·体系结构与软件技术·　　文章编号：1000-3428(2022)07-0199-07　　文献标志码：A　　中图分类号：TP391

FPGA架构上面向稀疏矩阵求解的静态调度算法

王晞阳[1]，陈继林[2]，李　猛[1]，刘首文[3]

（1.国家超级计算无锡中心，江苏 无锡 214072；2.中国电力科学研究院有限公司，北京 100192；
3.国网湖北省电力有限公司，武汉 430070）

摘　要：在电力系统仿真中，大型稀疏矩阵的求解会消耗大量存储和计算资源，未有效利用矩阵的稀疏性将导致存储空间浪费以及计算效率低下的问题。当前关于稀疏矩阵求解算法的研究主要针对众核加速硬件，聚焦于挖掘层次集合的并行度以提升算法的并行效率，而在众核处理器架构上频繁地进行缓存判断及细粒度访问可能导致潜在的性能问题。针对基于现场可编程门阵列（FPGA）的下三角稀疏矩阵求解问题，在吴志勇等设计的FPGA稀疏矩阵求解器硬件结构的基础上，提出一种静态调度求解算法。通过对稀疏矩阵进行预处理，设计数据分布和指令排布流程，将下三角稀疏矩阵的求解过程静态映射到多个FPGA片上的处理单元，以实现下三角稀疏矩阵在FPGA上的并行高速求解。将串行算法中所有的隐式并行关系排布到缓冲中，使得所有计算单元都能实现计算、访存和单元间通信的高效并行，从而最大限度地利用FPGA的硬件资源。典型算例上的测试结果表明，相较传统的CPU/GPU求解算法，该算法能够实现5~10倍的加速效果。

关键词：下三角稀疏矩阵；静态调度算法；数据分布；指令排布；静态映射

开放科学（资源服务）标志码（OSID）：

中文引用格式：王晞阳，陈继林，李猛，等.FPGA架构上面向稀疏矩阵求解的静态调度算法[J].计算机工程，2022，48(7)：199-205，213.

英文引用格式：WANG X Y，CHEN J L，LI M，et al.Static scheduling algorithm for solving sparse matrixes on FPGA architecture[J].Computer Engineering，2022，48(7)：199-205，213.

Static Scheduling Algorithm for Solving Sparse Matrixes on FPGA Architecture

WANG Xiyang[1]，CHEN Jilin[2]，LI Meng[1]，LIU Shouwen[3]

（1.National Supercomputing Center in Wuxi，Wuxi，Jiangsu 214072，China；
2.China Electronic Power Research Institute，Beijing 100192，China；
3.State Grid Hubei Electric Power Co.，Ltd.，Wuhan 430070，China）

【Abstract】In power system simulations，the solution of large-scale sparse matrixes will consume a lot of storage and computing resources，and a failure to effectively utilize the sparsity of the matrix will lead to a waste of storage space and low computing efficiency.Current studies on algorithms that solve sparse matrixes focus mainly on the multi-core acceleration hardware，with emphasis on mining the parallelism of hierarchical sets to improve the parallel efficiency of the algorithm.Frequent cache judgment and fine-grained access on the multi-core processor architecture may lead to potential performance problems.To solve the lower triangular sparse matrix based on Field Programmable Gate Array（FPGA），a static scheduling algorithm is proposed based on the hardware structure of an FPGA sparse matrix solver designed by WU Zhiyong et al.By preprocessing the sparse matrix，designing the data distribution and instruction layout process，the solution process of the lower triangular sparse matrix is statically mapped to multiple Process Elements（PEs）on an FPGA chip to realize the parallel high-speed solution of the lower triangular sparse matrix on the FPGA.By arranging all implicit parallel relations in the serial algorithm into the buffer，all computing units can achieve efficient parallel computing，memory access，and inter unit communication，enabling users to maximize the use of FPGA hardware resources.The test results obtained using typical examples show that compared with traditional CPU/GPU algorithms，the proposed algorithm can achieve an acceleration that is 5—10 times greater than existing methods.

【Key words】sparse matrix of the lower triangular；static scheduling algorithm；data distribution；instruction layout；static mapping

DOI：10. 19678/j. issn. 1000-3428. 0062198

基金项目：国家电网公司科技项目“适应于电力系统应用的高性能计算技术研究与开发”（XT71-19-022）。

作者简介：王晞阳（1976—），男，高级工程师、硕士，主研方向为并行算法与体系结构；陈继林，高级工程师、硕士；李　猛，助理工程师；刘首文，高级工程师、博士。

收稿日期：2021-07-29　　**修回日期：**2021-09-30　　**E-mail：**lim@tecorigin.com

0 概述

在电力系统仿真中,电磁暂态仿真是主要的系统应用之一,也是电力系统安全分析和运行的关键组件[1],该应用的核心算法是对大规模线性方程组$\boldsymbol{Ax}=\boldsymbol{b}$进行求解。通过对实际数据的分析可知,线性方程组中的系数矩阵$\boldsymbol{A}$通常为稀疏矩阵,具体到电力系统,其稠密度通常小于1%[2-3],如果不能有效利用矩阵的稀疏性,在使用计算机处理大型稀疏矩阵时,大量的存储和计算资源将会浪费在无效的零元上,导致存储空间不足和处理效率低下[4]。因此,在实现大型稀疏矩阵存储和计算时,需要利用专用的算法和数据结构,基于特殊设计的硬件架构,最大化地利用计算系统的算力和效能[5-6]。综上,电力系统的电磁暂态加速求解问题最终聚焦于稀疏线性方程组的加速求解问题。

稀疏线性方程组的求解方法包括直接法和迭代法两大类。由于迭代法存在迭代次数不可控、结果精度较低等问题,因此一般使用直接法进行求解。直接法是指在不考虑计算舍入误差的情况下,通过矩阵分解和三角方程[7-8]进行求解。下三角稀疏矩阵求解是求解稀疏线性方程组的核心算法的组成部分[9],而层次集合法[10-11]是最重要的并行性分析方法之一。近年来,越来越多的算法开始使用众核加速硬件来实现并行加速。NAUMOV[12]提出基于层次集合的方法,其将多个小的层次集合进行合并,以减少同步运算。PARK等[13]通过对同步进行剪枝来提升效率。LIU等[14-15]提出利用图形处理器(Graphics Processing Unit,GPU)的原子操作来实现无同步的算法。SU等[16]介绍大规模的线程级无同步算法。LU等[17]介绍基于循环块优化的算法。上述算法主要在GPU上进行优化,WANG等[18]则提出了在国产神威众核上实现的加速算法。

相比众核架构(GPU或神威众核),现场可编程门阵列(Field Programmable Gate Array,FPGA)具有片上缓冲大、传输带宽可定制、调度灵活等优势,可以最大化地利用计算系统的算力[19]。吴志勇等[20]设计了一种面向FPGA求解稀疏矩阵的硬件架构,本文基于这一硬件结构,提出软件映射和调度求解算法。该算法给出数据分布和指令排布的过程,通过将下三角稀疏矩阵的求解过程静态映射到多个FPGA片上的处理单元(Process Elements,PEs),以软硬件协同的方法实现下三角稀疏矩阵在定制化FPGA架构上的高速求解。

1 FPGA稀疏矩阵求解器硬件设计

FPGA稀疏矩阵求解器的硬件设计在文献[20]中已经详细说明,本文只简述该求解器的基本原理和结构,以便进一步介绍本文所提软件调度算法。直接法需要遵循下三角稀疏矩阵内部的对角元依赖关系,其求解路径构成一个有向无环图(Directed Acyclic Graph,DAG)[21],如图1所示。由于DAG的条件路径依赖关系无法直接映射为并行化算法,因此无法直接在算法层进行并行优化。下三角稀疏矩阵具有稀疏性,构成的DAG存在隐式的数据并行,意味着多个对角元和非对角元之间存在并行处理的可能。整个系统的基本思想是设计多个硬件处理单元,每个处理单元都能够有效地处理对角元和非对角元。通过软件对该稀疏矩阵进行预处理,找到稀疏下三角的并行性,并且将划分完成的下三角稀疏矩阵映射到这些硬件处理单元上,从而实现软硬件协同的并行求解。

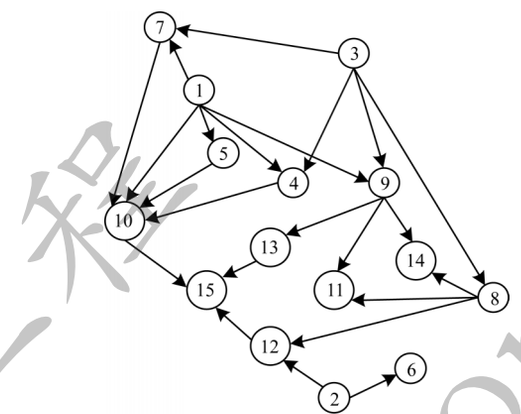

图1 有向无环图

Fig.1 Directed acyclic graph

由于软件预处理及划分完成后的稀疏下三角各个部分(对角元及非对角元之间)存在相互依赖,因此需要为多个处理单元间提供硬件连接通路。在实际的FPGA实现过程中,使用二维单向环网来实现多个处理单元间的通信连接,从而完成各单元间的数据传输[22]。FPGA稀疏矩阵求解器的硬件结构如图2所示。

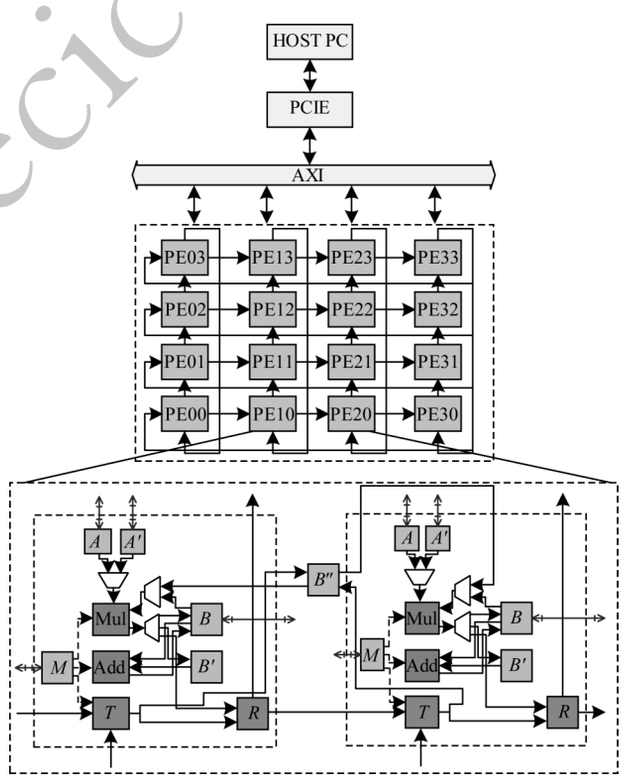

图2 求解器的硬件结构

Fig.2 Hardware structure of solver

从图2可以看出,系统中存在多个处理单元,每个处理单元都有一个指令控制部件M,用来存储PE内部控制部件的指令码。另外,每个处理单元内还有3类操作部件,分别是Mul部件、Add部件和T/R部件,其中:Mul部件用来处理稀疏矩阵对角元和非对角元的双精度浮点复数乘法(包括除法)操作;Add部件用来处理非对角元对同行对角元的更新;T/R部件用来实现相邻PE间的数据传输。

每个PE内部包括5个专用缓冲,分别是A、A'、B、B'和B'',其中:缓冲A用来存储稀疏矩阵对角元系数;缓冲A'用来存储稀疏矩阵非对角元系数;缓冲B用来存储稀疏矩阵右端项和求得的未知数;缓冲B'用来存储非对角元和同列未知数的乘积;缓冲B''用来存储其他PE传输过来的已求解的未知数x。由于每个PE对B''缓冲的写操作一定都指向不同的地址,因此B''缓冲可以在2个相邻单元间共享,以减少对FPGA资源的占用,有利于FPGA综合实现。处理单元间则通过二维单向环网相连来进行通信。基于该硬件架构,原本串行的稀疏矩阵求解过程被分解为以下操作步骤:

1)使用Mul部件,从缓冲A中读取对角元系数,从缓冲B中读取右端项,求解获得未知数x,再存放到缓冲B中。这个x不被其他PE上的非对角元需求,无需传输。

2)使用Mul部件,从缓冲A中读取对角元系数,从缓冲B中读取右端项,求解获得未知数x,存放到缓冲B中。如果这个x被其他PE的非对角元需求,则同时将其发送到R部件。

3)使用Mul部件,从缓冲A'中读取非对角元系数,从缓冲B中读取求得的x,计算获得非对角元乘积,将其存放在缓冲B'中。

4)使用Mul部件,从缓冲A'中读取非对角元系数,从缓冲B''中读取求得的x,计算获得非对角元乘积,将其存放在缓冲B'中。本次读取到的x对应第2步操作中传输到R部件中的x。

5)使用Add部件,从缓冲B'中读取非对角元乘积,从缓冲B中读取对角元,计算更新对角元,并将结果存放到缓冲B中。

6)使用T部件,将左方或者下方传输来的x通过网络传输到下一个PE,并保存到R部件和缓冲B''中。

根据上述稀疏矩阵求解过程,双精度浮点复数乘法单元和加法单元的数据调度过程分别如图3和图4所示。

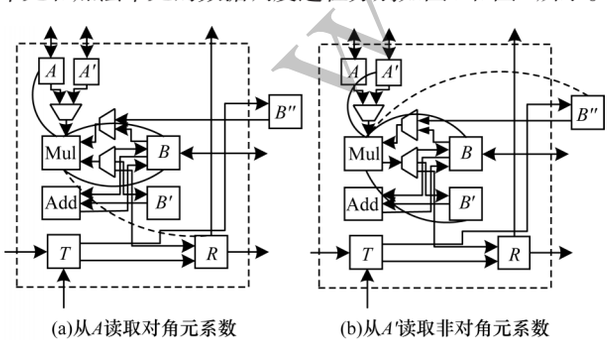

图3　乘法单元的数据调度

Fig.3　Data scheduling of the multiplication unit

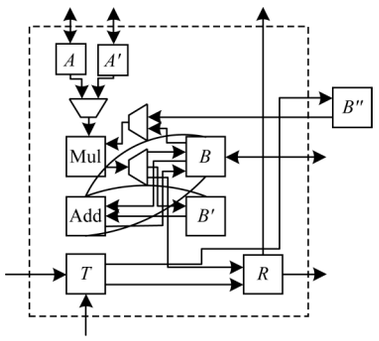

图4　加法单元的数据调度

Fig.4　Data scheduling of the addition unit

2　软件静态调度算法

基于上文FPGA稀疏矩阵求解器的硬件架构,本文实现了稀疏矩阵求解算法和求解过程的映射,充分挖掘了稀疏矩阵求解过程中的并行性,这一映射过程通过软件静态调度算法来实现。中央处理器(Central Processing Unit,CPU)一般通过顺序求解所有解向量来实现,而GPU则会对解向量进行分块,在分块后的解向量之间通过引用计数构建依赖关系,然后逐块触发从而完成求解。无论是CPU还是GPU,都可以通过动态寻址来定位解向量,但是,FPGA不具备动态寻址的能力,因此,必须构建静态指令流,指示每个操作部件在特定步时的访存地址和处理操作。

2.1　稀疏矩阵预处理

在正式求解之前,需要对矩阵进行预处理。稀疏矩阵存储通常使用压缩稀疏列矩阵(Compressed Sparse Column matrix,CSC)或压缩稀疏行矩阵(Compressed Sparse Row matrix,CSR)[23]。首先,将矩阵进行重排序,以减少LU分解增加的额外非零元,本文使用metis[24]工具进行重排序,其能提供一组可以独立运行的命令行程序,同时也提供应用程序接口(Application Programming Interface,API),方便集成到C/C++或Fortran程序中,该程序可以得到上述算法目标的一个近似解;其次,对完成重排序的稀疏矩阵进行LU分解,获得右下局部稠密的下三角稀疏矩阵,如果没有特殊说明,下文以$\boldsymbol{L}$代指这里的下三角稀疏矩阵。

2.2　矩阵划分

针对上述硬件架构,软件调度算法需要为其每个PE进行任务分派和调度,将求解下三角稀疏矩阵$\boldsymbol{L}$需要用到的数据分布到多个计算单元上。

数据分布的设计目标是使得尽可能多的计算单元尽可能饱和地运行。第一个目标是使得分布到多个PE上的数据尽可能均匀,以保证每个PE需要求解的数据量接近。由于稀疏矩阵的非零元计算过程实际上是一个有向无环图,因此将一个稀疏矩阵映射到多个PE上的问题可以认为是一个图划分(Graph Partition)问题。第二个目标是使得分割后的

各个PE间通信尽可能少,从而减小通信开销,使计算单元尽可能饱和地运行。

图的均匀划分是一个NP困难(NP-hard)问题,同样使用metis提供的接口,对$\boldsymbol{L}$矩阵的对角元进行划分。划分完成的对角元分布在不同的分块中,这些分块一一映射到多个不同的PE上。原有的各个对角元之间的依赖关系被分成了PE内部的对角元依赖关系和跨PE的对角元依赖关系,分别保存在pe_inside_diag_dep和pe_outside_diag_dep数组中。

非对角元被分布到其所在行的对角元所在的PE上,这样操作的好处是使得非对角元就绪后,可以直接更新其所在行的对角元,传输部件只需要传输对角元,能够大幅简化与传输部件相关的算法设计。

2.3 传输算法

在实际的硬件实现中,当PE数较多时,PE间直接相连会带来巨大的网络资源消耗,因此,硬件设计通常使用二维mesh网络来替代PE间的直接连接。但是,二维mesh网络带来的问题是每个PE只和周围2个维度上的4个PE直接相连,当要和远端的PE通信时,需要经过中间的PE进行连接,此时需要多个时钟周期来完成传输。

当前某个PE计算获得的x有可能会被最远端的PE访问,因此,在软件调度算法上,一开始选择广播算法进行数据共享。广播算法可以保证当前PE计算求得的未知数x最终一定能够被所有的PE访问到,并且对于二维单向环网而言,广播算法非常容易设计和实现,如图5所示。二维单向环网中所有的PE对于网络是完全对称的,从非0号PE发起的广播传输过程也是类似的。在使用广播算法时,同一时刻多个PE发起的消息只要满足PE的行号与列号之和不同于PE数量(mod PE),就可以在系统中并行传输。

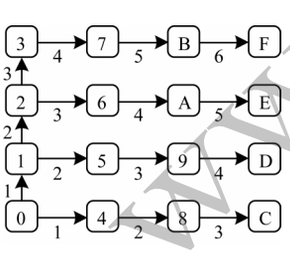

图5 从0号PE开始的16PE广播算法

Fig.5 16PE broadcast algorithm starting from PE 0

实际上,并非所有的PE都需要获得当前正在传输的x,广播算法会占用不需要x的PE的传输部件T/R。为了解决这一问题,可以将广播算法改为点对点传输算法,点对点算法的路由与广播算法一致,但只占用发送PE和接收PE之间的通路,空出的通路可以同时传输其他x,这使得系统中能够同时传输更多的x,进一步提升了传输效率。

2.4 指令排布和生成

在数据分布完成后,就可以对指令进行排布,需要完整地规划每个PE上的部件操作,从而充分利用每个PE的操作能力。在指令排布时需要确定以下信息:

1)需要明确硬件时钟的频率和延迟。

上述硬件设计中的操作部件并不是立刻得出结果的,当频率不同时,各个操作部件可能会产生一定的延迟。在一般情况下,频率越高,延迟越大。当频率控制在300~400 MHz时,Mul部件的延迟是5拍,而Add部件的延迟是3拍。在指令排布时,设定Mul_lat和Add_lat分别表示Mul部件和Add部件的延迟。

2)需要明确系统中各个部件间的依赖关系。

如上文所述,系统中主要存在2种依赖关系,即PE内部对角元之间的依赖和跨PE的对角元之间的依赖,分别被保存在pe_inside_diag_dep和pe_outside_diag_dep数组中,这里直接使用这2个数组即可。

3)需要明确系统中各个部件间的冲突关系。

通过分析可以发现,当前硬件系统中没有访存冲突,所有的内存模块至多只有2个写入端口和2个读出端口,可以通过分频实现复用。然而,从上述求解过程可以看出,系统中存在不同操作对部件的争用冲突,主要包括以下3种:

(1)对Mul部件的争用。当Mul部件用于对角元求解时,就无法用于非对角元的乘法,反之亦然。

(2)对R部件的冲突。如果T部件在某一时刻被占用,那么相应的R部件也被占用,无法再进行传输。

(3)对Add部件的争用。当Add部件正在处理某一行的右端项更新时,同行的非对角元不能同时更新右端项,否则会造成读写冲突。

为了解决冲突问题,需要为上述可能发生冲突的部件设置部件占用标记。某一时刻,当部件占用标记被置位时,意味着某个部件已经被占用,无法再用其排布指令。本文将上述3个部件占用标记分别设置为Mul_occupy、R_occupy和Add_occupy,其中,Add_occupy是一个长度为Add_lat的列表,用来标记当前加法部件正在处理的行。

图6所示为指令排布的算法流程,算法的伪代码如算法1所示。在排布时,乘法单元会优先处理非对角元,只有当非对角元全部处理完成后才会处理对角元,这样做是为了尽可能多地在PE上生成可供处理的对角元,以便保持PE的满载,提高整个系统的并行度。

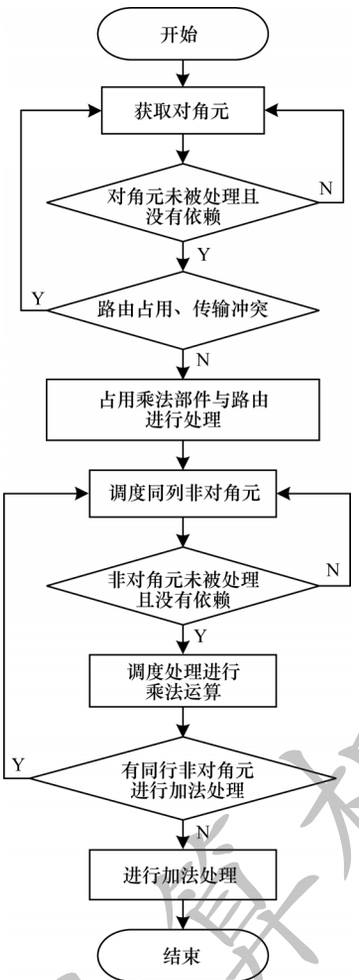

图6　指令排布算法流程

Fig.6　The procedure of instruction layout algorithm

算法1　指令排布算法

while(对角元未处理完成)

对于所有的PE循环:

寻找该PE上就绪的非对角元(同列对角元已被处理,且可用)

在找到非对角元后,占用乘法部件,处理该非对角元;

如果乘法部件未被占用:

寻找到该PE上就绪的对角元(所有前驱非对角元都已经更新到右端):

如果有后继对角元在其他PE上:

后继对角元处理需要用到的其他PE的传输部件没有被占用:

占用乘法部件,更新该对角元;

占用后继步需要使用的多个传输部件。

后继对角元处理需要用到的其他PE的传输部件被占用:

寻找该PE上处理完成的非对角元:

如果该非对角元所在的行右端项当前没有在更新:

占用加法部件更新右端项;

对于当前拍各个部件的操作结果,更新该PE上相关的依赖关系和状态。

指令排布完成后就可以生成指令。系统中主要有6种不同的操作,其中,Mul部件占据了4种操作,Add部件和T部件各占据1种操作,每个时刻都有可能出现所有操作部件同时工作的情况。由于各操作的取数空间基址和偏移都不同,需要将多种不同的操作整合起来生成指令。生成的指令放在缓冲M中,每个存储单元的位宽为128位,指令格式中的每个字段的位、名称和含义如表1所示。

表1　指令及其含义

Table 1　Instructions and their meanings

位	名称	含义
15:0	addr0	乘数从A_RAM或A'_RAM取数的地址
31:16	addr1	乘数从B_RAM或B''_RAM取数的地址(乘数读入A时,必定使用B的地址进行更新)
47:32	addr2	乘数读入A'和B(或B'')时写入B'_RAM的地址
63:48	addr3	加数从B_RAM取数的地址(加数一定写入B)
79:64	addr4	加数从B'_RAM取数的地址
95:80	t_addr	传输部件需要写入B''_RAM的地址
98:96	mul_mode	乘法部件模式,处理对角元或非对角元或空转
99	add_mode	加法部件模式,更新右端项或空转
103:100	trans_mode	T/R模块的传输方式,向上或向右传输
127:104	保留	保留位,当前没有使用

2.5　硬件空间限制

在硬件实现的过程中,通常都存在一定的空间限制。由于缓冲区A、A'、B、B'、B''都使用片上内存实现,导致其空间相对受限,也意味着求解的矩阵大小是受限的。

缓冲区A和B分别存储分布在某个PE上的对角元系数和右端项(包括求解得到的x)。假设对图的分割相对均匀,则A和B的大小约为(N/PE_num),其中,N表示矩阵阶数,PE_num表示PE的个数。

A'存储非对角元系数,B'暂存乘以同列未知数x后的非对角元。非对角元不作划分,而是直接分配到所在行对角元的PE上。假设分布均匀,则A'和B'的大小约为(nnz/PE_num),其中,nnz表示非零元的个数。

B''存储所有PE间求解得到并且需要传输的x。B''的大小受矩阵稀疏度和PE_num影响,矩阵越稠密,PE的个数越多(划分越多),B''则越大。B''最大不会超过N,实际大小依赖于具体矩阵,一般来说会比N小得多。

硬件总的空间占用约为(N+nnz+N×PE_num)个实数或复数,在使用算法前,需要评估FPGA上提供的空间是否能满足实际的空间需求。

3　性能测试分析

算法求解的过程可以分为3个阶段:

1)第一阶段是矩阵求解初期的稀疏部分,此时绝大部分PE(大于80%)能够找到就绪的对角元或非对角元,用于运行乘法部件,即使有依赖元暂未求解或硬件资源存在冲突,通常也只需等待1~2拍就能够继续运行。

2)第二阶段是矩阵求解中期的相对稀疏部分,此时只有部分PE(大于10%而小于80%)能够运行乘法部件,剩余的PE由于对角元未就绪(需要等待同行非对角元处理完成)或找不到非对角元(需要等待求解出的未知数x)进行处理,因此只能等待。

3)第三阶段是矩阵求解后期的稠密部分,此时只有极少部分PE(小于10%)能够运行乘法部件,多个未知数x求解时存在强相关性,求解过程串行化明显增加。

图7所示为FPGA加速器的逻辑结构。本文对应的项目在Xilinx XCVU37P(8 GB HBM、4 200万等效逻辑门)FPGA上实现硬件加速器,其与主机间采用PCIE4.0 X16接口(带宽为64 GB/s)连接。主机采用20核Intel Xeon Gold 5320H(4.3 GHz)处理器,128 GB 3 200 Mb/s DDR4配置,运行Linux操作系统,移植和适配了电力系统仿真所需的库环境。FPGA加速器实物如图8所示。本文硬件结构设计开销情况如表2所示。

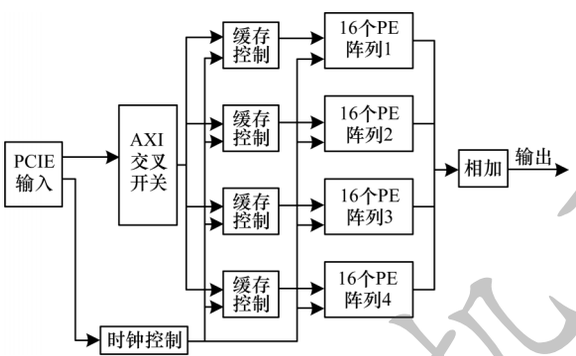

图7 FPGA加速器逻辑结构

Fig.7 Logic structure of FPGA accelerator

图8 FPGA加速器实物示意图

Fig.8 Physical diagram of FPGA accelerator

表2 硬件资源利用情况

Table 2 Utilization of hardware resource

硬件资源	使用量	可用量	利用率/%
LUT	336 646	1 303 680	25.82
LUTRAM	7 109	600 960	1.18
FF	186 534	2 607 360	7.15
BRAM	1 074	2 016	53.27
URAM	640	960	66.67
DSP	2 560	9 024	28.37
IO	1	624	0.16
GT	8	24	33.33
BUFG	18	1 008	1.79
MMCM	1	12	8.33
PCIE	1	6	16.67

本文对2个典型算例进行测试,2个算例均来源于实际的电网模型。算例1的矩阵大小为10 188×10 188,非零元为25 720个;算例2的矩阵大小为21 464×21 464,非零元为121 890个。算例测试结果如表3所示。

表3 2个典型算例测试结果

Table 3 Test results of two typical examples

算例	第一阶段节拍数	第二阶段节拍数	第三阶段节拍数	总节拍数
算例1	约为350	约为600	约为400	1 251
算例2	约为1 200	约为2 770	约为700	4 672

从表3可以看出,在64个PE配置的FPGA加速器中,算例1约耗时1 251拍,则每拍能够处理约20个非零元,算例2约耗时4 672拍,每拍能够处理约25个非零元。

将当前国网电力系统电磁暂态仿真程序中计算最为密集的稀疏矩阵消元求解核心段作为实际测试对象,对基于FPGA和基于传统CPU/GPU的2种环境进行对比测试和分析。当FPGA加速器系统在300 MHz频率、256个PE配置下时,针对上述核心段的处理效率能达到30.84 GFLOPs。与此同时,将该电磁暂态仿真程序移植到20核Intel Xeon Gold 5320H(4.3 GHz)处理器平台上,进行多核上的多线程并行优化,同时降低大数据量离散访问导致的CACHE缓存颠簸,减少访存延迟。处理器中4核用于操作系统和驱动运行,其余16核用于优化后的稀疏矩阵并行运算,其处理效率仅为5.22 GFLOPs。基于FPGA算法的实测性能是传统CPU/GPU求解算法的5.9倍,加速效果显著。

虽然本文所提算法相较传统CPU/GPU算法有明显的加速效果,但仍然存在优化的空间。从上述3个求解阶段来看,第一阶段已经充分利用了所有的硬件计算资源,几乎没有可以优化的余地;第二阶段的节拍占比最大,随着矩阵规模的增大,其增长速度也最快,存在优化算法调度的可能,主要优化思路是更好地进行矩阵的划分和映射,进一步减小通信代价,提升算法并行度;第三阶段主要是矩阵的稠密部分,可以使用逆矩阵乘法进行优化。在LU分解时,通过行列调整可以使得矩阵$\boldsymbol{L}$右下角局部稠密,对于稠密的部分,可以求其逆矩阵,通过逆矩阵乘法进行求解。由于矩阵乘法可以在多个PE上完全并行,因此当矩阵有一定的稠密度时,用逆矩阵乘法进行求解相比直接法更具性能优势。

4 结束语

本文提出一种基于FPGA硬件的静态调度优化算法,利用该算法实现了下三角稀疏矩阵的求解。通过对稀疏矩阵直接法求解步骤的分解和对稀疏矩阵的解析排布,设计一种节拍级的静态调度流程,以

充分利用FPGA的硬件资源获取较高的求解效率和硬件利用率。性能测试结果验证了该算法的高效性,对于在FPGA上实现类似的基于图划分的隐式数据并行算法具有一定的参考意义。下一步拟对LU分解中的右下角局部稠密矩阵进行优化,无需修改现有的硬件拓扑,仅增加稠密惩罚操作指令,在软件方面,对稠密部分进行求逆并均匀划分求得的逆矩阵,然后通过乘法来求解该矩阵。

参考文献

[1] 罗捷,袁康龙,钟杰峰,等. 考虑新能源接入对系统调峰性能影响的随机生产模拟算法[J]. 电力系统保护与控制,2019,47(8):180-187.
LUO J, YUAN K L, ZHONG J F, et al. Probabilistic production simulation algorithm considering new energy's impact on regulation[J]. Power System Protection and Control, 2019,47(8):180-187. (in Chinese)

[2] 周挺辉,赵文恺,严正,等. 基于图形处理器的电力系统稀疏线性方程组求解方法[J]. 电力系统自动化,2015,39(2):74-80.
ZHOU T H, ZHAO W K, YAN Z, et al. A method for solving sparse linear equations of power systems based on GPU[J]. Automation of Electric Power Systems, 2015, 39(2):74-80. (in Chinese)

[3] 刘珊瑕,靳松,王文琛,等. 基于变量相关性分解方法的稀疏线性方程组并行求解算法[J]. 清华大学学报(自然科学版),2020,60(4):312-320.
LIU S X, JIN S, WANG W C, et al. Parallel algorithm for solving sparse linear equations based on variable correlation decomposition[J]. Journal of Tsinghua University (Science and Technology), 2020, 60(4):312-320. (in Chinese)

[4] 彭宇,仲雪洁,王少军. 基于FPGA线性方程组的存储优化设计[J]. 计算机工程,2013,39(4):287-290,295.
PENG Y, ZHONG X J, WANG S J. Design of storage optimization based on FPGA linear equations system[J]. Computer Engineering, 2013, 39(4):287-290, 295. (in Chinese)

[5] 李亿渊,薛巍,陈德训,等. 稀疏矩阵向量乘法在申威众核架构上的性能优化[J]. 计算机学报,2020,43(6):1010-1024.
LI Y Y, XUE W, CHEN D X, et al. Performance optimization for sparse matrix-vector multiplication on Sunway architecture[J]. Chinese Journal of Computers, 2020, 43(6):1010-1024. (in Chinese)

[6] 张禾,陈客松. 基于FPGA的稀疏矩阵向量乘的设计研究[J]. 计算机应用研究,2014,31(6):1756-1759.
ZHANG H, CHEN K S. Design and implementation of sparse matrix vector multiplication on FPGA [J]. Application Research of Computers, 2014, 31(6):1756-1759. (in Chinese)

[7] MOSHFEGH J, MAKRIS D G, VOUVAKIS M N. Parallel direct domain decomposition methods (D3M) for finite elements[C]//Proceedings of IEEE International Symposium on Antennas and Propagation and USNC-URSI Radio Science Meeting. Washington D. C., USA: IEEE Press, 2019:777-778.

[8] GE X, ZHU H L, YANG F, et al. Parallel sparse LU decomposition using FPGA with an efficient cache architecture[C]//Proceedings of IEEE International Conference on ASIC. Washington D. C., USA: IEEE Press, 2017: 259-262.

[9] DUFF I S, ERISMAN A M, REID J K. Direct methods for sparse matrices[M]. New York, USA: Oxford University Press, 2017.

[10] CHOW E, ANZT H, SCOTT J, et al. Using Jacobi iterations and blocking for solving sparse triangular systems in incomplete factorization preconditioning[J]. Journal of Parallel and Distributed Computing, 2018, 119:219-230.

[11] MARRAKCHI S, JEMNI M. Fine-grained parallel solution for solving sparse triangular systems on multicore platform using OpenMP interface[C]//Proceedings of International Conference on High Performance Computing & Simulation. Washington D. C., USA: IEEE Press, 2017:659-666.

[12] NAUMOV M. Parallel solution of sparse triangular linear systems in the preconditioned iterative methods on the GPU [EB/OL]. [2021-06-05]. https://research. nvidia. com/sites/default/files/publications/nvr-2011-001. pdf.

[13] PARK J, SMELYANSKIY M, SUNDARAM N, et al. Sparsifying synchronization for high-performance shared-memory sparse triangular solver [C]//Proceedings of International Supercomputing Conference. Berlin, Germany: Springer, 2014:124-140.

[14] LIU W F, LI A, HOGG J D, et al. A synchronization-free algorithm for parallel sparse triangular solves[C]//Proceedings of International Supercomputing Conference. Berlin, Germany: Springer, 2016:617-630.

[15] LIU W F, LI A, HOGG J D, et al. Fast synchronization-free algorithms for parallel sparse triangular solves with multiple right-hand sides[J]. Concurrency and Computation: Practice and Experience, 2017, 29(21):e4244.

[16] SU J Y, ZHANG F, LIU W F, et al. CapelliniSpTRSV: a thread-level synchronization-free sparse triangular solve on GPUs[C]//Proceedings of the 49th International Conference on Parallel Processing. Washington D. C., USA: IEEE Press, 2020:1-11.

[17] LU Z, NIU Y, LIU W. Efficient block algorithms for parallel parse triangular solve[C]// Proceedings of the 49th International Conference on Parallel Processing. Washington D. C., USA: IEEE Press, 2020:3-11.

[18] WANG X L, LIU W F, XUE W, et al. swSpTRSV: a fast sparse triangular solve with sparse level tile layout on Sunway architectures[C]//Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. New York, USA: ACM Press, 2018: 338-353.

[19] 贾迅,钱磊,邬贵明,等. FPGA应用于高性能计算的研究现状和未来挑战[J]. 计算机科学,2019,46(11):11-19.
JIA X, QIAN L, WU G M, et al. Research advances and future challenges of FPGA-based high performance computing[J]. Computer Science, 2019, 46(11):11-19. (in Chinese)

[20] 吴志勇,王晞阳,陈继林. 一种基于FPGA并行加速的稀疏矩阵求解方法[J]. 电力系统保护与控制,2021,49(11):155-162.
WU Z Y, WANG X Y, CHEN J L. A method for solving a sparse matrix based on FPGA parallel acceleration[J]. Power System Protection and Control, 2021, 49(11):155-162. (in Chinese)

(下转第213页)

(上接第205页)

[21] 王智铎,江波,苗瑞,等. 基于有向图的外键冲突解决算法设计与实现[J]. 计算机工程,2021,47(2):254-260.
WANG Z D, JIANG B, MIAO R, et al. Design and implementation of solution algorithm for foreign key conflict based on directed graph[J]. Computer Engineering, 2021,47(2):254-260. (in Chinese)

[22] 苏锦柱,邬贵明,贾迅. 二元域大型稀疏矩阵向量乘的FPGA设计与实现[J]. 计算机工程与科学,2016,38(8):1530-1535.
SU J Z, WU G M, JIA X. Design and implementation of large sparse matrix vector multiplication on FPGA over GF(2)[J]. Computer Engineering & Science, 2016,38(8): 1530-1535. (in Chinese)

[23] 程凯,田瑾,马瑞琳. 基于GPU的高效稀疏矩阵存储格式研究[J]. 计算机工程,2018,44(8):54-60.
CHENG K, TIAN J, MA R L. Study on efficient storage format of sparse matrix based on GPU[J]. Computer Engineering, 2018,44(8):54-60. (in Chinese)

[24] 苗新强,金先龙,丁峻宏. 结构动力数值仿真两级并行计算系统开发及应用[J]. 计算机辅助设计与图形学学报,2015,27(6):1126-1133.
MIAO X Q, JIN X L, DING J H. The development and application of two-level parallel computing system for structural dynamic numerical simulation[J]. Journal of Computer-Aided Design & Computer Graphics, 2015, 27(6): 1126-1133. (in Chinese)

编辑　吴云芳