软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, 2022,33(12):4452−4463 [doi: 10.13328/j.cnki.jos.006381]

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

SW26010 处理器上的并行结构化稀疏三角方程组求解器*

陈道琨[1,2], 刘芳芳[1,2], 杨 超[3]

[1](中国科学院 软件研究所 并行软件与计算科学实验室, 北京 100190)

[2](中国科学院大学, 北京 100049)

[3](北京大学 数学科学学院, 北京 100871)

通信作者: 杨超, E-mail: chao_yang@pku.edu.cn

摘 要: 稀疏三角线性方程组求解(SpTRSV)在科学与工程计算领域是比较重要的核心计算函数, 其中基于结构化网格构造的线性方程组是 SpTRSV 求解器经常遇到的一类问题. 在国产神威·太湖之光超级计算机所配备的 SW26010 处理器上, SpTRSV 求解器通常需要结合该平台的架构特点, 通过搭建一定的数据路由体系来满足各工作线程对未知量数据的需求. 面向与结构化网格相关的稀疏三角方程组问题, 提出一套适用于 SW26010 处理器的并行求解器. 该求解器在任务划分阶段将各线程的数据依赖模式限制在相对可控的范围之内, 并在无数据路由的条件下解决线程的通信问题, 不仅消除了数据路由带来的额外通信开销, 而且适用的问题范围也不再受数据路由规则的制约. 经测试, 针对多种不同类型的结构化网格问题, 提出的求解器框架的平均访存带宽利用率达 88.2%, 部分问题的访存带宽可达平台峰值带宽的 94.5%(24.5 GB/s), 整体性能相比现有工作有较为明显的提高.

关键词: 稀疏三角线性方程求解(SpTRSV); 结构化网格; SW26010 处理器; 异构计算

中图法分类号: TP301

中文引用格式: 陈道琨, 刘芳芳, 杨超. SW26010 处理器上的并行结构化稀疏三角方程组求解器. 软件学报, 2022, 33(12): 4452−4463. http://www.jos.org.cn/1000-9825/6381.htm

英文引用格式: Chen DK, Liu FF, Yang C. Parallel Sparse Triangular Solver for Structured Grid Problems on SW26010 Processor. Ruan Jian Xue Bao/Journal of Software, 2022, 33(12): 4452−4463 (in Chinese). http://www.jos.org.cn/1000-9825/6381.htm

Parallel Sparse Triangular Solver for Structured Grid Problems on SW26010 Processor

CHEN Dao-Kun[1,2], LIU Fang-Fang[1,2], YANG Chao[3]

[1](Laboratory of Parallel Software and Computational Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)

[2](University of Chinese Academy of Sciences, Beijing 100049, China)

[3](School of Mathematical Sciences, Peking University, Beijing 100871, China)

Abstract: Sparse triangular solver (SpTRSV) is an important computation kernel in scientific computing. The irregular memory access pattern of SpTRSV makes efficient data reuse difficult to achieve. Structured grid problems possess special nonzero patterns. OnSW26010 processor, the major building block of Sunway Taihulight supercomputer, these patterns are often exploited during the task partitioning stage to facilitate on-chip reuse of computed unknowns. Software-based routing is usually employed to implement inter-thread communication. Routing incurs overhead and imposes certain restrictions on nonzero patterns. This study achieves on-chip data reuse without routing. The input problem is partitioned and mapped onto SW26010 such that threads with data dependencies are always connected by the register communication network. This enables direct thread communication and obviates routing. The proposed solver is described and it is tested over a variety of problems. In the experiments, the proposed solver sustains an average memory bandwidth utilization of 88.2% with peak efficiency reaching 94.5% (24.5 GB/s).

Key words: sparse triangular solver (SpTRSV); structured-grid; SW26010 processor; heterogeneous computing

* 基金项目: 国家重点研发计划 (2020YFB0204601,2016YFB0200603)

收稿时间: 2019-11-17; 修改时间: 2020-04-09; 采用时间: 2020-05-07; jos 在线出版时间: 2021-08-03

神威·太湖之光是我国自主研制的超级计算机, 浮点计算能力连续多年居世界前列[1], 已经在诸如气象模拟及地质分析等大型应用[2−5]中发挥了重要的作用. 在科学与工程计算领域, 稀疏三角方程组求解器(sparse triangular solver, SpTRSV)是一个重要的核心计算函数[2,6], 而且往往是应用的性能瓶颈[7]. 神威·太湖之光的主要计算部件是国产 SW26010 处理器. 在该处理器上, 受架构的制约并非所有线程相互之间都能够进行通信. 然而 SpTRSV 为追求高度的并发性又必须将任务划分到不同的工作线程, 其间线程不可避免地需要交换方程组的未知量信息. 在 SW26010 处理器上, 如何在任务划分和线程通信之间找到一个平衡点是 SpTRSV 求解器需要面对与解决的关键问题. 针对模板计算(stencil computation)中与结构化网格有关的方程组问题, 现有方案[8,9]主要通过利用结构化网格的信息, 在尽量契合 SW26010 处理器架构特点的条件下对问题进行划分. 对于无法满足平台架构约束的线程通信, 现有研究多以软件形式建立数据路由机制来解决. 数据路由会增加线程通信的开销, 同时框架的适用问题范围也会在一定程度上受路由规则设计的影响. 本文面向结构化网格相关的稀疏方程组问题, 基于 SW26010 处理器平台提出一套并行 SpTRSV 求解器框架, 在不引入数据路由的前提下解决线程的通信问题.

1 背景介绍

1.1 稀疏三角方程组问题

为方便本文假设矩阵以 CSC 格式[10]存储(SpTRSV 在预处理阶段可按算法的需求任意排布矩阵非零元数据, 矩阵的存储格式对算法而言几乎没有影响), 稀疏上/下三角方程组求解过程类似, 本文以下三角方程组为例进行介绍. 算法 1 是串行求解器的伪代码, 其中, I 代表每列的起始存储位置, J 和 A 分别为非零元的行号以及数值, $A[\omega]$是每一列的对角项. 数组 B 用于存放问题的初始右端项以及最终解. SpTRSV 每进行一次浮点乘加运算(第 6 行)需要访问 4 个操作数, 属典型的访存密集型(memory-bound)算法.

算法 1. 稀疏下三角方程组求解过程.

CscBacksub($I[N+1],J[*],A[*],x[N]$)

```
01  for p=0 to N−1 do
02      ω←I[p]
03      x[p]←A[ω]^{-1}x[p]
04      for q=I[p]+1 to I[p+1] do
05          δ←J[ω+q]
06          x[δ]←x[δ]−A[q]x[p]
07      end
08  end
```

当作为不完全 LU 矩阵分解(ILU(k))[11]预条件子的一部分时, SpTRSV 在每个迭代步中会多次复用分解得到的矩阵[12]. 因此在实际应用中允许对矩阵非零元进行一定的处理(称为预处理), 为后续求解提供便利. 一般情况下, 预处理的开销计入总迭代时间, 由所有 SpTRSV 的调用均摊(amortized)[13].

1.2 分层调度(Level Scheduling)算法

目前 SpTRSV 主流的并行算法以分层调度算法[14,15]为基础, 该算法将构成线性方程组的方程视作任务依赖图(task-dependency graph, TAG)的顶点, 将非零元所决定的未知量依赖关系看作调度图上的有向边. 图 1 是任务依赖图的一个例子, 其中顶点 1–4 对应等式 $E1$–$E4$, 有向边 a–e 则代表它们之间的未知量依赖关系. 由于任务依赖图是无环的, 按未知量依赖关系对任务依赖图的顶点进行拓扑排序可划分得到若干任务集合. 集合内部的顶点(方程)之间不存在任何未知量依赖关系, 因此可并行进行求解. 分层调度算法(伪代码如算法 2 所示)利用该特性将每个任务集内部的顶点(方程)分配到当前可用的线程资源上进行求解(算法 4–6 行), 每当一个任务集的处理工作完成之后算法会发起全局的线程同步操作(算法第 9 行)以确保当前计算结果(算法第 7 行

向内存写入的 $x[s_{ij}]$)对其他线程可见, 从而防止线程访存顺序的不同对计算结果产生影响.

$E1$: $l_1x_1=b_1$

$E2$: $l_2x_2=r_2-ax_1$

$E3$: $l_3x_3=r_3-bx_2$

$E4$: $l_4x_4=r_4-cx_1-dx_2-ex_3$

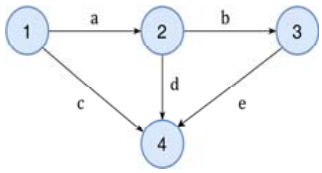


图 1 方程组 E1-E4 以及它们的任务依赖图

算法 2. 分层调度算法的伪代码.

Level Scheduling ($I[N+1],J[*],A[*],x[N]$)

01 Generate the set S of super-tasks.
02 **foreach** super-task $S_i \in S$ **do**
03 **parallel foreach** row $s_{ij} \in S_i$ **do**
04 **foreach** off-diagonal $A[\gamma]$ on row s_{ij} **do**
05 $x[s_{ij}] \leftarrow x[s_{ij}]-A[\gamma]\times x[\gamma]$.
06 **end**
07 $x[s_{ij}] \leftarrow A[I[s_{ij}]]^{-1}\cdot x[s_{ij}]$.
08 **end**
09 global thread synchronization.
10 **end**

1.3 SW26010处理器

神威·太湖之光的主要计算部件是国产 SW26010 处理器. 该处理器采用异构众核架构, 由 4 个独立的核组(core group, CG)组成, 图 2 是其硬件架构的示意图. 核组间通过片上网络(network on chip, NoC)互联. 每个核组包含运算控制核心(MPE, 主核)以及一个以 8×8 阵列结构的核簇(CPE cluster, 从核组). 核组的内存访问通过集成内存控制器(iMC)进行, 从核随机访存操作的延迟较高[16]. 在存储器层次方面, 主核配有独立的数据和指令缓存体系, 从核配备指令缓存体系(一级 16 KB, 二级共享 64 KB), 并提供 64 KB 高速可重构存储空间(LDM). 从核可通过直接内存访问模式(direct memory access, DMA)对主存中连续的数据区域进行操作, 在大粒度条件下 DMA 访存的带宽可接近平台访存带宽的峰值[17].

从核阵列由二维环面网格状的寄存器通信网络连接, 处在同行或同列的从核可通过该网络互相传递数据, 数据的传输以 32 字节为单位. 寄存器通信的发送与接收分别为两条独立的指令, 从核程序要在发送方(接收方)从核各执行一次相应的发送(接收)操作才能完成数据包的传递. 从核已发送但尚未接收的数据包将被临时存储于接收方从核的寄存器通信缓冲队列, 该队列最多可容纳 13 个数据包. 发送(接收)数据包时发送方(接受方)从核会阻塞直至相应队列有足够空间容纳所发送的(存在未被接收的)数据包. SW26010 处理器配备独立的行列通信总线和缓存队列, 从核在行列方向上的寄存器通信彼此独立.

1.4 研究现状

分层调度算法线程全局同步操作的开销较大, 部分研究采用点对点同步等替代机制来实现类似的功能来缓解全局同步的开销问题[18–20], 部分工作对冗余的点对点同步操作进行了优化[13]. 分层调度算法求解过程中由 $x[s_{ij}]$引起的离散访存次数较多, 算法数据局部性不好. 针对这个问题, 部分研究放弃了该算法的问题划

分模式, 转而以按硬件特点构造的性能模型为导向, 通过模型预测结果来选择性能较优分块方案, 目前在 GPU 平台已经取得了较为良好的效果[21,22]. SW26010 处理器离散访存的延迟较高, 相关研究大多从访存优化的角度开展工作, 其中具有代表性的工作是 swSpTRSV 求解器[8]以及针对结构化网格问题 SpTRSV 求解器(本文简称 GridSpTRSV)[9], 它们在各自的任务划分体系下, 通过软件形式复用 SpTRSV 的未知量数据, 从而实现减少从核离散访存的目的.

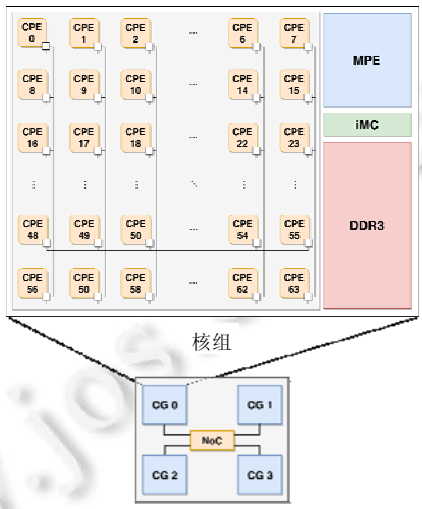


图 2 SW26010 处理器的架构示意图

swSpTRSV 将一部分 LDM 设置为变量 $x[p]$的临时缓冲空间, 在计算 $A[q]x[p]$之前会首先尝试查找并复用缓冲区内的$x[p]$, 若查找失败就从内存读取$x[p]$并放入缓冲区以备重用, 并设置了相应的数据丢弃规则来应对缓冲区空间不足的情况. 该方法适用一般意义上的稀疏方程组问题, 其数据复用规则没有利用与结构化网格相关非零元分布规律, 所以在处理这类问题时访存较多, 数据复用的比例不高. 通信方面 swSpTRSV 建立了点对点的通信网络来满足线程的通信需求, 其中部分从核负责根据设定的数据路由规则来转发其它从核的寄存器通信. P2P 网络中数据在抵达目标从核之前需要经过多次中间转发, 中转产生的时间开销较大, 且网络中提供路由功能的从核也无法参与计算, 这些都对求解器的性能有比较负面的影响.

为了改善未知量数据的复用比例, GridSpTRSV 转而以矩阵非零元分布为依据对问题进行划分, 之后再根据从核对数据需求以及方程未知量的分布情况, 有针对性地去规划从核的数据路由路径. 在这种模式下该工作不仅可以通过发掘问题非零元的分布规律来提高未知量的数据复用比例, 而且数据路由的规则比较简单, 从核可在求解方程的同时进行数据的转发操作, 路由不会占用从核的计算资源. 相对于上一个工作, 虽然 GridSpTRSV 框架的执行效率有了明显提高, 但受限于数据路由规则的特殊性, 仅支持三维 7 点格式以及 27 点格式等几种特定类型的问题, 而且时常需要根据问题类型对数据路由路径的设置进行调整.

本文面向 SW26010 处理器上结构化网格问题, 提出一套全新的 SpTRSV 并行求解器, 在数据复用的基础之上将从核通信的因素纳入了考虑的范畴. 在本文的任务划分体系下, 无论问题的网格类型, 从核始终都表现出完全相同的数据依赖关系. 这使得本文能够通过一定方式在无数据路由的条件下满足所有从核线程的未知量数据需求. 本文在解决通信问题时实现了未知量数据的完全复用, 访存仅包含稀疏矩阵本身以及问题的输入输出, 全局访存量在满足某些条件假设的情况下为 SpTRSV 问题的理论最优访存量. 借助通信以及数据复用上的优化, 本文在效率上相比已有方案有较大幅度的提高.

2 主要贡献

本文试图通过数据复用来完全消除在右端项累加运算(算法 1 第 6 行)中由 $x[p]$引起的离散访存, 在该限制条件下寄存器通信是从核获取未知量信息的唯一途径. 目标是让从核不经转发就能交换 $x[p]$, 这就要求在求解过程中 $x[p]$数据的供给方与接收方之间在寄存器通信网络上必须始终存在一条可行的通信路径, 对框架的任务划分策略提出了比较高的要求. 如何在满足寄存器通信约束条件的前提下寻找各方面兼顾的划分策略是整个求解器设计的核心同时也是难点所在.

2.1 任务划分

本文将框架任务划策略分视为一个和任务依赖图(记为图 G)的顶点有关的函数 T, 函数 T 给每个顶点指定一个标签(取值范围 0–63), 表示该顶点(计算任务)的所属从核线程. 函数 T 将任务依赖图的顶点划分成了若干互不相交的子集, 分别表示 64 个从核线程在并行阶段负责求解的方程以及它们所要存储的未知量信息($x[p]$). 在函数 T 的基础上构造一个新有向图 $H_{G,T}$, 其顶点集和有向边集合如下:

$$V(H_{G,T})=\{T(u)\,|\,u\in V(G)\},$$

$$E(H_{G,T})=\{T(u)\to T(v)\,|\,u\to v\in E(G)\}.$$

如将图 $H_{G,T}$ 的顶点视为参与计算的从核, 那么从顶点 $T(u)$到顶点 $T(v)$的有向边($T(u)\to T(v)$)表示由从核 $T(v)$负责求解的方程与由从核 $T(u)$负责计算的方程之间存在未知量依赖关系, 从核 $T(v)$在求解过程中需要使用从核 $T(u)$的计算结果, 因此从核 $T(u)$与从核 $T(v)$在寄存器通信网络上必须存在一条通信路径.

从核环面网格状的寄存器通信网络本身也可被视作为一个有向图. 本文将其记为图 $R_0=K_8\square K_8$, 其中, K_n 代表 n-完全图,符号 $\square$ 为图笛卡尔乘积运算(graph Cartesian product)). 倘若在某种划分方式下图 $H_{G,T}$ 与图 R_0 形成子图同构关系, 那么在 $H_{G,T}$ 和 R_0 之间存在从集合 $V(H_{G,T})$到集合 $V(R_0)$的同构映射函数, 使得图 $H_{G,T}$ 中的顶点在映射后在图 R_0 中仍保留原有的邻接关系(若 S 是从 $H_{G,T}$ 到 R_0 的同构映射函数, 对任意两顶点 $p,q\in V(H_{G,T})$, 如果 $p\to q\in E(H_{G,T})$成立, 那么, $S(p)\to S(q)\in E(R_0)$也成立). 若同构映射的形式已知, 可据此将负责计算的线程($V(H_{G,T})$)与物理从核($V(R_0)$)之间进行对应, 使得在求解过程中从核之间的寄存器通信路径($E(H_G)$)始终保持在寄存器通信网络支持的范畴($E(R_0)$)之内, 在满足从核数据需求的同时避免对 $x[p]$数据进行二次转发. 综上, 本文在任务划分阶段实际上要解决两个子问题: 假设待解问题的任务依赖图确定, 首先要寻找一个划分函数 T, 在该函数所确定的划分方案下图 $H_{G,T}$ 与图 R_0 之间必须形成子图同构关系, 而后需要找到从图 $H_{G,T}$ 到图 R_0 的同构映射函数, 以最终确定计算任务所在的从核.

本文首先固定待解问题的方程编号(记为 r)与三维空间坐标之间的线性对应关系:

$$F_{XYZ}(x,y,z)=zXY+yX+x,$$

$$F^{-1}_{XYZ}(r)=\left(r\bmod X,\ \frac{r}{X}\bmod Y,\ \frac{r}{XY}\right),$$

其中, XYZ 为线性对应关系的参数, 它们的乘积要不小于稀疏矩阵的总行数. 由于方程组与其任务依赖图之间存在相互对应关系, 通过方程组的行号映射 F_{XYZ} 自然也把任务依赖图嵌入[23]到了三维空间当中, 给任务依赖图中各顶点赋予了几何意义, 因此可以在各顶点之间定义几何意义上的距离. 本文约定顶点间的距离$(\cdot)$为它们对应坐标点之间的 L_1 范数距离, 采用如下 T'作为框架的任务划分函数:

$$T'(w)\equiv T'(w_x,w_y,w_z)=\frac{w_y}{B_Y},$$

$$B_Y\geqslant\max\left(R(G)\equiv\sup_{u\to v\in E(G)}u,v,\frac{Y}{64}\right),$$

其中, B_Y 是决定分块大小的参数, 需要满足右端的限制条件. 从函数 T'的表达式可知对图 $H_{G,T}$ 中任意顶点, 如下不等式成立:

$$|T'(u)-T'(v)|\leqslant 1,$$
$$T'(u),T'(v)\in V(H_{G,T'}).$$

如果用有向边将图 $H_{G,T'}$ 中所有满足上述不等式关系的顶点连接起来(不再重复添加 $E(H_{G,T'})$中已经存在的有向边)可得到图 $H_{G,T'}$ 的一个超图 $H^*_{G,T'}$. 该超图在定义上与含有 64 个顶点且步长为 1 循环图(记作 Ci^1_{64})等价. 因为图 Ci^1_{64} 与图 R_0 之间存在子图同构关系(如图 3 所示), 图 $H_{G,T'}$ 自然也与 R_0 形成子图同构关系(同构映射与 $H^*_{G,T'}$ 相同), 因此以函数 T'对问题进行划分满足框架对线程通信的要求. 本文按图 Ci^1_{64} 的形式将从核组织成了如图 3 所示的结构, 图中数字表示物理从核对应的逻辑线程编号. 每个逻辑线程(记为 t)对应一个前驱线程($t^-\equiv(t+1)\bmod 64$)以及一个后继线程($t^+\equiv(t-1)\bmod 64$), 图中的箭头表示了它们相互之间的通信关系.

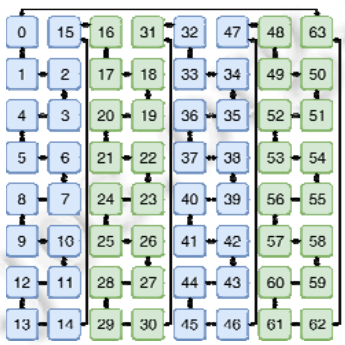


图 3 从核的逻辑组织结构

2.2 数据存储

本文在任务划分阶段将方程组的任务依赖图划分成了 64 个子区域(如图 4 所示), 每个子区域内部又按其顶点的 Z 轴坐标将它们切分为更小的问题单元(如图 5 虚线框所示), 从核依照顶点的 Z 轴坐标依次计算各单元内部方程的未知量. 为消除依赖关系, 每当得到一个未知量 $x[p]$, 从核要从其他相关的变量 $x[\delta]$中减去相应的数值 $A[p]x[q]$, 以此消除方程组内部与 $x[p]$有关的数据依赖关系. 为减少 $x[\delta]$引起的内存访问, 本文在从核 LDM 设置了用于临时存储 $x[\delta]$的缓冲空间, 从核 $x[\delta]$数据的分布与对应方程的划分方式相同. 从核在试图修改变量 $x[\delta]$时有两种可能的情况: 如果 $x[\delta]$位于当前从核缓冲区内, 从核则直接对其进行操作; 若 $x[\delta]$不在缓冲区内, 解决方案是通过寄存器通信向变量 $x[\delta]$所在的从核发送 $x[p]$的数据, 由后者负责 $A[p]x[q]$的计算并修改相应的 $x[\delta]$. 而这要求提供较为完善的通信规则, 使从核能够在寄存器通信上保持协调一致, 相关内容本文将在下一节介绍.

假设当前问题单元内部各顶点的 Z 轴坐标为 z, 单元内从核所修改的 $x[\delta]$, 其对应顶点的 Z 坐标一定不超过 $z+R(G)$, 否则, 在调度图内就会存在距离大于 $R(G)$而又相互依赖的顶点, 这与前文的定义是相矛盾的. 结合这个特点, 将从核变量 $x[\delta]$的缓冲空间分割成了$(R(G)+1)$个容量为 XB_Y 的缓冲区(如图 5 所示), 它们在逻辑上构成类似环状队列的数据结构, 分别用于暂存子区域内部 Z 轴坐标在 z 到 $z+R(G)$之间的 $x[\delta]$变量, 缓冲区的初值设置为方程组的输入右端项, 通过这种方式将变量 $x[\delta]$引起的全局访存全部转化成了针对 LDM 空间的访存操作, 从核只有在访问矩阵和输入(输出)右端项时会进行全局访存操作, 框架的全局内存访存量为理论最优.

2.3 未知量计算

为便于对从核寄存器通信进行管理, 本文将从核对问题单元(记为 P_z)处理分解成了以下基本操作, 图 6 在任务依赖图展示了它们的作用的方程及未知量依赖关系.

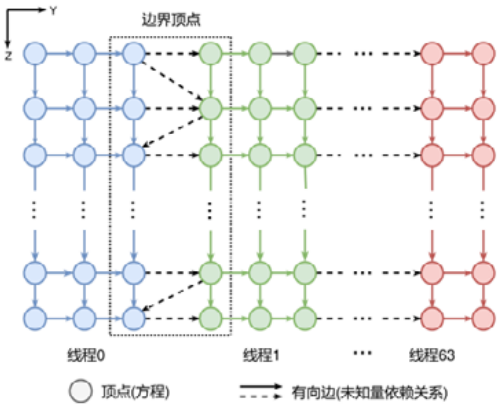


图 4 框架任务划分的示意图

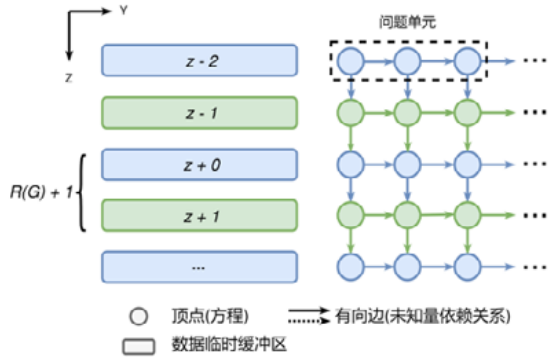


图 5 用于存储临时 $x[\delta]$的环状队列

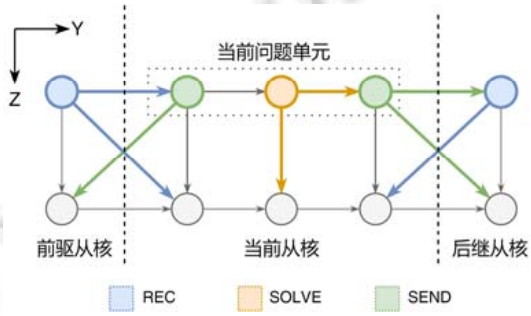


图 6 从核 REC、SEND 以及 SOLVE 操作处理的未知量依赖关系

REC(T,x): 当前线程(其编号记为 t)从前驱或后继线程对应的寄存器缓冲队列中读取所有 X 轴坐标为 x 的右端项 $x[q]$ (图中左侧(t^-)或右侧(t^+)的蓝色顶点), 然后消除由线程 t 负责的子区域内所有与 $x[q]$有关的未知量依赖关系(图中左侧(t^-)或右侧(t^+)的蓝色有向边), 若 $T=t^-$, 在 REC 阶段从核线程 0 不进行任何操作.

SEND(T,x): 假设 P_z 内部各方程的 Y 轴坐标取值在 y 到 $y+B_Y$ 之间, 当前线程在 SEND 阶段将以右端项的 Y 轴坐标的大小为序向前驱或后继线程发送单元内部 X 轴坐标为 x, 且 Y 轴坐标在以下区间内的右端项:

$$\left[y+B_Y-R(G),y+B_Y\right),T=t^+,$$

$$\left[y,y+R(G)\right),T=t^-.$$

若 $T=t^+$, 编号为 Y/B_Y 的从核线程在 SEND 阶段不执行任何操作.

SOLVE(p): 如果当前线程尚未求解方程 p, 根据矩阵的对角元素计算 $x[p]$ (图中的橙色顶点), 并消除由线程 t 负责的子区域内所有与 $x[p]$有关的未知量依赖关系(图中的橙色有向边).

在上述基本操作的框架下, 从核对问题单元的处理流程见算法 3a、3b 所示. 其中, 算法 3a 是子函数 ResolveDep 的定义. 指定单元内部的任意方程(e), ResolveDep 会根据方程组的 $R(G)$, 以递归的形式遍历并求解单元内部与方程 e 存在依赖关系的方程(d). 若方程 d 位于问题单元的边界, ResolveDep 在求解之前会首先通过 REC(t^-,x)操作的执行次数(R_x)来判别当前在前驱线程内部是否仍然存在与方程 d 有关的未知量依赖关系. 基于 ResolveDep 函数, 算法 3b 首先按方程的 X 轴坐标依次遍历单元内部的方程, 遍历过程中未知量数据的发送顺序与后继线程 REC(t^-,x)阶段的接收顺序保持一致. 单元的处理完成后从核会向其前驱发送必要的未知量信息, 发送顺序同样与前驱 REC(T^+,x)阶段保持一致. 在整个单元的遍历过程中从核 Send 阶段的寄存器通信总量为 $2R(G)\cdot X$.

算法 3a. 问题单元内部消除未知量依赖关系的步骤.

ResolveDep (P_z,R_x,e)

```
01 procedure ResolveDep (P_z,R_x,e)
02     foreach vertex d≠e in P_z with
03         ‖d,e‖≤R(G)∧F_XYZ(d)<F_XYZ(e) do
04             while R_x<min(d_x+R(G),X) do
05                 REC(t⁻,R_x); R_x←R_x+1;
06             end
07             ResolveDep(d); SOLVE(d);
08     end
09 end
```

算法 3b. 从核问题单元的遍历求解步骤.

```
SolveUnit(P_z)
01 R_x←0
02 for x=0 to X−1 do
03     foreach vertex e∈P_z with e_x=x do
04         ResolveDep(P_z,R_x,e); SOLVE(e);
05     end
06     SEND(t⁺,x)
07 end
08 if there are dependencies from t⁺ to t in G then
09     for x=0 to X−1 do
10         SEND(t⁻,x); REC(t⁺,x);
11     end
12 end
```

2.4 数据预处理

若方程组不变, 任务划分后单元内部各未知量依赖关系的计算顺序是确定的, 因此从核对稀疏矩阵元素的访存顺序也是固定的. 求解器可据此对它们进行重排来给从核的 DMA 访问提供便利. 预处理阶段, 记录非零元的访问顺序, 得到一个索引数组 *s*, 之后再进行相应的重排序操作. 预处理阶段以访存操作为主, 其中前一阶段访存为数组 *I*,*J* 及 *s*, 后一阶段为数组 *s* 和 *A*, 预处理操作期间的访存总量接近两次主核 SpTRSV 的规模, 因此本文以两次主核 SpTRSV 的时间作为预处理步骤的近似开销, 并由此估算预处理对整体加速性能的影响.

若将主核和从核 SpTRSV 的时间分别记为 T_0 和 T_1, 设 n 为应用迭代过程中从核 SpTRSV 函数的调用总次数, 框架 SpTRSV 的整体加速比(θ)满足如下等式:

$$\theta=\frac{nT_0}{2T_0+nT_1}=\frac{n\phi}{2\phi+n},$$

其中, $\phi=T_0/T_1$ 是单次从核算法的加速比. 结合实测得到的性能数据($\phi\geqslant 24$, 详见实验部分), 只要从核调用次数足够多($n\geqslant 20$), 预处理开销均摊后, SpTRSV 仍可实现较为可观的整体加速效果(如图 7 所示).

3 实验测试

3.1 实验设置

本文选取包括 7 点和 13 点[2,6,24]在内的一些比较典型模板格式, 自动生成不同规模的问题实例对 SpTRSV 求解器进行测试, 图 8 是测试问题经 ILU 分解后未知量依赖关系的示意图. 实验的对象为主核及从核版本的

SpTRSV, 测试问题的规模从 128^3 逐步增长至 320^3, 执行时间取 100 次测量的均值. 本文以从核访存带宽和主从核加速比作为框架性能的主要评估依据. 访存带宽的计算以稀疏矩阵本身以及问题输入输出右端项之和作为基准访存量. 峰值访存带宽取 Stream 测试[25]的峰值[8], 具体约为 26 GB/s.

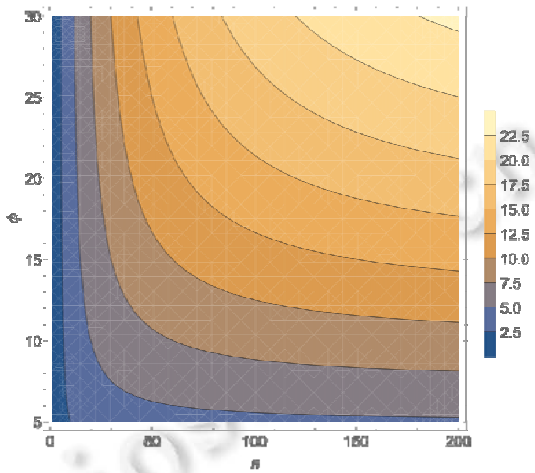


图 7 SpTRSV 调用次数与整体加速效果的示意图

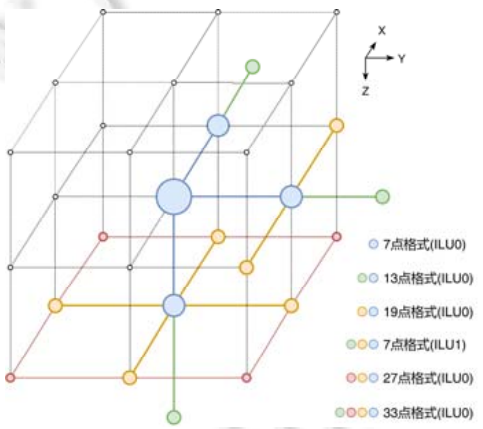


图 8 测试问题经 ILU 分解后依赖关系的示意图

3.2 实验结果

经测试, 本文提出的 SpTRSV 求解器算法的访存带宽介于 20.41–24.57 GB/s 之间, 带宽随着问题规模的扩大而逐渐增长, 表 1 列举了详细测试带宽数据. 测试问题平均访存带宽利用率约为 88.1%, 对于 33 点格式问题, 框架实测访存带宽可达到带宽峰值的 94.5%. 从核版本相比主核实现了 24.2–29.1 倍的加速, 图 9 和图 10 是测试问题主从核加速效果的对比. 如果问题的类型固定不变, 即便忽略算法稀疏矩阵的访存, 从核通信的数据量$(64{\cdot}2R(G)XZ)$相比问题右端项的访存量$(2XYZ)$, 其占比$(64R(G){\cdot}Y^{-1})$随着问题规模的增长也越来越少, 故算法实际的性能越来越好. 比如表 1 中第 1 行 7 点格式(ILU0)类型的问题, 在问题规模为 128^3 时, 框架的访存带宽为 20.41 GB/s, 而当规模变为 320^3 时框架的访存带宽上升到了 22.92 GB/s, 相比有约 12%左右的性能提升. 在其他问题的测试中, 本文也可观察到类似的规律.

表 1 性能测试的访存带宽数据(GB/s)

问题类型	$R(G)$	128^3	192^3	256^3	320^3
7 点格式(ILU0)	1	20.41	21.80	22.60	22.92
13 点格式(ILU0)	2	21.17	22.37	22.81	23.03
19 点格式(ILU0)	2	21.23	22.73	23.41	23.50
7 点格式(ILU1)	2	21.87	23.41	23.87	23.93
27 点格式(ILU0)	2	22.55	23.78	24.24	24.43
33 点格式(ILU0)	2	22.82	23.94	24.51	24.57

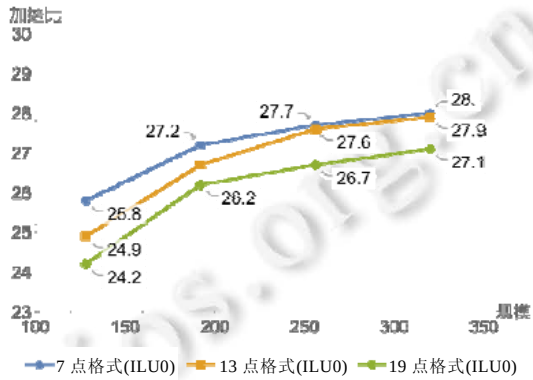


图 9 7 点(ILU0)、13 点(ILU0)以及 19 点(ILU0)类型问题的加速效果对比

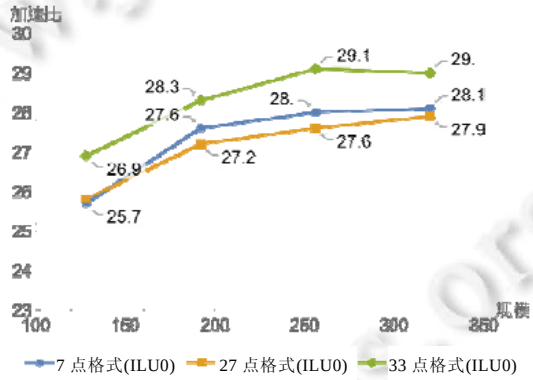


图 10 7 点(ILU1)、27 点(ILU0)以及 33 点(ILU0)类型问题的加速效果对比

3.3 对比分析

针对测试问题, 目前性能最好的 GridSpTRSV 的平均带宽利用率为 79.7%, 而本文的带宽利用率可达 88.1%, 在访存带宽上有约 1.5 GB/s 的提升. GridSpTRSV 的任务划分主要针对 7 点和 27 点格式的问题而设计, 如果方程组的任务依赖图在非对角方向存在有向边, 在该工作的数据路由体系下难以做到高效的线程通信. 本文的方法虽然会耗费一定的 LDM 空间, 但通信不受数据路由问题的制约, 对任务依赖图没有特殊的要求.

针对同一方程组问题, GridSpTRSV 和本文在求解阶段中间结果的数据规模均为 $O(R(G)\cdot XY)$, 所以可供调度的从核空间越多, 能够处理的问题规模就越大. 本文将 $x[\delta]$平均分配到了各从核的 LDM 空间当中, 理论上这种分配方式可调动核组的全部 LDM 空间. 而在 GridSpTRSV 中只有一部分从核负责存储 $x[\delta]$的中间结果, 故框架能处理的问题的规模就比较有限. 例如, 就 33 点格式类型的网格而言, 该工作要求网格在 X-Y 轴方向满足 $XY\leqslant 200^2$ 的约束条件, 而本文可将该约束条件放宽至 $XY\leqslant 400^2$. 而且对于二维的方程组问题, 本文可在 Y-Z 方向(以 F_{1XY} 作为线性化对应关系)进行任务划分, 从而打破框架在 X-Y 轴方向上的限制. 本文的方法已经被应用到了二维磁场重联问题的数值模拟研究[26]当中, 取得了较为显著的加速效果(测试问题的规模为 1920×1920, 使用 16 个进程计算, 测试过程中总计 1 224 次 SpTRSV 调用, 应用的整体加速比为 11.2×).

4 结 论

本文面向与结构化网格相关的稀疏线性方程组问题，基于 SW26010 处理器平台设计了一套并行 SpTRSV 求解. 本文利用结构化网格提供的几何信息，通过特定的任务划分以及线程排布方式，在不采用数据路由的前提下解决了从核线程未知量数据的传递问题. 在此基础上线程通信得以大幅简化，实测数据也说明通信层面的优化的确能够带来比较明显的性能提升. 在设计上本文的工作只对从核的存储空间有一定的要求，线程通信的方式也比较单一，因此适合于类似 GPU 这种在各计算单元(流处理器)间缺少有效通信机制的平台. 然而 GPU 与 SW26010 处理器的架构差异明显，SpTRSV 在任务划分和算法设计层面需要额外考虑 GPU 平台在硬件调度及访存等方面独有的特性，这些都是下一步工作中需要深入研究和探索的问题.

References:

[1] The Top500 list. 2018. https://www.top500.org

[2] Ao Y, Yang C, Wang X, *et al*. 26pflops stencil computations for atmospheric modeling on sunway taihulight. In: Proc. of the 2017 IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS). 2017. 535−544.

[3] Lin J, Wen MH, Meng DL, *et al*. Optimizing preconditioned conjugate gradient on TaihuLight for OpenFOAM. In: Proc. of the IEEE/ACM Int'l Symp. on Cluster, Cloud and Grid Computing (CCGRID). 2018.

[4] Liu FF, Chen DK, Chao Y, *et al*. Research on heterogeneous many-core fully-implicit solver for MHD dynamical equations. Journal on Numerical Methods and Computer Applications, 2019.

[5] Fu HH, Chen BW, Zhang WQ, *et al*. Extreme-scale earthquake simulations on Sunway TaihuLight. CCF Trans. on High Performance Computing, 2019.

[6] Nagel GR. Solving the Generalized Poisson Equation Using the Finite-difference Method. 2012. https://www.researchgate.net/publication/228411289_Solving_the_Generalized_Poisson_Equation_Using_the_Finite-Difference_Method_FDM

[7] Li R, Saad Y. GPU-accelerated preconditioned iterative linear solvers. J. Supercomputing, 2013, 63(2): 443−466.

[8] Wang X, Xu P, Xue W, *et al*. A fast sparse triangular solver for structured-grid problems on sunway many-core processor SW26010. In: Proc. of the 47th Int'l Conf. on Parallel Processing. New York: ACM, 2018. 53:1−53:11.

[9] Wang X, Xue W, Liu W, *et al*. swSpTRSV: A fast sparse triangular solver with sparse level tile layout on sunway architecture. In: Proc. of the 23rd ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming, PpoPP 2018. ACM Press, 2018.

[10] Barrett R, Berry M, Chan TF, *et al*. Templates for the solutions of linear systems: Building blocks for iterative methods. SIAM, 1994.

[11] Chow E. A survey of incomplete factorization preconditioners. 2003. https://faculty.cc.gatech.edu/~echow/pubs/pims_talk.pdf

[12] Grossmann C, Roos H. Numerical Treatment of Partial Differential Equations. Springer, 2005.

[13] Park J, Smolyansky M, Sundaram N, Dubey P. Sparsifying synchronization for high-performance shared-memory sparse triangular solver. In: Lecture Notes in Computer Science. Springer Int'l Publishing, 2014. 124−140.

[14] Anderson E, Saad Y. Solving sparse triangular linear systems on parallel computers. Int. J. High Speed Computing, 1989, 1(1): 73−95.

[15] Saltz JH. Aggregation methods for solving sparse triangular systems on multiprocessors. SIAM Journal on Scientific and Statistical Computing, 1990, 11(1): 123−144.

[16] https://max.book118.com/html/2019/0630/8053075117002032.shtm

[17] Xu S, Xu Y, Wei X, *et al*. Taming the monster: Overcoming program optimization challenges on SW26010 through precise performance modeling. In: Proc. of the IEEE Int'l Parallel and Distributed Processing Symp. 2018.

[18] Wu C, Xiao S. To GPU synchronize or not GPU synchronize? In: Proc. of the 2010 IEEE Int'l Symp. on Circuits and Systems. IEEE, 2010.

[19] Liu W, Li A, Hogg J, *et al*. A synchronization-free algorithm for parallel sparse triangular solves. In: Proc. of the Euro-Par 2016: Parallel Processing. Springer Int'l Publishing, 2016. 617−630.

[20] Liu W, Li A, Hogg JD, *et al*. Fast synchronization-free algorithms for parallel sparse triangular solves with multiple right-hand sides. Concurrency and Computation: Practice and Experience, 2017, 29(21): 4244.

[21] Wolf MM, Heroux MA, Boman EG. Factors impacting performance of multithreaded sparse triangular solve. In: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2011. 32−44.

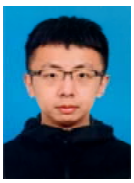
[22] Picciau A, Inggs GE, Wickerson J, *et al*. Balancing locality and concurrency: Solving sparse triangular systems on GPUs. In: Proc. of the 23rd IEEE Int'l Conf. on High Performance Computing (HiPC). 2016.

[23] Battista D, Eades PG, Tamassia R, *et al*. Graph Drawing: Algorithms for the Visualization of Graphs. Prentice-Hall, 1989.

[24] Ao Y, Yang C, Liu F, *et al*. Performance optimization of the HPCG benchmark on the sunway taihulight supercomputer. ACM Trans. on Archit. Code Optimization, 2018, 15(1): 11:1–11:20.

[25] McCalpin JD. Stream: Sustainable memory bandwidth in high performance computers (1991-2007). Technical Report, University of Virginia, http://www.cs.virginia.edu/stream/

[26] Liu FF, Chen DK, Chao Y, *et al*. Research on heterogeneous many-core fully implicit solver for MHD dynamical equations. Journal on Numerical Methods and Computer Applications, 2019.



陈道琨(1994－), 男, 博士生, 主要研究领域为异构架构并行算法, 稀疏矩阵相关算法设计与优化.

刘芳芳(1982－), 女, 博士, 正高级工程师, CCF 专业会员, 主要研究领域为高性能扩展数学库, 稀疏迭代解法器, 异构众核并行.

杨超(1979－), 男, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为高性能计算, 科学与工程计算.