操作进行如下表示和变形:首先,使用 x_i^{k+1},x_l^{k+1} 分别表示 x_i 和 x_l 在第(k+1)步局部计算时得到的最新值.

$$x_i^{(k+1)}=\frac{1}{a_{ii}}\left(b_i-\sum_{j<i}a_{ij}x_j^{(k+1)}-a_{il}x_l^{(k)}-\sum_{j>i,\,j\neq l}a_{ij}x_j^{(k)}\right),$$

$$x_l^{(k+1)}=\frac{1}{a_{ll}}\left(b_l-a_{li}x_i^{(k+1)}-\sum_{j<l,\,l\neq i}a_{lj}x_j^{(k+1)}-\sum_{j>l}a_{lj}x_j^{(k)}\right),$$

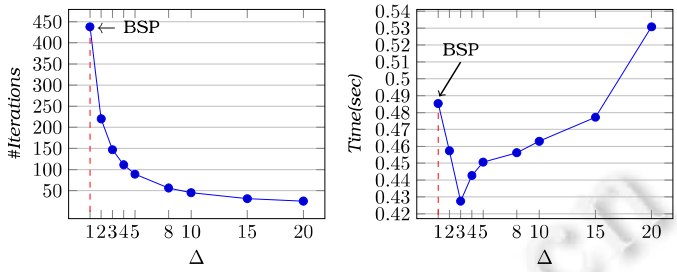
其中,$i<l$.将 x_i^{k+1} 带入 x_l^{k+1},得到:

$$\left.\begin{aligned} x_l^{(k+1)}&=\frac{1}{a_{ll}}\left(b_l-\frac{a_{li}}{a_{ii}}\left(b_i-\sum_{j<i}a_{ij}x_j^{(k+1)}-a_{il}x_l^{(k)}-\sum_{j>i,\,j\neq l}a_{ij}x_j^{(k)}\right)-\sum_{j<l,\,l\neq i}a_{lj}x_j^{(k+1)}-\sum_{j>l}a_{lj}x_j^{(k)}\right) \\ &=\frac{a_{li}a_{il}}{a_{ll}a_{ii}}a_{il}x_l^{(k)}+\frac{1}{a_{ll}}\left(b_l-\frac{a_{li}}{a_{ii}}\left(b_i-\sum_{j<i}a_{ij}x_j^{(k+1)}-\sum_{j>i,\,j\neq l}a_{ij}x_j^{(k)}\right)-\sum_{j<l,\,l\neq i}a_{lj}x_j^{(k+1)}-\sum_{j>l}a_{lj}x_j^{(k)}\right) \end{aligned}\right\} \tag{8}$$

比较公式(7)和公式(8)不难发现,DSP 的迭代递推公式和 SOR 的递推公式有着类似的形式,这进一步验证了实验中的发现,即 DSP 和 SOR 可以同时减少计算量和通信量.

实验采用了由 10 000 个线性方程组成的线性方程组.将未知数向量 X 等分为 20 份,并分发到不同的服务器中.每台服务器每进行 Δ 步局部计算,对应进行一次全局大同步.实验结果如图 7 所示(收敛精度设为 10^{-14}):迭代的步数首先随着 Δ 的增加而减少,随后稳定在一个值周围.收敛时间随着 Δ 的增加,先减少再增加.收敛时间下降后再增加是因为投机计算步并非越多越好,它同时也会增加计算负担,进而抵消减少通信带来的好处.

广义最小残差算法(generalized minimal residual method,简称 GMRES)[27]是一种被广泛应用的线性方程组迭代求解算法,为了评估 DSP 的加速性能和扩展性,我们将其与 GMRES 算法的速度进行了比较.在求解一个由 10 000 个线性方程组成的线性方程组时,GMRES 耗时 0.0373 18s 和 71 轮迭代.DSP 的性能如图 8 所示(收敛精度设为 10^{-14}),当 Δ=100 时,DSP 仅耗时 0.0187 14s 和 2 轮迭代就达到了和 GMRES 同样的收敛精度.这一结果显示,DSP 在加速 Jacobi 迭代法时具有比肩最好算法的性能.



(a) 迭代轮数随Δ变化的趋势 (b) 执行时间随Δ变化的趋势

Fig.7 Acceleration of DSP working on Jacobi iterative method

图 7 DSP 对 Jacobi 迭代算法的加速性能

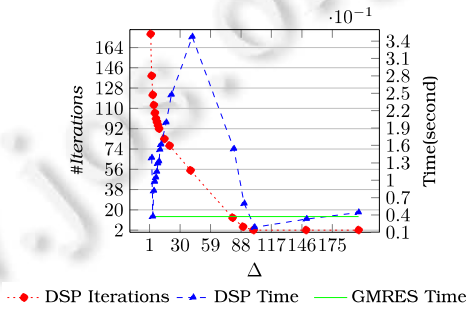


Fig.8 Performance comparison between DSP and GMRES

图 8 DSP 与 GMRES 性能的比较

4.3 加速分布式优化算法

随机梯度下降(stochastic gradient descent,简称 SGD)[28]是梯度下降优化算法的一种近似算法.它通过迭代来求解目标函数的最大或最小值.大量经典机器学习算法的训练过程都可以用 SGD 来进行优化,如 Perceptron[29]、Adaline[30]、*k*-Means[31]和 SVM[32].

为了验证 DSP 模型对 SGD 算法的加速性能,我们进行了一组简单的验证实验.通过分布式的 SGD 算法来训练一个简单的线性分类器.样本集合由维度为 100 的稀疏向量组成,所有的样本点被标记为 4 类,其类别为非零元的位置模 4 运算的结果所决定.

实验结果如图 9 所示(全局同步轮数的对数先随Δ的增加呈指数下降趋势,随后稳定在一个数字周围),全局同步轮数的对数首先随着增加的Δ呈指数级下降趋势,随后稳定在一个值的周围.

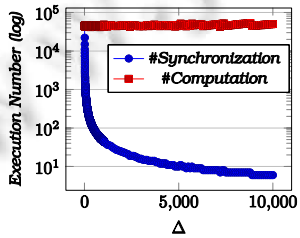


Fig.9 Performance of DSP working on SGD

图 9 DSP 对 SGD 的加速性能

当Δ足够大时,我们发现参数的收敛仅仅需要数轮同步就可以完成.也就是说,在参数收敛的过程中,大部分数据同步都是不必要的.分析其原因,我们认为模型的参数之间的关系也是相对稀疏的,这才导致所有参数之间

并不需要严格的一致也能使模型收敛.

5 总结和工作展望

本文提出了一种并行计算模型,它可以显著地减少迭代计算的轮数.对于瓶颈为通信的算法,迭代轮数的减少直接意味着通信开销的减少,算法的收敛速度也能因此得到相应的提升.除此之外,本文还提出了一种迭代算法的形式化表示方法.使用这种方法,我们表示了 BSP 和 DSP 并行计算模型及其迭代过程,发现 DSP 是一种比 BSP 更一般的计算模型.在此基础之上,本文进一步给出了 DSP 的适用条件和收敛性证明.

为了保证第 3.1 节中推导和证明的一般性,其中出现的符号可用于表示任意关系函数和聚合函数.我们没有提供具体函数的收敛性证明,但通过数值分析的不动点迭代或级数的收敛推导,都可以得到与第 3 节中相同的结论.通过形式化的表示和推导,我们发现,DSP 中增加的局部计算步可以更充分地挖掘和利用数据分区内部的局部性,加速局部收敛,进而促进全局收敛.

实验评估部分展示了若干有趣的发现,比如,加速 Jacobi 迭代时,DSP 模型的执行过程实际上模拟了超松弛的计算;分布式 SGD 的计算过程实际上只需少量的同步即可实现收敛.在通用计算集群上,DSP 可以贡献更好的性能,原因是通用计算集群的通信延迟通常会大得多.

因为 DSP 加速的原理是“计算换通信”,所以过多的局部计算不仅会增加节点的计算负担,而且当迭代计算的初始负载不均衡时,负载不均衡也可能被加剧.

References:

[1] White T. Hadoop: The Definitive Guide. O’Reilly Media, Inc., 2010.

[2] Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I. Spark: Cluster computing with working sets. In: Proc. of the Usenix Conf. on Hot Topics in Cloud Computing. 2010.

[3] Isard M, Budiu M, Yu Y, Birrell A, Fetterly D. Dryad: Distributed data-parallel programs from sequential building blocks. Operating Systems Review, 2007,41(3):59−72.

[4] Brooks BR, Bruccoleri RE, Olafson BD, States DJ, Swaminathan S, Karplus M. Gromacs: Fast, flexible and free. Journal of Computational Chemistry, 2005,26(16):1701−1718.

[5] Heck D, Knapp J, Capdevielle JN, Schatz G, Thouw T. Corsika: A Monte Carlo code to simulate extensive air showers. In: Proc. of the Corsika A Monte Carlo Code to Simulate Extensive Air Showers. 1998.

[6] Kale L. Flexibility and Interoperability in a Parallel Molecular Dynamics Code. ASQ Quality Press, 2008.

[7] Fryxell B, Olson K, Ricker P, Timmes FX, Zingale M, Lamb DQ, Macneice P, Rosner R, Truran JW, Tufo H. Flash: An adaptive mesh hydrodynamics code for modeling astrophysical thermonuclear flashes. Astrophysical Journal Supplement, 2008,131(1): 273−334.

[8] Swarm main page. 2017. http://www.swarm.org/wiki/Swarm_main_page

[9] Valiant LG. A bridging model for parallel computation. Communications of the ACM, 1990,33(8):103−111.

[10] Kollias G, Grama AF, Li Z. Asynchronous Iterative Algorithms. Springer-Verlag, 2011.

[11] Baco L, Bernard S, Maday Y, Turinici G, Zérah G. Parallel-in-time molecular-dynamics simulations. Physical Review E Statistical Nonlinear & Soft Matter Physics, 2002,66(5 Pt 2):Article No.057701.

[12] Culler DE, Karp RM, Patterson D, Sahay A, Santos EE, Schauser KE, Subramonian R, Von Eicken T. Logp: A practical model of parallel computation. Communications of the ACM, 1996,39(11):78−85.

[13] Chen Y, Huang K, Wang B, Li G, Cui X. Samsara parallel: A non-bsp parallel-in-time model. In: Proc. of the ACM Sigplan Symp. on Principles and Practice of Parallel Programming. 2016.

[14] Bulksynchronousparallel. 2017. https://en.wikipedia.org/wiki/Bulk_synchronous_parallel

[15] Smola A, Narayanamurthy S. An architecture for parallel topic models. Proc. of the Vldb Endowment, 2010,3(1):703−710.

[16] Dean J, Corrado GS, Monga R, Chen K, Devin M, Le QV, Mao MZ, Ranzato M, Senior A, Tucker P. Large scale distributed deep networks. In: Proc. of the Advances in Neural Information Processing Systems. 2012. 1223−1231.

[17] Li M, Zhou L, Yang Z, Li A, Xia DA, Fei, Smola AJ. Parameter server for distributed machine learning. In: Proc. of the Big Learning NIPS Workshop. 2013.

[18] Li M. Scaling distributed machine learning with the parameter server. In: Proc. of the Int'l Conf. on Big Data Science and Computing. 2014.

[19] Li M, Andersen DG, Smola A, Yu K. Communication efficient distributed machine learning with the parameter server. In: Proc. of the Int'l Conf. on Neural Information Processing Systems. 2014. 19−27.

[20] Harshvardhan, Fidel A, Amato NM, Rauchwerger L. KLA: A new algorithmic paradigm for parallel graph computations. In: Proc. of the 23rd Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT 2014). 2014. 27−38.

[21] Meyer U, Sanders P. Δ-Stepping: A parallelizable shortest path algorithm. Journal of Algorithms, 2003,49(1):114−152.

[22] Kranjc˘evic´ M, Palossi D, Pintarelli S. Parallel delta-stepping algorithm for shared memory architectures. arXiv preprint arXiv: 1604.02113, 2016.

[23] Yin H, Lee R, Zhang S, Xia CH, Zhang X. Dot: A matrix model for analyzing, optimizing and deploying software for big data analytics in distributed systems. In: Proc. of the ACM Symp. on Cloud Computing. 2011. 1−14.

[24] Feldman J, Muthukrishnan S, Sidiropoulos A, Stein C, Svitkina Z. On distributing symmetric streaming computations. ACM Trans. on Algorithms, 2010,6(4):Article No.66.

[25] Hadjidimos A. Successive overrelaxation (sor) and related methods. Journal of Computational & Applied Mathematics, 2000, 123(1-2):177−199.

[26] Prabhu P, Ramalingam G, Vaswani K. Safe programmable speculative parallelism. In: Proc. of the ACM Sigplan Conf. on Programming Language Design and Implementation. 2010. 50−61.

[27] Burton FW. Speculative computation, parallelism, and functional programming. IEEE Trans. on Computers, 1985,c-34(12): 1190−1193.

[28] Sohn A. Parallel speculative computation of simulated annealing. In: Proc. of the Int'l Conf. on Parallel Processing (ICPP). 1994. 8−11.

[29] Sohn A. Parallel *n*-ary speculative computation of simulated annealing. IEEE Trans. on Parallel & Distributed Systems, 1995,6(10): 997−1005.

[30] Karypis G, Kumar V. Metis: A software package for partitioning unstructured graphs. In: Proc. of the Int'l Cryogenics Monograph. 1998. 121−124.

[31] Saad Y, Schultz MH. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. SIAM Journal on Scientific and Statistical Computing, 1986,7(3):856−869.

[32] Ferguson TS. An inconsistent maximum likelihood estimate. Journal of the American Statistical Association, 1982,77(380): 831−834.

张尉东(1986－),男,四川达州人,博士,主要研究领域为并行与高性能计算.

崔唱(1996－),男,学士,主要研究领域为并行算法设计.