基于 GPU 的电磁暂态并行仿真研究

姚蜀军[1], 韩民晓[1], 张 硕[1], 林芝茂[1], 李 昊[2]

(1. 华北电力大学电气与电子工程学院, 北京 102206;
2. 国网山东省电力公司烟台栖霞供电局, 山东 烟台 265300)

摘要: 随着电力系统大量高压直流、新能源、柔性交流输电系统的接入,传统电磁暂态仿真由于步长小、仿真速度慢,已无法满足大规模系统的仿真需求。近年来,图形处理器(GPU)在高性能计算领域发展迅速,通过 GPU 优化从而提高电磁暂态仿真计算效率成为可能。本文在研究了 GPU 结构特点后,提出了一种适用于电磁暂态仿真线性方程组的并行 LU 分解算法,从三个方面对传统算法进行了加速。通过和基于 CPU 的传统算法的仿真结果对比,证明了本算法的优越性。

关键词: 电磁暂态仿真; 图形处理器; LU 分解; 并行算法

DOI: 10.12067/ATEEE1810040 **文章编号**: 1003-3076(2019)01-0010-07 **中图分类号**: TM743

1 引言

近年来,我国已经形成了全世界电压等级最高、规模最大的电力系统网络,并且正在从传统电网向智能电网方向发展。然而,现代电力系统由于大量高压直流、新能源、柔性交流输电系统的接入,在带来巨大经济效益的同时,电网整体的安全性也受到了严重的影响[1]。随着电力电子设备大规模应用于电力系统,系统谐波和系统之间的相互耦合作用日益增加[2]。高频动作的电子设备使得分析系统的暂态过程越来越复杂,微秒级和毫秒级的电磁暂态过程、毫秒级和秒级的机电暂态过程以及分钟级以上的中长期动态过程之间的耦合程度越来越高,相互影响也越来越大。这就使得原来以准稳态模型为基础的机电暂态仿真,在某些特殊运行工况下不能再客观地反映真实情况[3]。因此,要准确地得到电力系统暂态过程的详细数据,就需要进行电力系统全电磁暂态仿真分析[4]。

传统的电磁暂态仿真大多采用中央处理器(Central Processing Unit, CPU)来实现,其计算效率也受到 CPU 体系结构的制约。一方面,计算机硬件的更新速度逐渐放缓,传统单核 CPU 的计算能力已接近极限,多核 CPU 所能集成的核数有限且导致并行计算开展困难。因而单纯依靠传统 CPU 架构的计算机来实现大规模电力系统全电磁暂态仿真分析显然不切实际[5]。另一方面,图形处理器(Graphics Processing Unit, GPU)迅速崛起,在高性能计算领域逐渐占据一席之地。GPU 多核心高集成度的特点使得其在进行大数据并行计算时有着得天独厚的优势。因此,通过研究基于 GPU 多核处理器的并行仿真技术,开发适合大规模电力系统的电磁暂态并行仿真程序,可以进一步提高现有电网仿真软件的分析仿真能力,为国家电网的规划发展和安全稳定运行提供强有力的技术支撑[6]。

目前,在电力系统领域,基于 GPU 的高性能计算也有了一些探索性的研究。文献[7]通过对 GPU 架构和电磁暂态仿真原理的研究,论证了通过 GPU 加速电磁暂态仿真的可行性。文献[8]总结了当前电力系统仿真的主要问题,提出了建立元件平均化模型和建立元件向量化计算模型通过 GPU 加速计算的两个策略。文献[9]首次提出了基于消去树的分层并行 LU 分解方法,其原理是将关系树中每一层的节点进行并行 LU 分解。这种分层并行的方法虽然也取得了一定的加速效果,但由于消去树分层并不均匀,底层节点数较多而顶层节点数较少,因而并行效果并不理想。

收稿日期: 2018-10-24

基金项目: 国家电网公司技术项目(SGHA0000KXJS1700353)

作者简介: 姚蜀军(1973-), 男, 湖北籍, 副教授, 研究方向为电力系统建模仿真、电力系统运行与控制;
韩民晓(1963-), 男, 陕西籍, 教授, 博导, 博士, 研究方向为直流输电、电力电子技术在电力系统中的应用。

本文在详细研究了线性方程组直接求解法后,提出了一种基于 GPU 的电磁暂态线性方程组并行求解算法。首先,本文对 GPU 架构及其特点进行了详细分析;然后,结合线性方程组串行求解算法提出了并行 Crout 分解算法,分别从 LU 并行分解、多流并行和前代回代并行三个方面对电磁暂态线性方程组求解过程进行加速;最后,分别采用改进并行算法与传统串行求解算法对同一仿真系统进行分析计算,对比两者计算结果及效率。

2　GPU 架构及编程特点

2.1　GPU 架构特点

在架构组成上,CPU 和 GPU 都具有控制单元、运算单元和存储单元这三个基本单元,但它们在核心中所占的比重却各不相同,图 1 展示了 CPU 与 GPU 在架构上的主要差异。可以看出,两者的区别在于缓存体系和数字逻辑运算单元的结构差异:CPU 每个核中缓存和控制单元所占的面积较大,计算单元面积则较小;而 GPU 则恰恰相反,每个核拥有的存储和控制单元较小,运算单元面积却远远大于 CPU[10]。这种结构上的差异导致 CPU 更适合处理强逻辑的运算,而 GPU 更适合多数据的运算。

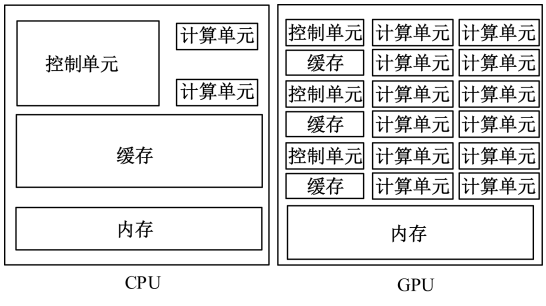

图 1　CPU 和 GPU 结构

Fig. 1　CPU and GPU structure

GPU 实际上由众多可伸缩处理器阵列(Thread Processing Cluster,TPC)组成,每个 TPC 包含多个流处理器簇(Streaming Multiprocessor,SM)、一个图形控制器和一个 SM 控制器[9]。每个 SM 核心包含完整的控制、存储和计算单元(SP),相当于 CPU 的一个核心。而 SP 仅含有寄存器和指令指针,是 GPU 中最基本的运算单元,具体的指令和任务都是通过它完成的,GPU 的内部结构如图 2 所示。

2.2　GPU 编程特点

CUDA(Compute Unified Device Architecture)程序分为两部分,一部分主机(Host)代码由 CPU 编译执行,包括设备(Device)代码启动程序、设备内存分配程序、主机内存和设备数据传输程序;另一部分设备代码由 GPU 执行,即所谓的内核函数(kernel),内核函数的调用是异步的,即主机仅仅把要执行的内核函数顺序提交给 GPU,并不等待其执行完成,然后直接处理后面其他的任务。另外,在调用内核函数时需要指定并行线程的数量。

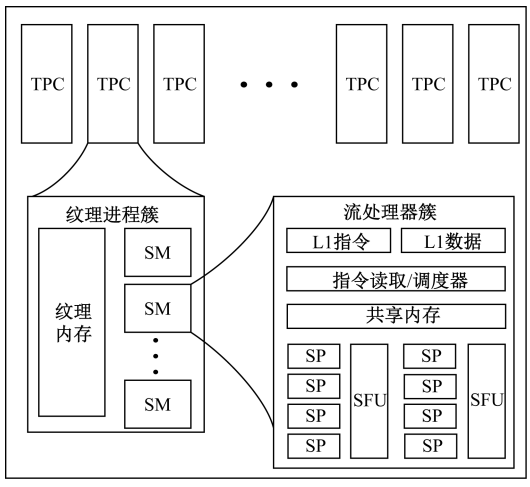

图 2　GPU 内部结构图

Fig. 2　GPU internal structure diagram

线程(Thread)是最基本的执行流,线程块(Block)是线程的组合,一个线程块中包含多个线程,线程网络(Grid)又包含多个线程块[11]。CUDA 运行时允许启动一个二维线程网络,并且线程网络中的线程块可以是三维线程数组。CUDA 的线程模型是一个三级结构,分别有线程、线程块和线程网络,如图 3 所示。

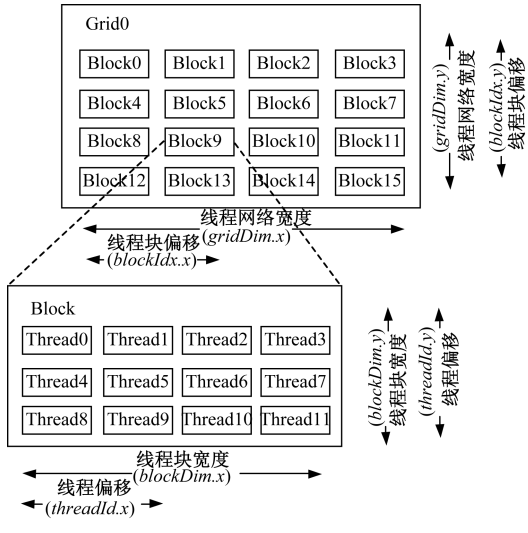

图 3　线程结构图

Fig. 3　Thread structure diagram

线程网络中某一维线程块的数量由 *gridDim* 表示,如网络中 x 维度上线程块的数量为 *gridDim. x*。同理,一个线程块中某一维线程的数量由 *blockDim* 表示。线程块在网络中的坐标由 *blockIdx* 表示,对于二维线程网络来说线程块的坐标即表示为(*blockIdx. x*,*blockIdx. y*)。同理,线程在线程块中的坐标由 *threadIdx* 表示,对于二维线程块下,线程的坐标即为(*treadIdx. x*,*treadIdx. y*),线程网络中的每个线程都有自己对应的线程块号和线程号。对于一个线程网络中的某个线程的索引可表示为:

$$\begin{cases} idx = (blockIdx.x \cdot blockDim.x) + threadIdx.x \\ idy = (blockIdx.y \cdot blockDim.y) + threadIdx.y \\ thread\_idx = (gridDim.x \cdot blockDim.x) \cdot idy + idx \end{cases} \tag{1}$$

CUDA 中的内核函数实质上是以线程块为单位执行的,每个网络内的多个线程块需要分配到不同的流处理器簇上分别调度运行,而同一个线程块中的线程需要共享数据,它们必须在同一个流处理器簇中执行。同一个流处理器簇上可以有多个活动的线程块轮转执行,即流处理器簇可以同时存储多个线程块的上下文数据,但同一时刻只能有一个线程块运行,不同线程块之间无法通信,也没有执行顺序。这样做的好处是,当一个线程块在进行同步或者访问显存等高延时操作时,流处理器簇可以转去处理另一个就绪的线程块,从而提高了计算速度[12]。

3 基于 GPU 的并行 Crout 求解算法

3.1 矩阵分块排序

稀疏线性方程组求解主要包括四个过程:预处理、符号分解、数值分解和求解,预处理主要是对稀疏矩阵进行分块和节点重排序,稀疏矩阵的 LU 分解可能有大量注入元产生,不同的消元顺序对应的 LU 矩阵规模会相差数倍[13]。因此,在求解稀疏线性方程组之前需要对其进行节点重排序,以寻找最利于消元的矩阵结构。符号分解是不进行具体元素数值的分解计算,其目的是在数值 LU 分解过程中根据这些记录,采用简单直接的寻址方式,使得数据查询量大大减少,计算速度明显提升。

为了提高开发效率,本文采用了应用最广泛的 KLU 软件包中的分块排序函数和符号分解函数进行矩阵的预处理及符号分解步骤。KLU 是由 Clark Kent 公司开发的一种用于求解稀疏线性方程组的高性能软件包[14],其排序算法原理如下:首先,KLU 通过深度优先算法形成节点关系图,根据节点间的连接关系将原矩阵排列成一个块对角矩阵,即对角线由一组块矩阵组成,对角块的左侧均为 0 元素,如图 4 所示;然后,KLU 会应用减小注入元排序算法对每个对角块矩阵进行重排序,保证其在后续分解过程中可以产生较小的注入元,从而加快计算速度。

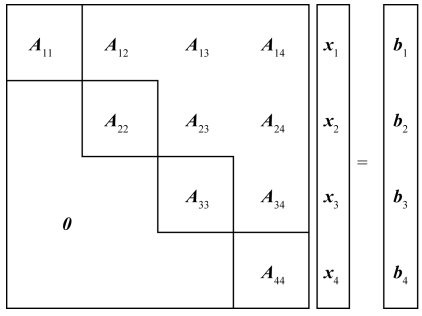

图 4 上对角块矩阵方程组

Fig. 4 Upper diagonal block matrix equations

由图 4 可知,KLU 函数库的排序算法可以将原系数矩阵变换为一个上对角块矩阵,这样原本稀疏矩阵的元素就集中在对角块上。这种方法使得只需对对角块进行后续的 LU 分解即可,大大降低了数值分解的计算量,并且并行分解也因此成为可能。

3.2 并行 Crout 分解算法

常用的 LU 分解有四种算法,分别为 Left-looking、Right-looking、Up-looking 以及 Crout 算法。其中 Crout 算法由于其分解过程中元素之间的计算依赖性较小,更利于并发多线程计算。基本 Crout 分解原理如下:

假设分解进行到第 i 步,该步之前的 $\boldsymbol{A}$ 矩阵元素已完成 LU 分解,如图 5 所示。矩阵中黑色部分为已知量,颜色较浅部分为第 i 步要分解的元素。

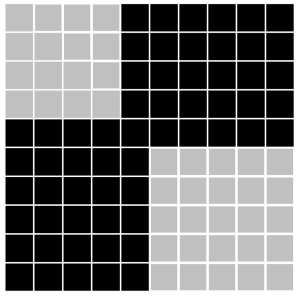

图 5 Crout 分解算法

Fig. 5 Crout decomposition algorithm

(1)按照式(2)计算 $\boldsymbol{U}$ 矩阵的第 i 行元素值:

$$u_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik}u_{kj} \quad i+1 \leqslant j \leqslant n \qquad (2)$$

(2)按照式(3)计算 u_{ii} 元素值:

$$u_{ii} = a_{ii} - \sum_{k=1}^{i-1} l_{ik}u_{ki} \qquad (3)$$

(3)通过式(4)和式(5)计算 $\boldsymbol{L}$ 矩阵的第 i 列元素:

$$l_{ji} = a_{ji} - \sum_{k=1}^{i-1} l_{jk}u_{ki} \quad i+1 \leqslant j \leqslant n \qquad (4)$$

$$l_{ji} = l_{ji}/u_{ii}, \quad l_{ii} = 1 \quad i+1 \leqslant j \leqslant n \qquad (5)$$

从上述计算过程可知,在 Crout 算法计算过程中,$\boldsymbol{U}$ 矩阵的第 i 行和 $\boldsymbol{L}$ 矩阵的第 i 列元素在计算时彼此无关,仅与 i 步之前的计算结果以及 $\boldsymbol{A}$ 矩阵元素有关,因此,此过程可通过 GPU 并行计算。而且,$\boldsymbol{L}$ 矩阵第 i 行最后的除法计算同样也可以配置独立的线程进行计算,但是其计算过程与上一步不同,因此需要重新配置一个内核函数进行计算。这样,Crout 串行算法即可改造成两个内核函数串联执行的多线程并行执行的算法。

综上所述,Crout 分解法的 GPU 优化算法计算过程如图 6 所示。流程中 Kernel 1 函数并发的线程数是 $\boldsymbol{L}$ 矩阵第 i 列元素数目和 $\boldsymbol{U}$ 矩阵第 i 行元素数目之和,Kernel 2 函数并发的线程数是 $\boldsymbol{L}$ 矩阵第 i 列元素数目。Kernel 2 函数的计算过程需要在 Kernel 1 函数的所有线程计算结束后才能进行,否则会出现计算错误。因而在 Kernel 1 函数最后加入线程同步操作,确保所有线程均已计算结束。同样,Kernel 2 函数最后加入线程同步操作是为了保证后续计算时该行的分解计算已完全结束。

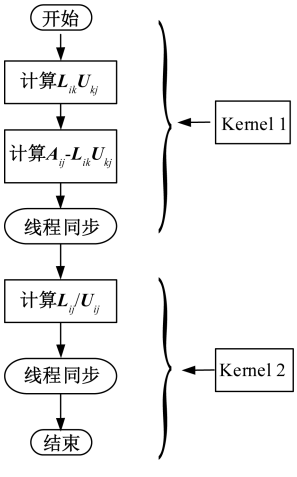

图 6　优化 Crout 算法流程

Fig. 6　Optimization of Crout algorithm flow

3.3　多流并行优化

GPU 在支持多线程并行的同时,也可以通过创建两个或多个不同的 CUDA 流水线进行多任务的并行。但 GPU 的多流并行与 CPU 不同,并不是完全意义上的并行,而是一种重叠并行的过程。

单个流水线执行可以划分为三个过程:CPU 向 GPU 传输数据、GPU 执行、GPU 向 CPU 传输数据,如图 7 所示。图 7 展示了一个 CUDA 执行流的时间线,其中中间部分代表 GPU 执行计算任务,第一部分代表 CPU 通过 PCI-E 总线向 GPU 传输数据,第三部分代表结果数据从 GPU 拷贝到 CPU 的过程。GPU 的多流并行是指同时创建多个执行流,利用执行部分与数据传输部分的重叠并行从而一定程度上隐藏数据传输带来的延时,达到提高运算效率的目的[5,6]。以两个执行流并行为例,流水线各部分配合的原理如图 8 所示。

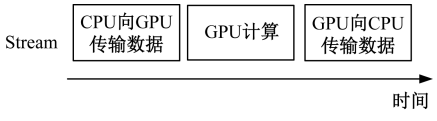

图 7　执行时间线

Fig. 7　Execution timeline

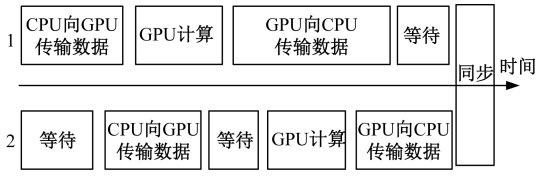

图 8　执行流重叠原理

Fig. 8　Principle of execution stream overlap

程序开始时创建两个 Stream,Stream 1 在进行数据传输时 Stream 2 处于等待状态。Stream 1 执行运行核函数时,Stream 2 执行 CPU 传输数据给 GPU,期间核函数执行过程一般会比数据传输时间长,因而 Stream 2 传输完成后会有一段等待时间。当 Stream 1 开始将数据从 GPU 拷贝到 CPU 中时,Stream 2 进入核函数执行过程。从图 8 中可以看出,虽然在并行过程中 Stream 1 和 Stream 2 都仍有等待时间,但这种重叠并行的执行总时长相对来说还是较短的。

根据 GPU 重叠并行的特点,本算法设计使用两条执行流来进行对角块矩阵的分解工作,具体方案如图 9 所示。可以看出,以四对角块矩阵为例,在主机端形成任务池并且开辟两条执行流,Stream 1 串

行计算 $\boldsymbol{A}_{11}$ 和 $\boldsymbol{A}_{22}$ 的分解过程，Stream 2 串行计算 $\boldsymbol{A}_{33}$ 和 $\boldsymbol{A}_{44}$ 的分解过程，两个执行流按照重叠并行规则并行运算。

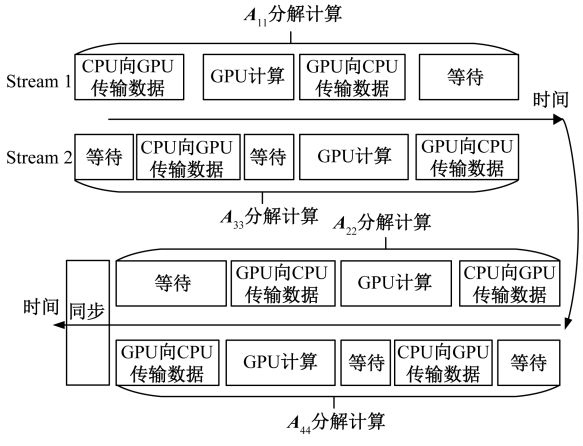

图 9 并行流分解原理

Fig. 9 Parallel flow decomposition principle

3.4 前推回代算法并行优化

在 LU 分解之前，矩阵会通过排序方式排列为对角块矩阵，然后对对角块矩阵进行分解，而非对角块矩阵不进行分解。分解结束后需要进行前推回代运算才能求得最终解，然而这一部分运算由于前后关联度较大，无法通过多线程并行计算，因而在 CPU 中运算较为合适。虽然基础前推回代运算无法并行，但 GPU 可对其中的矩阵与相量间运算进行加速。

仍以图 4 为例，当对角阵 $\boldsymbol{A}_{11}$、$\boldsymbol{A}_{22}$、$\boldsymbol{A}_{33}$ 和 $\boldsymbol{A}_{44}$ 均分解结束后，CPU 需要首先计算 $\boldsymbol{A}_{44}$ 矩阵对应的解，然后矩阵 $\boldsymbol{A}_{14}$、$\boldsymbol{A}_{24}$ 和 $\boldsymbol{A}_{34}$ 需要与解相量相乘从而更新右端项。首先，由式(6)求出对角块 $\boldsymbol{A}_{44}$ 对应的解向量 $\boldsymbol{x}_4$，然后由式(7)更新右端项 $\boldsymbol{b}_1$、$\boldsymbol{b}_2$ 和 $\boldsymbol{b}_3$。式(6)和式(7)联合可依次从对角块矩阵 A_{44} 开始倒序计算出线性方程组的解向量。

$$\begin{cases}\boldsymbol{L}_{44}\boldsymbol{y}_4 = \boldsymbol{b}_4 \\ \boldsymbol{U}_{44}\boldsymbol{x}_4 = \boldsymbol{y}_4\end{cases} \tag{6}$$

$$\begin{cases}\boldsymbol{b}_1 = \boldsymbol{b}_1 - \boldsymbol{A}_{14}\boldsymbol{x}_4 \\ \boldsymbol{b}_2 = \boldsymbol{b}_2 - \boldsymbol{A}_{24}\boldsymbol{x}_4 \\ \boldsymbol{b}_3 = \boldsymbol{b}_3 - \boldsymbol{A}_{34}\boldsymbol{x}_4\end{cases} \tag{7}$$

式(7)运算过程可通过 GPU 计算加速，其并行方法与 3.2 节中的 Kernel 1 类似，不再详细叙述。

4 算法测试

4.1 算例说明

本文采用微软公司的 Visual Studio 2015 作为开发环境，英伟达公司的 CUDA Toolkit 作为 GPU 的开发工具，采用 C++语言编写程序。算例采用四组 IEEE14 节点系统拼接而成，连接节点为 IEEE14 节点中的 Bus9 和 Bus13 引出线。左右两部分采用长输电线路连接，系统接线图如图 10 所示。

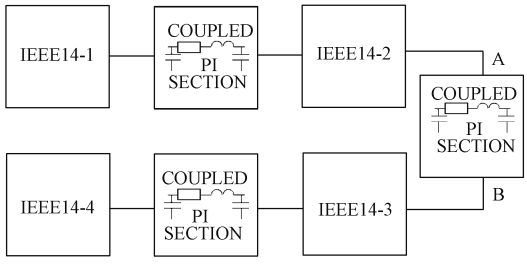

图 10 56 节点系统

Fig. 10 Power system with 56 nodes

4.2 正确性测试

设置故障发生在 IEEE14-2 系统出线 Bus3，分别测量线路 Tline1 两端节点 A 和 B 电压。传统电磁暂态仿真程序仿真步长采用 50μs，仿真时长为 0.5s，基于 GPU 的改进算法程序仿真步长采用 0.5ms，仿真时长为 0.5s。

(1)情形 1。当系统在 A 点 0.2s 时发生单相接地故障，故障持续 0.05s 时，A 点电压波形如图 11 和图 12 所示。

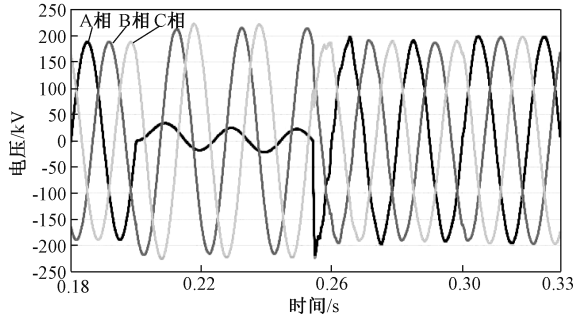

图 11 原始算法结果(情形 1)

Fig. 11 Result of original algorithm (example 1)

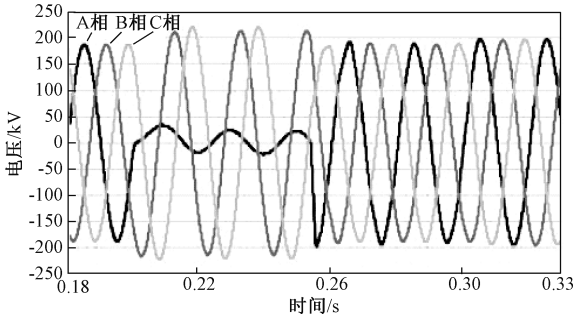

图 12 改进算法结果(情形 1)

Fig. 12 Result of improved algorithm (example 1)

(2)情形 2。当系统在 A 点 0.2s 时发生三相接地故障,故障持续 0.05s 时,A 点电压波形如图 13 和图 14 所示,B 点电压波形如图 15 和图 16 所示。

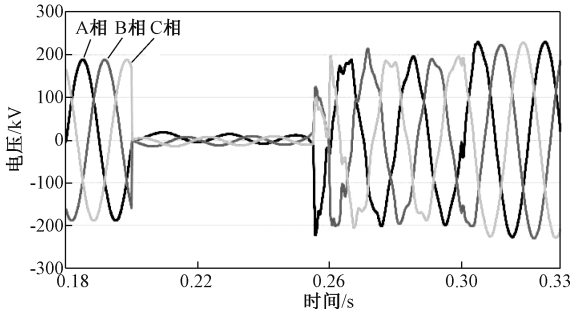

图 13　原始算法结果(情形 2)

Fig. 13　Result of original algorithm (example 2)

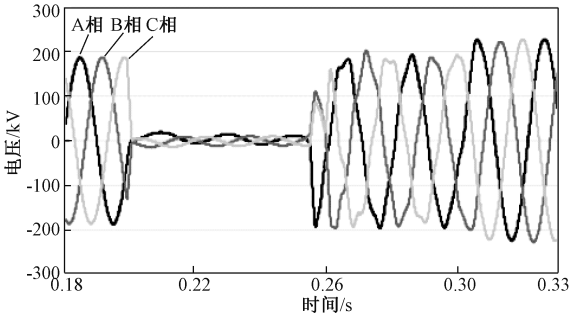

图 14　改进算法结果(情形 2)

Fig. 14　Result of improved algorithm (example 2)

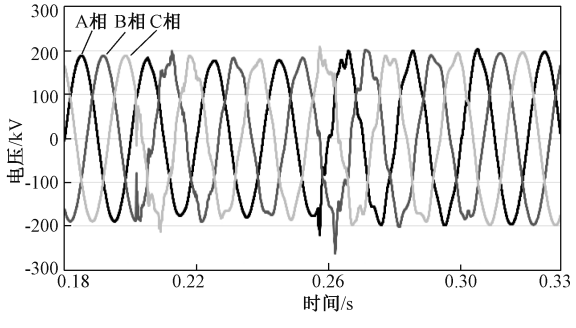

图 15　原始算法结果(B 点)

Fig. 15　Result of original algorithm (point B)

4.3　效率测试

为了测试新算法的高效性,本文以 4.1 节系统为基本单元,以 Bus9 和 Bus13 作为引出端,通过输电线路以串联的方式连接各子系统,分别构成节点数为 56、112、224、448 的系统,各系统执行时间对比如表 1 所示。由表 1 数据可知,当节点数较少时,改进算法效率相对原始算法加速效果不明显,甚至可能效率更低。随着节点数目的增加,改进算法的加速性逐渐体现,当节点数达到 448 时,改进算法可达到 1.5 倍加速效果。

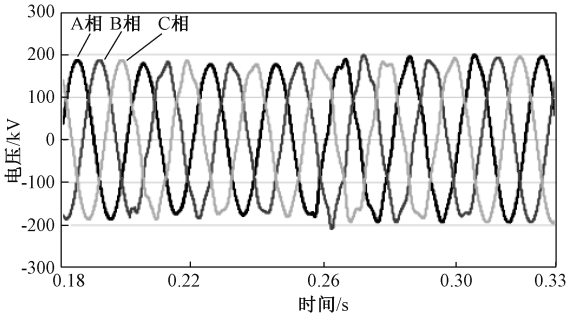

图 16　改进算法结果(B 点)

Fig. 16　Result of improved algorithm (point B)

表 1　算法耗时与加速比

Tab. 1　Algorithm time consumption and speedup ratio

节点数	执行时长/s		加速比
	原始算法	改进算法	
56	4.847	5.513	0.879
112	10.913	10.837	1.007
224	19.369	17.873	1.084
448	85.291	56.562	1.508

5　结论

本文首先对 GPU 的架构特点和编程特点分别作了详细的介绍,通过和 CPU 比较,详细研究了基于 GPU 并行计算的优点。然后结合电磁暂态线性方程组求解特点,提出了基于 GPU 的并行 Crout 求解算法,分别从三个方面对传统算法进行了加速优化。最后,通过实验取得了最高 1.5 倍的加速比。

由于 GPU 的并行加速性适用于大规模的数据,因此理论上将节点数目拓展到上千时,改进算法的加速性能才会完全体现。然而由于实验工作量巨大,研究时间有限,对于如此庞大的数据量处理较为耗时,本文并未做更进一步的效率测试。

参考文献(References):

[1] 王维洲, 刘茜, 但扬清, 等 (Wang Weizhou, Liu Qian, Dan Yangqing, et al.). 大规模新能源接入电网连锁故障预防控制策略研究 (Study of strategy to prevent and control power grid cascading failure connecting large-scale new energy) [J]. 电工电能新技术 (Advanced Technology of Electrical Engineering and Energy), 2015, 34(3): 12-17.

[2] 刘婷婷, 文俊, 乔光尧, 等 (Liu Tingting, Wen Jun, Qiao Guangyao, et al.). 多馈入直流输电系统谐波相

互影响的研究（Study on harmonic effects of multi-infeed HVDC systems）[J]. 电工电能新技术（Advanced Technology of Electrical Engineering and Energy），2016，35（1）：42-47.

[3] 李亚楼，张星，李勇杰，等（Li Yalou，Zhang Xing，Li Yongjie，et al.）. 交直流混联大电网仿真技术现状及面临挑战（Present situation and challenges of AC/DC hybrid large-scale power grid simulation technology）[J]. 电力建设（Electric Power Construction），2015，36（12）：1-8.

[4] Yu Zhitong，Chen Ying，Song Yankan，et al. Comparison of parallel implementations of controls on GPU for transient simulation of power system [A]. Proceedings of the 35th Chinese Control Conference [C]. 2016. 9996-10001.

[5] Jalili-Marandi V，Dinavahi V. SIMD-based large-scale transient stability simulation on the graphics processing unit [J]. IEEE Transactions on Power Systems，2010，25（3）：1589-1599.

[6] Debnath J K，Fung W K，Gole A M，et al. Simulation of large-scale electrical power networks on graphics processing units [A]. IEEE Electrical Power and Energy Conference [C]. 2011. 199-204.

[7] Inoue Y，Asai H. Acceleration of large electromagnetic simulation including non-orthogonally aligned thin structures by using multi-GPU HIE/C-FDTD method [A]. IEEE Electrical Design of Advanced Packaging and Systems Symposium [C]. 2016. 170-173.

[8] Zhao Jinli，Liu Juntao，Li Peng. GPU based parallel matrix exponential algorithm for large scale power system electromagnetic transient simulation [A]. Innovative Smart Grid Technologies [C]. 2016. 110-114.

[9] Zhou Zhiyin，Dinavahi Venkata. Parallel massive thread electromagnetic transient simulation on GPU [J]. IEEE Transactions on Power Delivery，2014，29（3）：1045-1053.

[10] 鄂志君，应迪生，陈家荣，等（E Zhijun，Ying Disheng，Chen Jiarong，et al.）. 动态相量法在电力系统仿真中的应用（Application of dynamic phasor theory in modern power system simulation）[J]. 中国电机工程学报（Proceedings of the CSEE），2008，28（31）：42-47.

[11] Ren Ling，Chen Xiaoming，Wang Yu. Sparse LU factorization for parallel circuit simulation on GPU [A]. Design Automation Conference [C]. 2012. 1125-1130.

[12] Kerr A，Dan C，Richards M. QR decomposition on GPUs [A]. Workshop on General Purpose Processing on Graphics Processing Units [C]. 2009. 71-78.

[13] Tinney W F，Brandwajn V，Chan S M. Sparse vector method [J]. IEEE Transactions on Power Apparatus and Systems，1985，PAS-104（2）：295-301.

[14] Debnath J K，Gole A M，Fung W K. Graphics processing unit based acceleration of electromagnetic transients simulation [J]. IEEE Transactions on Power Delivery，2016，31（5）：2036-2044.

Research on electromagnetic transient parallel simulation based on GPU

YAO Shu-jun[1]，HAN Min-xiao[1]，ZHANG Shuo[1]，LIN Zhi-mao[1]，LI Hao[2]

（1. School of Electrical and Electronic Engineering，North China Electric Power University，Beijing 102206，China；2. Yantai Qixia Power Supply Bureau，State Grid Shandong Electric Power Company，Yantai 265300，China）

Abstract：In the power system，due to the access of a large number of high-voltage direct current（HVDC），new energy，and flexible AC transmission systems，the traditional electromagnetic transient simulation can not meet the simulation requirements of large-scale systems due to the small step size and slow simulation speed. In recent years，graphics processing unit（GPU）has been developing rapidly in the field of high performance computing. It is possible to optimize electromagnetic transient simulation by GPUs so as to improve computational efficiency. After studying the structure of GPU，a parallel LU decomposition algorithm for electromagnetic transient simulation linear equations is proposed，which accelerates the traditional algorithm from three aspects. Compared with the simulation results based on CPU，the superiority of the algorithm is proved.

Key words：electromagnetic transient simulation；graphic processing unit；LU decomposition；parallel algorithm