计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.201100051

基于神威平台的 Floyd 并行算法的实现和优化

何亚茹[1]　庞建民[1,2]　徐金龙[2]　朱　雨[2]　陶小涵[2]
1 郑州大学中原网络安全研究院　郑州 450000
2 信息工程大学网络空间安全学院　郑州 450000
(i_heyr@163.com)

摘　要　求解全源最短路径的 Floyd 算法是许多实际应用基础上的关键构建块,由于其时间复杂度较高,串行 Floyd 算法不适用于大规模输入图计算,针对不同平台的并行 Floyd 算法设计可为解决现实问题提供有效帮助。针对 Floyd 算法与国产自主研发处理器匹配滞后的问题,首次提出基于神威平台的 Floyd 并行算法的实现和优化。根据 SW26010 处理器主-从核架构的特点,采用主从加速编程模型进行并行实现,并分析了影响该算法性能的关键因素,通过算法优化、数组划分和双缓冲技术进行优化,逐步提升算法性能。测试结果表明,与主核上串行算法相比,基于神威平台的 Floyd 并行算法在单个 SW26010 处理器上可以获得 106 倍的最高加速。

关键词:SW26010;Floyd 算法;并行计算;数组划分
中图法分类号　TP391

Implementation and Optimization of Floyd Parallel Algorithm Based on Sunway Platform

HE Ya-ru[1],PANG Jian-min[1,2],XU Jin-long[2],ZHU Yu[2] and TAO Xiao-han[2]
1 Zhong Yuan Network Security Research Institute,Zhengzhou University,Zhengzhou 450000,China
2 School of Cyberspace Security,Information Engineering University,Zhengzhou 450000,China

Abstract　The Floyd algorithm for finding shortest paths in a weighted graph is a key building block which is used frequently in a variety of practical applications. However,the Floyd algorithm cannot scale to large-scale graphs due to its time complexity. Its parallel implementations for different architectures are thus proposed and have been proved effective. To address the mismatching between existing ineffective parallel implementation of the Floyd algorithm and domestically designed processors,this paper implements and optimizes the Floyd algorithm targeting the Sunway platform. More specifically,this paper implements the algorithm using the programming model designed for the heterogeneous architecture of the Sunway TaihuLight,and captures the performance bottleneck when executed on the target. This paper next improves the performance of the Floyd algorithm by means of algorithmic optimization,array partitioning and double buffering. The experimental results show that the implementation of the Floyd algorithm on the Sunway platform can achieve the highest speedup of 106X over the sequential version executed on the managing processing element of the SW26010 processor.

Keywords　SW26010,Floyd algorithm,Parallel computing,Array partitioning

1　引言

全源最短路径作为图论研究中的一个经典抽象问题,在计算机科学、路径规划和游戏开发等许多领域都有着十分广泛的应用。很多现实系统都可以用大型复杂网络来表示,计算顶点之间的最短路径是研究这些网络特征的关键。涉及规模较大的图的最短路径在实际应用程序中很常见,处理起来很有挑战性。因此对于该问题求解算法的并行设计和优化有重要的理论研究和实际应用价值。

目前,关于最短路径问题的研究已有较多的成果[1-3],具体可以分为两类:1)单源最短路径问题的解决方法,如 Dijkstra 算法、Bellman-Ford 算法、SPFA 算法等;2)多源最短路径问题的解决方法,如 n 次 Dijkstra 算法、Floyd 算法等。其中 Floyd 算法是一个经典算法,随着计算机硬件架构的不断升级,尤其是超级计算机的发展,其并行算法的设计和优化依然是研究重点。

"神威·太湖之光"是世界首台峰值性能达 12.5 亿亿次每秒的国产超级计算机[4],已经有很多软件和算法成功移植和优化并得到了非常可观的加速比[5-7]。但目前在神威平台上关于 Floyd 算法求解全源最短路径的研究还处于空缺状

到稿日期:2020-11-05　返修日期:2021-03-21　本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:之江实验室重大科研基金资助项目(2018FD0ZX01)
This work was supported by the Major Scientific Project of Zhejiang Lab Advanced Industrial Network Security Platform(2018FD0ZX01).
通信作者:庞建民(jianmin_pang@126.com)

态，为了让神威平台上的并行计算应用生态更加丰富以及使 Floyd 最短路径算法在神威平台上高效实现，本文首先基于神威平台设计并实现了 Floyd 并行算法，然后根据“神威·太湖之光”超算平台的硬件特性逐步进行优化。

本文的主要贡献如下：

(1)首先设计并实现了基于神威平台的 Floyd 并行算法，提高了程序效率。

(2)针对从核上局部存储空间 LDM(Local Data Memory)有限和较高的通信计算时间之比问题，对于一些不必要的计算进行了算法优化，并采用数组划分和双缓冲优化方法，进一步提高了程序效率。

(3)对随机生成的稠密矩阵采用不同从核数目，测试各输入规模下 Floyd 并行算法的性能。测试结果表明，在 SW26010 处理器上，与主核上的串行算法相比，Floyd 并行算法可以获得 106 倍的最高加速。

本文第 2 节介绍了 SW26010 处理器和 Floyd 算法的相关研究；第 3 节设计并实现了基于神威平台的 Floyd 并行算法；第 4 节给出了 Floyd 并行算法针对神威平台硬件特性的优化；第 5 节针对本文的 Floyd 并行算法进行了测试和分析；最后总结全文并展望未来。

2 相关工作

2.1 SW26010 处理器

处理器是超级计算机强大计算能力的支撑。“神威·太湖之光”的每个运算节点都装有我国自主研发的 SW26010 处理器，其峰值性能超过 100 PFlops/s。该处理器源自 DEC 的 Alpha 架构，工作频率为 1.45 GHz，但其 90.4 GB/s 的实测带宽[8]相对受限。

SW26010 处理器和单核组的结构如图 1 所示。每个处理器由 4 个核组构成，核组之间通过片上网络进行互联，通常作为 4 个单独的节点使用。每个核组由 1 个运算控制核心(MPE)和 1 个运算核心阵列(CPEs)组成，核组内运算核心阵列按照 8×8 的拓扑网络进行布局，可通过寄存器进行通信。整个芯片上集成了 260 个计算核心，每个运算核心有 1 个大小为 64 kB 的可重构局部数据存储、1 个大小为 16 kB 的一级指令 cache 和 32 个大小为 256 位的寄存器。

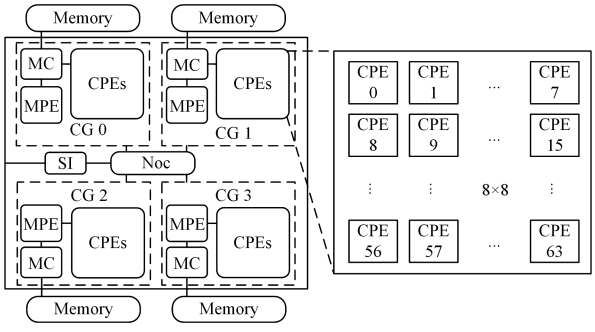

图 1 SW26010 异构众核处理器架构图

Fig. 1 Architecture of SW26010 heterogeneous multicore processor

针对单个运算核心，从核不仅可以通过 gld/gst 直接进行细粒度离散的访问主存，还可以通过 DMA 方式对主存上大量连续的数据进行粗粒度的批量访存。DMA 访存是异步的，因此可通过计算与通信的隐藏来提升程序的性能。

虽然可重构的局部存储给编码带来了困难，但是合理的算法设计和有效协调程序内存占用，可极大地提高程序的执行效率。

2.2 Floyd 算法的相关研究

Floyd 算法最早是由 Floyd[9]在 1962 年提出。基于给定的带有 n 个顶点的无环有向图 $G(V,E)$，通过不断尝试插入两个顶点之间的所有可达顶点，使得两个顶点之间的权值变小则更新两个顶点的权值，否则继续尝试插入下一个可达边的顶点，直到两个顶点之间的所有可达边遍历一遍为止。算法 1 给出了 Floyd 串行算法的伪代码。用矩阵 $\boldsymbol{A}$ 来存储图中任意两顶点之间的权值，令

$$\boldsymbol{A}[i,j]=\begin{cases}0, & (i=j)\\ \text{边}(i,j)\text{的权值}, & (i\neq j \text{ 且 } (i,j)\in E)\\ \infty, & (i\neq j \text{ 且 } (i,j)\notin E)\end{cases}$$

最外层 for 循环用 k 来遍历两个顶点之间所有的可达顶点。中间层 for 循环用 i 来遍历任意两顶点中的源点。最内层 for 循环用 j 来遍历任意两顶点中的汇点。如果所插入的顶点 k 使得 $\boldsymbol{A}[i][k]+\boldsymbol{A}[k][j]<\boldsymbol{A}[i][j]$，说明经过该顶点使得边$(i,j)$之间的距离变短，则在矩阵 $\boldsymbol{A}$ 中用 $\boldsymbol{A}[i][k]+\boldsymbol{A}[k][j]$的值更新 $\boldsymbol{A}[i][j]$的值；否则不做任何处理，继续遍历下一个顶点。最后矩阵 $\boldsymbol{A}$ 中的 $\boldsymbol{A}[i,j]$值就是顶点 i 到顶点 j 的最短路径。

算法 1 Floyd 串行算法

```
input:matrix A,n
output:matrix A
1. for k←0 to n
2.   for i←0 to n
3.     for j←0 to n
4.       if(A[i][j]>A[i][k] +A[k][j])
5.         A[i][j]=A[i][k]+A[k][j]
6. Return A
```

关于 Floyd 算法求解最短路径的研究已有较多成果。Venkataraman 等[10]提出了一种阻塞算法来优化现代处理器的缓存层次结构，在有效利用 cache 方面取得了最重要的突破，该方法提出将矩阵划分为子矩阵，每个子矩阵经过多次处理，改进了求解方法。与标准 CPU 实现相比，该方法实现了 1.9 倍的加速比。基于 CPU 的 Floyd 算法优化主要从减少计算量等角度展开研究[11]。Zuo 等[12]基于全源最短路径的 Floyd 算法设计了求解多重等价最短路径算法，以解决多重等价最短路问题。Lu 等[13]对 Floyd 算法从搜索方向和数据存储方面进行了改进，改进的算法在运行时间和程序占用内存方面均优于传统的 Floyd 算法。

由于 Floyd 算法的复杂度高，在求解大规模图的所有顶点对之间的最短路径时，需要较长的计算时间。因此 Floyd 算法的并行实现和优化是高性能计算领域的研究重点，每当一款新体系结构的处理器出现，基于该处理器的 Floyd 算法的并行实现和优化工作就会相继出现。

针对不同体系结构的 Floyd 算法并行化在国内外取得了显著的成果。Srinivasan 等[14]针对分布式环境提出了一种优

化计算通信的分阶段的并行 Floyd 算法。Teskeredzic 等[15]充分利用 GPU 提供的共享内存克服了串行算法和基本并行实现的缺点，提升了性能。Bondhugula 等[16]使用可编程的门阵列(FPGA)来改进 Floyd 算法，相比于标准的 CPU 实现有几十倍的加速。Xing 等[17]提出了基于棋盘划分方式的 GPU 并行算法，且当图的规模超过 GPU 显存的限制时进一步提出了异步并行处理的 GPU 最短路径算法。

3 SW26010 处理器上 Floyd 并行算法的设计和实现

本节基于 SW26010 处理器，设计并实现了 Floyd 并行算法，根据处理器的特点，采用进程级和线程级并行，提高了 Floyd 算法在 SW26010 处理器上的执行效率。

3.1 SW26010 处理器上 Floyd 并行算法的设计

每个 SW26010 处理器包含 4 个核组，为充分利用计算资源，核组之间采用进程级并行。本文基于 Foster[18]提出的分为 4 个阶段的 PCAM(Partitioning, Communication, Agglomeration和 Mapping)并行算法设计方法对 Floyd 串行算法进行设计。通过分析算法 1 的 Floyd 串行算法可知，该算法对一个赋值语句执行了 n^3 次，因此对其进行域分解。将权值矩阵 $\boldsymbol{A}$ 划分成 n^2 个元素，并对每个元素执行一个原始任务。

如图 2 所示，更新 $\boldsymbol{A}[i,j]$需要访问元素 $\boldsymbol{A}[i,k]$和 $\boldsymbol{A}[k,j]$。这意味着在循环迭代 k 中，第 k 行的每个元素都要被广播到同一列任务中，第 k 列的每个元素也要被广播到同一行任务中。$\boldsymbol{A}[i,k]$和 $\boldsymbol{A}[k,j]$的值在循环迭代 k 中不会改变，因为 $\boldsymbol{A}[i,k]$的更新是按如下方式进行的：$\boldsymbol{A}[i,k]=\min(\boldsymbol{A}[i,k],\boldsymbol{A}[i,k]+\boldsymbol{A}[k,k])$。同理对 $\boldsymbol{A}[k,j]$的更新是按如下方式进行的：$\boldsymbol{A}[k,j]=\min(\boldsymbol{A}[k,j],\boldsymbol{A}[k,k]+\boldsymbol{A}[k,j])$。

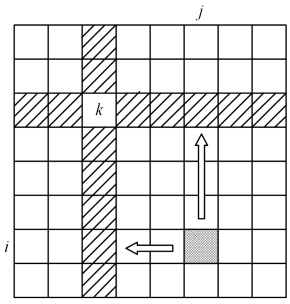

图 2 Floyd 串行算法

Fig. 2 Floyd serial algorithm

由于所有值都是非负的，因此 $\boldsymbol{A}[i,k]$和 $\boldsymbol{A}[k,j]$的值均不会减少。所以 $\boldsymbol{A}[i,j]$的更新与 $\boldsymbol{A}[i,k]$和 $\boldsymbol{A}[k,j]$的更新之间没有依赖关系。总之，对于最外层 k 循环的每一个迭代来说，可以进行广播然后更新权值矩阵 $\boldsymbol{A}$ 中的所有元素。

任务个数是固定的，任务之间的通信模式是结构化的，且每个任务的计算时间是常数，因此本文将任务聚合起来以减少通信开销，每个 MPI 进程对应一个任务。

对于将 n^2 个原始任务聚合成 p(进程数)个任务，有 3 种常用的聚合方法：按行、按列和按块对任务进行分组。按行划分时算法总的复杂度为 $O(n^3/+n^2\log p)$。按列划分时算法总的复杂度为 $O(n^3\sqrt{p}+n^2\log\sqrt{p})$。按块划分时算法总的复杂度是 $O(n^3/\sqrt{p}+n^2\log\sqrt{p})$。按块划分的时间复杂度最低。因此本文选择按块划分，每个进程负责一块大小为$(n/\sqrt{p})\times(n/\sqrt{p})$的矩阵。

针对 SW26010 的主从架构特点，在核组内的从核上采用线程级并行。由于每个从核的 LDM 大小只有 64 kB，当 n 比较大时，不足以放下大小为$(n/\sqrt{p})\times(n/\sqrt{p})/64$ 的数据，因此每个从核上只能存放部分数据。与主核上的划分相同，从核上的矩阵也有 3 种划分方法：按行划分、按列划分和按块划分。由于 DMA 连续访存的限制，本文放弃按列划分的方法。虽然进程级划分时按块划分的效果更好，但是由于 DMA 访存带来的开销，从核上的划分暂时还不能通过理论分析得出最优的划分方法。

3.2 SW26010 处理器上 Floyd 并行算法的实现

首先将权值矩阵 $\boldsymbol{A}$ 按块划分给各进程(进程数为$\sqrt{p}*\sqrt{p}$)并创建行通信域和列通信域，然后包含第 k 行(列)数据的进程将需要发送的部分 k 行(列)数据广播至列(行)通信域的其他进程，最后每个进程根据已有的或接收到的第 k 行和第 k 列的部分数据计算最短路径并更新权值矩阵 $\boldsymbol{A}$。一直循环执行，直到行(列)通信域的每个进程都将权值矩阵 $\boldsymbol{A}$ 中的行(列)都广播过一次为止。

基于以上进程级别的按块划分，虽然可以使程序的执行效率得到极大提高，但是计算时间占总执行时间的 80%，计算通信时间之比较高，属于计算密集型。根据 Amdahl 定律，计算部分需要进一步并行。

为将计算部分进一步并行，应充分利用 SW26010 处理器上从核的强大计算能力。如图 3 所示，主核主要负责串行代码核心段 A 的 I/O、通信和少量的计算，而拥有大量计算的核心代码段 B 将放在从核上进行加速计算，并把从核加速计算时主核处于等待状态的并行模式称为主从加速并行。

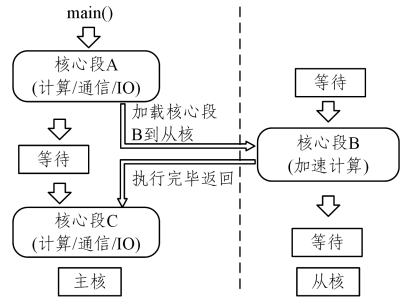

图 3 主从加速并行

Fig. 3 Master and slave accelerate parallel

主核有两种方法调用从核集群的计算资源：一种是类似于 OpenMP 的编程方法 OpenACC*，在需要并行的程序前面添加指导语句进行自动并行化，然后编译器根据标识将代码编译成并行代码；另一种是使用 Athread 加速线程库进行手工并行化。

为了更好地控制从核 LDM 的使用情况，本文使用 Athread 加速线程库实现核心代码段 B 在从核上的并行。Athread 加速线程库是针对两级并行编程模型所涉及的程序进行加速计算，起到对核组内的线程进行灵活控制的作用，使其能

够更好地发挥核组内多从核并发执行的加速性能。Athread 加速线程库主要用于控制线程的初始化、启动、结束等一系列操作。

如图 4 中的主核代码所示，主核通过调用 Athread 加速线程库中的 athread_init()，athread_spawn()，athread_join() 和 athread_halt() 4 个函数接口来实现对 Floyd 算法在从核上的并行。由于从核线程的初始化函数 athread_init() 和结束函数 athread_halt() 在并行优化过程中只能被调用一次，因此不仅需要在循环最外层开始处调用 athread_init() 函数接口对从核线程进行初始化，还需要在循环最外层结束处调用 athread_halt() 函数接口来结束从核线程。调用 athread_spawn() 函数接口用于启动核组中的所有可用从核资源，用 athread_join() 函数接口显式阻塞调用该线程组，直到指定的线程终止为止。

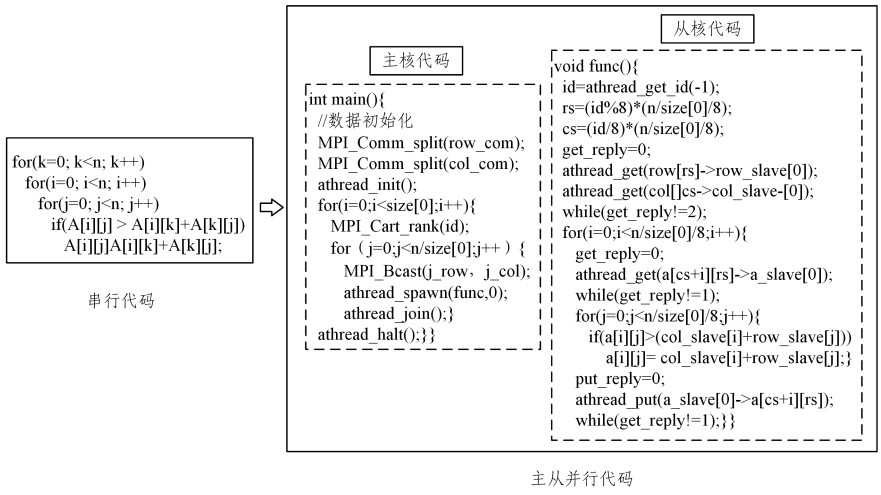

图 4 Floyd 串行核心代码和主从核并行核心代码

Fig. 4 Floyd serial core code and master-slave parallel core code

如图 4 中从核代码所示，从核通过调用 Athread 加速线程库中的 athread_get() 和 athread_put() 这两个函数接口进行数据传输。athread_get() 函数接口用于将主存中指定位置的数据读取到从核 LDM 中指定的位置。当从核上的计算完成时，通过调用 athread_put() 函数接口将从核 LDM 中的数据写回主存中对应的位置。根据 3.1 节可知，将数据按块划分时的时间复杂度最低，扩展性最好，因此将各个进程的数据按块分给 8×8 的从核阵列。由于从核 LDM 的大小有限，不能存放计算时所需的所有数据，因此在手工写从核代码时必须以字节为单位提前计算好每个从核将要存放数据的大小，提前将计算所需的 row 数组放在 LDM，防止从核空间溢出。

4 SW26010 处理器上 Floyd 并行算法的优化

虽然第 3 节实现了 Floyd 算法的进程级和线程级并行，并给程序带来了明显加速，但是主从加速并行的实现既没有考虑从核与主存之间的访存开销，也没有充分地利用主从核的通信和计算部件，因此提高访存效率和充分地利用通信和计算部件成为下一步优化的重点。本节将使用 3 种优化方法解决以上问题。

4.1 算法优化

基于有向无环有权图对 Floyd 进行算法层面的优化。计算节点 i 和 j 之间的最短路径时，对待插入的节点 k 先进行权值判断，若 $\boldsymbol{A}[i][k]$为不可达，则说明插入节点 k 后明显不能使节点 i 到节点 j 的路径变短，因此不需要再计算 $\boldsymbol{A}[i][k]+A[k][j]$，直接进入下一个节点的搜索。该方法虽然使其时间复杂度与串行的时间复杂度一样，但是所需的总的计算量减少了。每减少一个循环迭代步就减少了一次从核 athread_get() 和 athread_put() 函数的调用，同时还减少了 $n/\sqrt{p}$ 次计算和 if 判断。

4.2 数组划分

表 1 列出了单核组的 DMA_GET 的实测性能数据，测试时主存地址递增，每次都判断回答字。随着传输粒度逐渐增大，单从核和单核组的访存带宽逐渐变大。DMA 控制器处理 DMA 请求的能力有限，当 DMA 请求次数较少时，DMA 控制器能够较好地处理 DMA 请求，因此减少 DMA 的请求次数是降低从核访存时间的有效途径。虽然有多种数组的划分方法，但是总的数据量是一定的，只有当一次存取的数据块最大时，DMA 的请求次数才最少，才能充分地利用带宽。

表 1 单从核模式 DMA_GET 的性能

Table 1 DMA_GET performance in single-slave kernel mode

Granularity/B	1 pebw/ (GB/s)	64 pebw/ (GB/s)
512	2.65	28.98
1024	4.28	27.97
2048	6.33	30.48
4096	8.14	30.18
8192	9.47	30.78

为充分利用从核和主核之间的访存带宽，应尽量增大传输数据的粒度，对数组划分进行优化。如果按块处理，每个从核一次接收和计算一行的一部分数据。使用分块的方法，数组 $\boldsymbol{A}[8][8]$被分配到 4 个 CPEs，总共需要对 ***a_row*** 矩阵中的数据进行 32 次存取操作，如图 5 的左部分所示。在采用图 5 右部分所示的带状划分后，将从核阵列按行划分，每个从核分得两行数据，只需要进行 16 次存取操作，提高了效率。在实际应用程序中，矩阵的规模更大且开启的从核数更多，对程序的性能提升得更高。当从核的 LDM 不足以放下计算所需要的数据时，可以让两个或者多个从核计算一行数据，即从核采用带状划分与块状划分相结合的方式。以上两种划分不仅增

大了传输数据的粒度，还降低了数据传输的次数。

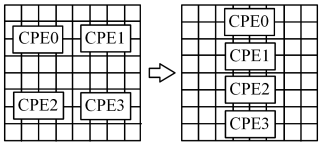

图 5 从核数据的划分

Fig. 5 Partition of data from slave

从理论上分析，不论是时间复杂度还是可扩展性，矩阵的块状划分的效果均好于按行划分，但由于从核 LDM 大小、DMA 传输带宽和 DMA 传输粒度的特有限制，导致理论与实践存在偏差。

基于表面容积效应，通信量与任务子集的表面成正比，计算量与任务子集的体积成正比。增加任务子集的表面，不仅可以提升通信量，还增加了计算量，有利于使用双缓冲进行下一步的优化。

4.3 双缓冲

为继续降低通信时间占比，本文将使用双缓冲进行优化。双缓冲优化方法的使用主要是为了充分地利用通信和计算部件将通信时间和计算时间隐藏，从而减少程序总运行时间。本文设置 LDM 上 a_row 的双缓冲，每个从核先获得本次计算所需的 row、col 和 a_row[index]数组，计算 a_row[index]时获取下次计算所需数据 a_row[next]，然后将本次计算结果 a_row[index]传回到主存。最后一次迭代结束后，对最后一次迭代取得的数据进行计算并将结果传回到主存。以此达到用部分通信时间来隐藏计算时间，减少总运行时间的目的。

图 6 上部分展示了单缓冲的计算和访存流程，下部分展示了双缓冲的计算和访存流程。在实际优化过程中，并行 Floyd 算法中的 DMA 传输时间所占比重较大，而计算时间所占比重较小，所以该优化对程序的整体性能提升较小。

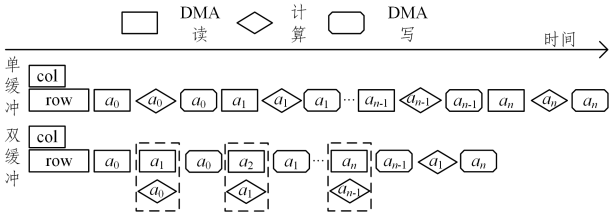

图 6 双缓冲

Fig. 6 Double buffering

5 测试结果

本文实验在单个 SW26010 处理器上进行。实验所用数据为随机生成的稠密矩阵。Floyd 并行算法的正确性测试是比较并行和串行的运行结果。Floyd 并行算法的性能测试分为 3 部分，首先是测试一级并行和两级并行的性能并分析其计算通信时间之比，其次是测试当每个核组开启 64 个从核时数组划分优化的性能并分析其通信计算时间之比，最后是测试当每个核组开启 16 个从核时数组划分优化的性能并分析其通信计算时间之比。

5.1 主核并行测试

图 7 给出了 3.1 节中进程级的任务划分方法(MPI)和在此基础上在从核上实现线程级并行(MPI_Block)的加速比。从图中可以看出，MPI_Block 版本的加速比随着矩阵规模的增大呈直线上升趋势，而 MPI 版本的加速比趋于水平，因此 MPI_Block 版本大大提升了程序性能。

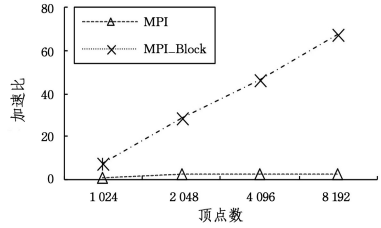

图 7 基于串行的加速比

Fig. 7 Serialization-based acceleration ratio

下面进一步分析 MPI 和 MPI_Block 两个版本主核上的计算通信时间之比。如图 8 所示，MPI_Block 版本的计算通信时间的之比远远小于 MPI 版本，实现了更合理的任务划分，充分利用了从核的计算资源。

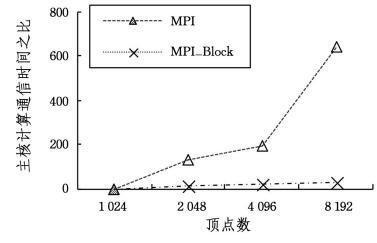

图 8 $MPE=4$ 时主核上程序的计算通信时间之比

Fig. 8 Ratio of computational communication times of programs on master core when $MPE=4$

5.2 cgsp=64 时数组划分的优化效果

虽然 MPI_Block 版本的计算通信时间之比与 MPI 版本相比较低，但是相对于自身的 MPI 通信时间，计算成为下一步优化的核心。由于数组的二维划分有多种划分方法，不同的方法对 DMA 的传输有较大的影响，导致优化效果相差较大。图 9 给出了 4 种划分方法下基于 MPI 版本的加速比，分别是 8×8(block)，4×16(row_4)，2×32(row_2)的二维划分和 1×64(row_1)的一维划分。由图 9 可知，在 LDM 可以存储所需计算数据时，一维划分的效果最好。是因为一维划分时，DMA 的传输粒度和请求次数最低。由表 1 可知随着传输粒度的增大，DMA 的传输带宽也随之增大，因此当数据规模增大时，一维划分和二维划分基于 MPI 版本的加速比均逐渐增大。

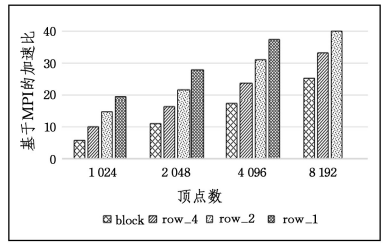

图 9 $CPE=64$ 时基于 MPI 的加速比

Fig. 9 MPI based acceleration ratio when $CPE=64$

如图 10 所示，使用 64 个从核进行计算时，从核上程序的通信计算时间之比高达 9 倍，说明从核大部分时间都处于等待数据的状态，其计算部件的大部分时间都处于空闲状态，计算资源没有得到充分利用。此时从核访存受限，同时解释了双缓冲与单缓冲的效果几乎一样的原因。

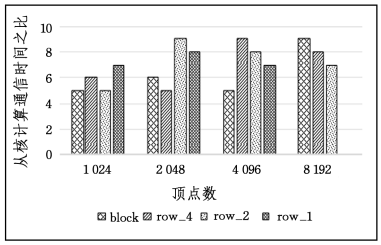

图 10 *CPE*=64 时从核上程序的通信计算时间之比

Fig. 10 Ratio of communication computation times of programs on slave core when *CPE*=64

5.3 cgsp=16 时数组划分的优化效果

为了分析从核上通信计算时间之比较高的原因，本节采用 16 个从核进行测试。相比于 64 个从核，在数据规模和划分方式相同的情况下，16 个从核每次调用从核 func()函数时，每个从核的 get 和 put 次数相同，计算量也相同，保证了其与使用 64 个从核时相同的测试条件。

图 11 给出了使用 16 个从核计算时基于 MPI 版本的加速比，其与图 9 相比普遍优于使用 64 个从核计算时的加速比。

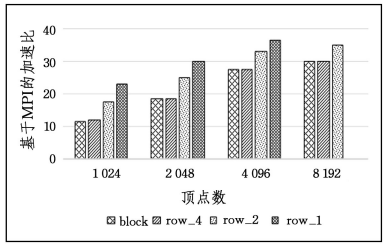

图 11 *CPE*=16 时基于 MPI 的加速比

Fig. 11 MPI based acceleration ratio when *CPE*=16

在每个从核 get 和 put 次数、传输粒度和计算量相同的情况下，使用 16 个从核计算时总的 get 和 put 次数远小于 64 个从核的 get 和 put 次数。如图 12 所示，从核上程序的通信计算时间之比明显降低。说明了从核之间激烈的带宽竞争，即使使用 64 个从核也达不到预期的加速效果。

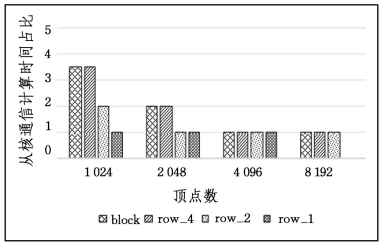

图 12 *CPE*=16 时从核上程序的通信计算时间之比

Fig. 12 Ratio of communication on computation times of programs on slave core when *CPE*=16

结束语 本文首次在神威平台上对 Floyd 并行算法进行实现和优化。本文基于主从核架构采用进程级和线程级的并行化，并利用了算法优化、数组划分优化和双缓冲方法来优化 Floyd 并行算法。本文通过实验评估了不同从核数目、各输入规模下 Floyd 并行算法的性能。结果表明，与串行 Floyd 算法相比，优化后的并行 Floyd 算法在 SW26010 处理器上可达到最高 106 倍的加速比。本文后续工作是通过使用寄存器通信来进一步缓解从核的访存压力。

参 考 文 献

[1] DOR D, HALPERIN S, ZWICK U. All pairs almost shortest paths[C]// SIAM Journal on Computing. 1996, 29(5): 452-461.

[2] CHAN T M. More algorithms for all-pairs shortest paths in weighted graphs[J]. Proceedings of the Annual ACM Symposium on Theory of Computing, 2010, 39: 2075-2089.

[3] LI J W, ZHANG J, ZHAO J C, et al. A Load Balancing Shortest Path Routing Algorithm for SRIO Network[J]. Computer Engineering, 2020, 46(3): 214-221, 228.

[4] DONGARRA J. Report on the sunway taihulight system[D]. Knoxville: University of Tennessee, 2016.

[5] LIU X, GUO H, SUN R J, et al. The Characteristic Analysis and Exascale Scalability Research of Large Scale Parallel Applications on Sunway TaihuLight Supercomputer[J]. Chinese Journal of Computers, 2018, 41(10): 2209-2220.

[6] LI M, YANG C, SUN Q, et al. Enabling highly efficient k-Means computations on the SW26010 many-core processor of sunway taihulight[J]. Journal of Computer Science & Technology, 2019, 34(1): 77-93.

[7] NI H, LIU X. Multi-Core optimization technology of unstructured grid based on Sunway TaihuLight[J]. Computer Engineering, 2019, 45(6): 45-51.

[8] XU Z, LIN J, MATSUOKA S. Benchmarking SW26010 many-core processor[C]// Parallel & Distributed Processing Symposium Workshops. IEEE, 2017.

[9] FLOYD R W. Algorithm 97, Shortest path algorithms[J]. Communications of the ACM, 1962, 5(6): 345.

[10] VENKATARAMAN G, SAHNI S, MUKHOPADHYAYA S. A blocked all-pairs shortest-paths algorithm [C]// Scandinavian Workshop on Algorithm Theory. Berlin, Heidelberg: Springer, 2000.

[11] ZHANG D Q, WU G L, LIU D F. Accelerated and Optimized Method of Floyd Algorithm to Find out Shortest Path[J]. Computer Engineering and Applications, 2009(17): 45-47, 50.

[12] ZUO X F, SHEN W J. Improved Algorithm about Multi-shortest Path Problem Based on Floyd Algorithm [J]. Computer Science, 2017, 44(5): 238-240, 273.

[13] LU L G, LIU L Y, LU T D, et al. A Modified Floyd Algorithm [J]. Journal of East China University of Technology, 2019, 42(1): 81-84.

[14] SRINIVASAN T, BALAKRISHNAN R, GANGADHARAN S A, et al. A scalable parallelization of all-pairs shortest path algorithm for a high performance cluster environment[C]// International Conference on Parallel & Distributed Systems. IEEE, 2007.

[15] TESKEREDZIC E, KARAHODZIC K, NOSOVIC N. Compari-

son of the non-blocked and blocked floyd-warshall algorithm with regard to speedup and energy saving on an embedded GPU [C]// 19th International Symposium INFOTEH-JAHORINA. 2020:18-20.

[16] BONDHUGULA U, DEVULAPALLI A, FERNANDO J, et al. Parallel FPGA-based all-pairs shortest-paths in a directed graph [C]// 20th International Parallel and Distributed Processing Symposium(IPDPS 2006). IEEE, 2006.

[17] XING X X, ZHAO G X, LUO Z Y, et al. GPU-based Algorithm of Shortest Path[J]. Computer Science, 2012, 39(3):299-303.

[18] FOSTER I. Designing and building parallel programs: concepts and tools for parallel software engineering[J]. Tetrahedron Letters, 1995, 11(3):296-300.

HE Ya-ru, born in 1994, postgraduate. Her main research interests include high-performance computing and so on.

PANG Jian-min, born in 1964, professor, Ph. D supervisor, is a member of China Computer Federation. His main research interests include high-performance computing and information security.