西北工業大學

本科毕业设计（论文）

题　目　　**基于华为鲲鹏 920 的 SpTRSV 算法优化**

专业名称　计算机科学与技术

学生姓名　严愉程

指导教师　王云岚

完成时间　2021 年 6 月 1 日

毕业 设计 论文 任务书

一、题目

题目这种东西随便起一个就行了

二、研究主要内容

用 word 做好任务书,打印成 pdf.

三、主要技术指标

50 页

四、进度和要求

第 1 周至第 2 周:完成第一章;
第 3 周至第 4 周:完成第二章;
第 5 周至第 6 周:完成第三章;
第 7 周至第 8 周:完成第四章;
第 9 周至第 10 周:完成第五章;
第 11 周至第 15 周:安心养老,等待毕业;

五、主要参考书及参考资料

[1] Shen S. LaTeX-Template-For-NPU-Thesis[Z]. 2016.05.

学生学号 _____ 学生姓名 _____

指导教师 _____ 系 主 任 _____

摘　要

稀疏下三角矩阵求解器，是数值计算中的一个重要的方法，在科学计算中有着广泛的应用。由于其离散的数据访存、任务之间存在着很强的依赖、需要细粒度的同步、任务之间负载不均衡等特点，在现代多核系统中，如何提升 SpTRSV 算法的并行度与扩展性，仍然是一个挑战性的课题。

华为鲲鹏 920 处理器基于 ARMv8 架构，最多拥有 64 个核心，支持 ARM NEON 128bits 浮点运算。华为鲲鹏系统支持 NUMA 内存架构，实现最多 4 个 920 互联，最高支持 256 个计算核心。

本文基于华为鲲鹏 920 芯片，进行稀疏下三角矩阵求解算法的设计与优化。

主要工作包括:

1. 设计并实现了一套高效的稀疏下三角矩阵求解算法。
2. 针对华为鲲鹏 920 处理器众核体系结构以及 ARMv8 指令集的特点进行了优化。
3. 采用 ARM NEON 指令，对向量乘法部分进行了 SIMD 的优化。
4. 结合处理器 NUMA 架构的特性，通过分配矩阵数据的存储，充分利用内存带宽。

关键词： 稀疏下三角矩阵求解器, 鲲鹏 920, 众核优化, 并行算法

Abstract

The sparse triangular solve kernel, SpTRSV, is an important building block for a number of numerical linear algebra routines. It is commonly considered challenging to parallelize and scale even on a moderate number of cores. This challenge is due to the fact that triangular solver typically has frequent discrete memory access, load embalances between the tasks.

Huawei Kunpeng920 is an ARMv8-based server CPU with up to 64 cores, supporting ARM NEON 128-bit SIMD instruction set.

The main work of this article includes:

1. Design and implement an efficient sparse matrix triangular solve algorithm.
2. Optizing for Huawei Kunpeng920 multi-core processor and features of ARMv8 instruction set.
3. Utilizing ARM NEON 128bits SIMD instruction set to speed up the vector multiplication which is part of SpTRSV algorithm.
4. By taking NUMA feature into consideration and allocating matrix data properly, achieve better use of memory bandwidth.

Key Words: SpTRSV, Kunpeng920, Multi-core Optimization, Parallel Algorithm

目 录

第 一 章　绪论

1.1 课题背景

稀疏下三角矩阵求解 (Sparse Triangular Solve, SpTRSV) 并行算法是很多数值计算方法的重要组成部分，比如用直接法求解稀疏矩阵的线性方程组 [1]，以及最小二乘法的求解 [2]，是现代科学计算中一个广泛使用的计算核心，在数值模拟计算中，通常会使用迭代法或直接法求解大规模稀疏线性方程组，而 SpTRSV 的效率直接影响了线性方程组的求解效率，故提高 SpTRSV 算法的性能至关重要。

相比于稠密下三角举证解法器 [3] 或者其他稀疏矩阵计算方法，例如稀疏矩阵转置 [4]，稀疏矩阵向量乘法 [5][6]，和稀疏矩阵乘法 [7]，SpTRSV 频繁且离散地数据访存、任务之间存在着很强的依赖、需要细粒度的同步、任务之间负载不均衡等特点是并行优化更加难以进行。

SpTRSV 算法主要是从方程组 $Lx = b$ 中求解未知数向量 x，其中 L 是下三角矩阵，且对角线上的元素都不为 0。因为这意味这要计算其中的一个未知数 x_k，需要先计算出其前置未知数 x_0, ..., x_{k-1} 中的一个子集。这些任务间的依赖关系，就使得 SpTRSV 的计算转换成了一个任务依赖图的计算（Task Dependent Graph，TDG），而且这个 TDG 是一张有方无环图 (Directed Acyclic Graph)。

华为作为鲲鹏计算产业的成员，聚焦于发展华为鲲鹏 + 昇腾双引擎芯片族，通过“硬件开放、软件开源、使能合作伙伴”来推动计算产业的发展。华为鲲鹏 920 处理器兼容了 ARMv8 架构。最多拥有 64 核心。其浮点及 SIMD 运算单元支持每拍 2 条 ARM Neon 128bits 浮点运算。华为鲲鹏系统支持 NUMA 内存架构，实现最多 4 个鲲鹏 920 互联和最高 256 个物理核的 NUMA 架构。

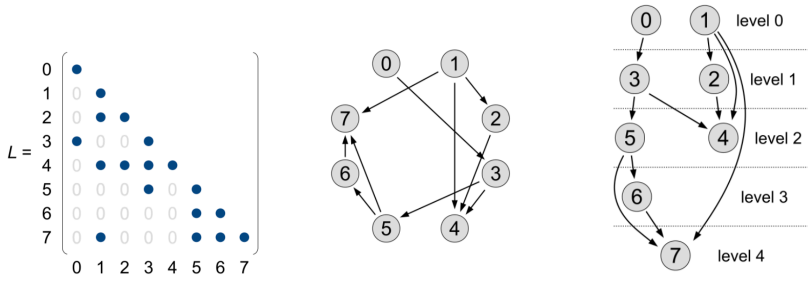
稀疏矩阵运算在科学计算、数据分析、机器学习等应用中十分常见，而稀疏矩阵频繁且离散地数据访存、任务之间存在着很强的依赖、需要细粒度的同步、任务之间负载不均衡等特点又给代码优化带来了巨大的挑战。

1.2 研究现状

有研究者 Saad[8]，以及 Saltz[9] 根据 SpTRSV 算法是一张任务依赖图1-1(b)的特点，提出了基于 level-sets 的方法，在预处理阶段构建一个任务依赖图，如图1-1(c)，之后就可以在 level 内部进行并行，在 level 之间设置同步屏障，一个 level 执行完了再执行下一个 level。

随后有研究人员在体系结构层面上针对 CPU[10][11][12][13] 以及 GPU[14][15][16] 进行了优化。

基于 level-sets 的算法，存在着两个缺陷：第一，虽然可以在每个 level 上进行并行取得良好的并行度，但是需要在预处理阶段构建一个 TAG 图，计算每个任务的 level，以及处理任务之间任务负载均衡的问题。然而，在预处理阶段往往会花费大量的时候，写出一个有良好扩展性的并行预处理算法同



(a) 下三角矩阵 (b) 任务依赖图 (c) level-sets

样具有挑战性，往往可能会导致预处理的时间远远大于并行计算任务图的时间，甚至在计算任务图上获得的加速甚至不能抵消预处理阶段产生的计算时间。第二，基于 level-sets 的的算法需要在每个 level 之间都要使用一个屏障确保进行同步，等待该 level 内的任务都执行完了再继续下个一个任务，在负载不均衡的情况下，这会产生大量的等待时间，随着核心数量的上升该同步方式的开销也会随之上升。

面对以上两个 level-sets 的缺陷，有研究人员提出了以下的优化算法。

Jongsoo Park[11] 在基于 level-sets 算法进行了一些优化。他发现传统的基于 level-sets 的算法所使用的屏障式的同步方式会产生大量的开销。因此作者提出了一种 peer-to-peer 的同步方式。线程在任务执行完了之后不再是等待在屏障处，等待其他线程执行到该屏障处，而是会判断下一个任务的前置需求是否被完成，如果已经完成了，那么就继续运行下去。相比于基于屏障的同步方式，peer-to-peer 的同步方式具有更好的扩展性。

在减少同步所需消耗方面，作者还发现，在 SpTRSV 算法的任务依赖图中，大部分的依赖都是多余的，作者期望通过一步预处理操作来删除这些多余依赖。例如在 2→3→5 和 2→5 这种边存在的情况下，删除了 2→5 这条边。另一方面，作者发现在这些多余依赖当中，大部分都是两跳的（比如上述的 2→5 这条边），少部分是三跳或者三跳以上。为了减少预处理的计算量，作者选择用粗略但快速的算法，只删除两跳的依赖边，而不是删除所有的多余依赖。减少了大约 90% 的多余依赖，具体效果如1-1。该算法运行在 12 核的 Xeon 处理器上，相比于传统的基于 level-sets 以及屏障式同步的算法获得了至少 1.6 倍的加速比。

同时作者也做了负载均衡的优化，根据每行非零元个数的多少来进行任务的合并，组合成一定数量的 super-task，使得每个 super-task 有相似的计算量。

刘伟峰 [17, 18] 从减少算法预处理和减少算法同步所需时间的角度出发，提出了无需同步的（synchronization-free）SpTRSV 算法。基于传统 level-set 的方法，随着矩阵大小的增加，其在预处理阶段所需的消耗会剧烈增加，且同步所需的时间在计算总时间中的占比也会大量增加。他的算法在预处理阶段只计算了每个 entry 的 in-degree 数组（用来记录每一个节点需要有多少入

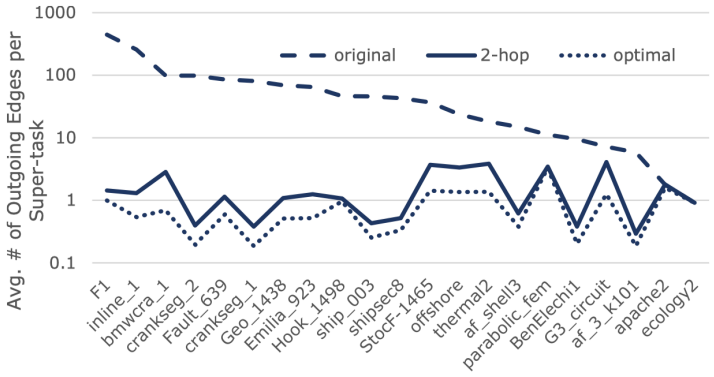


图 1-1　每个 super-task 的出度有了显著的减少

度)，而不是构造一个依赖图，大幅减少了预处理时间，而且该预处理方法能够轻易地使用多核来进行并行加速。对于每个 warp 使用自旋锁，不断判断前置依赖是否都被满足来决定该结点是否开始计算。当一行计算完成后，使用原子操作来“通知”后继节点，减少了同步的时间，该算法在 GPU TitanX 上比 Nvidia 提供的算法快 2-3 倍。

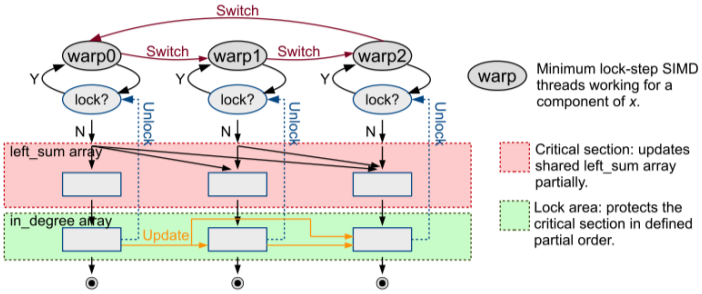


图 1-2　synchronization-free SpTRSV 算法的示意图

倪鸿、刘鑫 [19] 根据国产异构众核处理器 SW26010 体系结构的特点，针对非结构网络计算，提出了一种基于流水线串行-局部并行思想的通用众核 SpTRSV 优化方法。首先，为了减少预处理时间，根据核心的个数，将向量 x 平均划分为多个向量块。每个向量块 x 交给一个从核进行处理。每个从核的操作流程为：(1) 等待接受来自依赖向量块的数据，直到满足进入 (2)。(2) 进行 x 的计算。(3) 发送计算得到的 x 给需要的核心。其次，为了解决 SW26010 寄存器通讯的局限性，作者还搭建了众核通信架构。同时还有数据压缩、LDM 缓冲、访存掩盖以及数据压缩等优化。

1.3 研究内容

1.4 本文构成

第 二 章　相关理论与技术介绍

2.1 稀疏矩阵的压缩方式

存储矩阵的一般方法是采用二维数组，其优点是可以随机地访问每一个元素，因而能够容易实现矩阵的各种运算。对于稀疏矩阵，它通常具有很大的维度，有时甚大到整个矩阵（零元素）占用了绝大部分内存采用二维数组的存储方法既浪费大量的存储单元来存放零元素，又要在运算中浪费大量的时间来进行零元素的无效运算。因此必须考虑对稀疏矩阵进行压缩存储（只存储非零元素)。

2.1.1 COO Format

最简单的稀疏矩阵的存储格式称为 coordinate(COO) format，这个形式只保留哪些非 0 的值，分别使用三个数组：val, rolIdx, colIdx 来对应存储数值、行号以及列号，这三个数组的大小都为矩阵非零元的个数 NNZ。举例说明，

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 2 & 4 & 0 & 6 & 0 \\ 3 & 0 & 5 & 0 & 0 \\ 4 & 7 & 0 & 8 & 9 \end{pmatrix}$$

该矩阵在 COO 存储格式下，三个数组分别为：

COO Format

```
val = {1, 2, 4, 6, 3, 5, 4, 7, 8, 9}
rolIdx = {0, 1, 1, 1, 2, 2, 3, 3, 3, 3}
colIdx = {0, 0, 1, 3, 0, 2, 0, 1, 3, 4}
```

2.1.2 CSR Format

CSR（Compressed Sparse Row）是一种按行压缩的矩阵存储形式。分别使用三个数组：val, rowPtr, colIdx 来对应存储数值、行指针、以及列号。其中 val 和 colIdx 的数组大小为非零元的个数 NNZ，而 rowPtr 数组的大小一般为 m+1，其中 m 为矩阵行的个数。例如图2-1。

2.1.3 CSC Format

CSC（Compressed Sparse Column）与 CSR 相似，是一种按列压缩的矩阵存储格式。分别使用是那个数组：val, colPtr, rowIdx 来对应存储数值、列指针、以及行号。其中 val 和 rowIdx 的数组大小为非零元的个数 NNZ，而 colPtr 数组的大小一般为 n+1，其中 n 为矩阵列的个数。例如图2-2

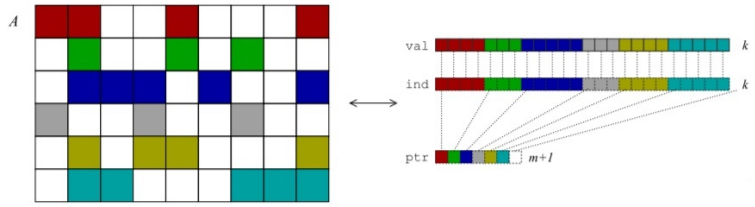


图 2-1　CSR 矩阵存储格式

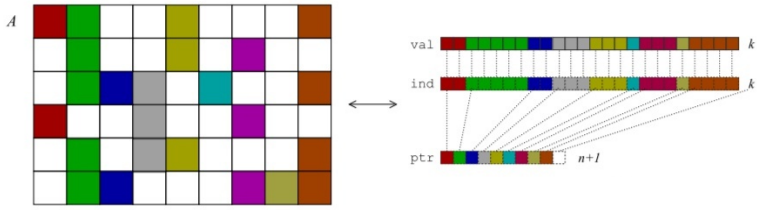


图 2-2　CSC 矩阵存储格式

2.2 现存 SpTRSV 算法

2.2.1 基于 CSR 格式的 SpTRSV 串行算法

Algorithm 1: CSR Based Serial SpTRSV

```
1 MALLOC(*left_sum, n) ;
2 MEMSET(*left_sum, 0) ;
3 for i = 0; i < n; i++ do
4 |   for j = row_ptr[i] ; j <row_ptr[i + 1] - 1 ; j++ do
5 |   |   left_sum[i] ⟵ left_sum[i] + val[j] x x[col_idx[j]] ;
6 |   end
7 |   x[i] ⟵ (b[i] - left_sum[i]) / val[col_ptr[i]] ;
8 end
```

基于 CSR 格式的 SpTRSV 串行算法第一个循环从矩阵的第 0 行开始遍历这个矩阵。在第二个循环进行向量乘法的操作，求出 left_sum[i]。之后再求出未知数 x[i]。

2.2.2 基于 CSC 格式的 SpTRSV 串行算法

该算法的第一个循环从矩阵的第 0 行开始遍历这个矩阵。首先求出该算该行未知数 $x[i]$，接着遍历该列的上的非零元，对该列非零元所对应的 $left\_sum$ 进行更新，$left\_sum[rowIdx[j]] += val[j]xx[i]$。

分析基于 CSR 压缩格式 SpTRSV 算法以及基于 CSC 压缩格式的 SpTRSV 算法可以得出，两者在访存模式上存在着一定的差异。基于 CSR 格式的算法需要进行频繁离散读取未知数向量 x[col_idx[j]]，相反基于 CSC 格式的 SpTRSV 算法只需要读取当前行号 i 所对应的 x[i]。在写数据方面，基于

Algorithm 2: CSC Based Serial SpTRSV

```
1 MALLOC(*left_sum, n) ;
2 MEMSET(*left_sum, 0) ;
3 for i = 0; i < n; i++ do
4 |   x[i] ⟵ (b[i] - left_sum[i]) / val[col_ptr[i]] ;
5 |   for j = col_ptr[i] ; j <col_ptr[i + 1] - 1 ; j++ do
6 |   |   left_sum[rowIdx[j]] ⟵ left_sum[row_idx[j]] + val[j] x x[i] ;
7 |   end
8 end
```

CSR 的并行算法需要集中的写 left_sum[i] 的数据，而基于 CSC 的并行算法则是离散的写 left_sum[rowIdx[j]]，在多核运算的条件下，离散的写比集中的写同一位置的数据更容易进行并行化。

2.2.3 基于 level-sets 的 SpTRSV 并行算法

基于 level-sets 的 SpTRSV 并行算法，主要分为两个阶段：预处理阶段和计算阶段。算法在预处理阶段通过并行的 BFS 算法构建一个任务依赖图，这往往需要大量的计算开销。

2.3 鲲鹏 920 的内存层次结构

2.4 ARM NEON 指令

2.5 NUMA 架构

2.6 本章小结

第 三 章　SpTRSV 算法的设计与实现

3.1 总结总结

3.2 总结总结

3.3 总结总结

参考文献

[1] Davis T A. Direct methods for sparse linear systems[M]. SIAM, 2006.

[2] Saad Y. Iterative methods for sparse linear systems[M]. SIAM, 2003.

[3] Hogg J D. A fast dense triangular solve in cuda[J]. SIAM Journal on Scientific Computing, 2013, 35(3):C303-C322.

[4] Wang H, Liu W, Hou K, et al. Parallel transposition of sparse data structures[C]// Proceedings of the 2016 International Conference on Supercomputing. 2016: 1-13.

[5] Liu W, Vinter B. Csr5: An efficient storage format for cross-platform sparse matrix-vector multiplication[C]//Proceedings of the 29th ACM on International Conference on Supercomputing. 2015: 339-350.

[6] Liu W, Vinter B. Speculative segmented sum for sparse matrix-vector multiplication on heterogeneous processors[J]. Parallel Computing, 2015, 49:179-193.

[7] Liu W, Vinter B. A framework for general sparse matrix–matrix multiplication on gpus and heterogeneous processors[J]. Journal of Parallel and Distributed Computing, 2015, 85:47-61.

[8] Anderson E, Saad Y. Solving sparse triangular linear systems on parallel computers [J]. International Journal of High Speed Computing, 1989, 1(01):73-95.

[9] Saltz J H. Aggregation methods for solving sparse triangular systems on multiprocessors[J]. SIAM journal on scientific and statistical computing, 1990, 11(1):123-144.

[10] Kabir H, Booth J D, Aupy G, et al. Sts-k: a multilevel sparse triangular solution scheme for numa multicores[C]//Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. 2015: 1-11.

[11] Park J, Smelyanskiy M, Sundaram N, et al. Sparsifying synchronization for high-performance shared-memory sparse triangular solver[C]//International Supercomputing Conference. Springer, 2014: 124-140.

[12] Schreiber T, R. Vectorizing the conjugate gradient method.[J]. Proceedings of the Symposium on CYBER 205 Applications, 1982.

[13] Wolf M M, Heroux M A, Boman E G. Factors impacting performance of multithreaded sparse triangular solve[C]//International Conference on High Performance Computing for Computational Science. Springer, 2010: 32-44.

[14] Li R, Saad Y. Gpu-accelerated preconditioned iterative linear solvers[J]. The Journal of Supercomputing, 2013, 63(2):443-466.

[15] Naumov M. Parallel solution of sparse triangular linear systems in the preconditioned iterative methods on the gpu[J]. NVIDIA Corp., Westford, MA, USA, Tech. Rep. NVR-2011, 2011, 1.

[16] Suchoski B, Severn C, Shantharam M, et al. Adapting sparse triangular solution to gpus[C]//2012 41st International Conference on Parallel Processing Workshops. IEEE, 2012: 140-148.

[17] Liu W, Li A, Hogg J, et al. A synchronization free algorithm for parallel sparse triangular solves[M]//Euro Par2016: Parallel Processing: volume 9833. ChamSpringer International Publishing, 2016: 617-630.

[18] Liu W, Li A, Hogg J D, et al. Fast synchronization-free algorithms for parallel sparse triangular solves with multiple right-hand sides: Fast synchronization-free algorithms for parallel sparse triangular solves[J/OL]. Concurrency and Computation: Practice and Experience, 2017, 29(21):e4244. DOI: 10.1002/cpe.4244.

[19] 倪鸿刘鑫. 非结构网格下稀疏下三角方程求解器众核优化技术研究[M]. 北京大学, 2019.

附　录

这是一份附录，请放置一些独立的证明、源代码、或其他辅助资料。

致　谢

感谢党和国家

感谢老师对我的指导和帮助

感谢中科院软件研究所和华为提供的计算资源

毕业设计小结

毕业论文是大学四年的最后一份大作业...