DOI：10.13334/j.0258-8013.pcsee.172148 **文章编号**：0258-8013 (2018) 15-4329-10 **中图分类号**：TM 744

一种 GPU-CPU 异构运算框架加速的实时 *N*–1 交流潮流计算方法

唐坤杰，董树锋*，宋永华

(浙江大学电气工程学院，浙江省 杭州市 310027)

A Real-time *N*–1 AC Power Flow Calculation Method Based on GPU-CPU Heterogeneous Computing Framework

TANG Kunjie, DONG Shufeng*, SONG Yonghua

(College of Electrical Engineering, Zhejiang University, Hangzhou 310027, Zhejiang Province, China)

ABSTRACT: With the increment of scale of power system, in order to satisfy the increasing real-time and accuracy requirements of *N*–1 security verification, a real-time *N*-1 AC power flow calculation algorithm based on graphics processing unit-central processing unit (GPU-CPU) heterogeneous computing framework was proposed. In the algorithm, a method for solving the *N*–1 power flow problem was designed, which concatenated several independent power flow problems into one. The concatenated Jacobi matrix was generated by parallel processing, and the linear equations were solved by the direct method or the iterative method according to the scale of equations, in which the iterative method was parallelized. The algorithm was divided into CPU processing part and GPU processing part. CPU processed iterative initial value set, node admittance matrix formation, check set formation, iterative value correction and convergence judgement while GPU processed generation of concatenated Jacobi matrix. Correction equations solution was processed by CPU or GPU depended on the scale of equations, in order to achieve fast solution. The case analysis shows that, the proposed algorithm has high efficiency, high accuracy and small storage space. It has advantages compared with traditional *N*–1 power flow algorithm, which can meet the real-time requirement of *N*–1 power flow calculation and has engineering application value.

KEY WORDS: *N*–1 power flow calculation; GPU-CPU heterogeneous computing framework; parallel processing; concatenation algorithm; iterative method

摘要：随着电力系统规模的扩大，为了适应 *N*–1 安全校验日益上升的实时性和精确性的需求，提出一种图形处理单元—中央处理单元(graphics processing unit-central processing unit，GPU-CPU)异构运算框架加速的实时 *N*–1 交流潮流计算方法。算法中设计一种 *N*–1 潮流问题的拼接求解方法，将原本多个独立的潮流问题组合为一个。雅可比矩阵的拼接生成采用并行化处理，线性方程组的求解根据规模大小选择直接法或迭代法处理，其中迭代法采用并行化处理。算法整体分为 CPU 处理部分和 GPU 处理部分，CPU 处理迭代初值的设定、节点导纳矩阵的形成、校验集合的形成、迭代值的修正、收敛性判断等步骤，GPU 处理雅可比矩阵的拼接生成等步骤，修正方程组的求解根据其规模选择 CPU 求解或 GPU 求解，以达到快速求解的目的。算例表明，所提算法效率和精度高、空间占用小，与传统 *N*–1 潮流算法相比具有明显优势，能够满足电网实时 *N*–1 潮流计算的需求，具有工程应用价值。

关键词：*N*–1 潮流计算；GPU-CPU 异构运算架构；并行化；拼接求解；迭代法

0 引言

随着电力系统规模的不断扩大，电力系统静态安全问题日益突出。*N*–1 安全校验是电力系统稳定性分析的重要手段，其实质是在系统中每个单一元件开断的情形下所进行的 *N*–1 潮流计算。由于 *N*–1 安全校验经常作为实时安全分析和辅助决策的手段之一，因此 *N*–1 潮流计算对计算精度和计算速度都有很高的要求。特别是在系统的节点、支路规模较大时，*N*–1 潮流计算需进行大量潮流问题的求解，因此研究快速且精确的 *N*–1 潮流计算方法在电力系统静态安全分析中具有重要的现实意义。

直流潮流法和灵敏度分析法是目前被广泛应用的 *N*–1 潮流计算方法。直流潮流法对潮流模型做了简化，不存在收敛性问题，计算简单。但是，直

基金项目：国家重点研发计划项目(2016YFB0901300)。

Project by National Key Research and Development Program of China (2016YFB0901300).

流潮流法的简化条件也使得其精度较差，特别是在实际网络中节点电压显著偏离额定电压、线路电抗没有显著大于线路电阻、网损较大等情形下，直流潮流法的误差更为严重[1]。为提高直流潮流法的精度，文献[2]利用补偿法优化 *N*–1 直流潮流法模型。文献[3]提出一种改进的直流潮流算法，通过网损迭代得到等值负荷的大小，以提高传统直流法的精度。但是，由于直流潮流法本身的模型做了较大简化，即使通过一定程度的改进，有些线路的计算结果误差仍然比较大，使得直流潮流法在实际应用中存在一定缺陷和局限性。

灵敏度分析法在正常接线和正常运行方式的基础上进行，不必对 *N*–1 情形下的每个潮流问题分别进行迭代求解，计算速度较快。文献[4-5]利用高阶泰勒级数来近似计算线路断开后系统的各个运行变量，但在计算灵敏度高阶导数值的过程中未计算节点电压高阶导数值，因此计算结果精确度不高。文献[6]提出一种新的基于灵敏度–补偿法的开断潮流计算方法，以节点注入功率模拟支路开断影响，计算结果一定程度上能够逼近牛顿–拉夫逊交流潮流法。但是，由于灵敏度分析法本身是一种近似算法，其计算准确度始终存在局限性，在某些对计算精度要求较高的场合不适用。

因此，为了保证 *N*–1 潮流计算的精确性，仍需要采用交流潮流法，但是传统的交流潮流计算方法如牛顿–拉夫逊法等需要反复迭代、形成雅可比矩阵、求解修正方程组，系统规模较大时计算量大，计算速度慢，不能够适应安全分析的实时性要求。

近年来，图形处理单元(graphics processing unit，GPU)得到飞速发展，特别是 GPU 通用计算被应用到诸多计算密集的领域中。图形处理单元–中央处理单元(graphics processing unit-central processing unit，GPU-CPU)异构协同的计算体系是实际工程应用中常用的一种框架，充分利用 CPU 与 GPU 在工作原理及特点上的各自优势，使得整个架构具有强大的并行计算能力、较高的性价比和性能能耗比[7]。在交流潮流法中，稀疏线性方程组求解占据了大部分计算时间，GPU 通用计算特别是 GPU-CPU 异构运算框架通过并行加速稀疏线性方程组的求解，为交流潮流法的加速计算提供实现平台，一些学者利用 GPU 加速已取得初步成果[8-14]。但是，上述研究利用 GPU 主要实现的是线性方程组直接法求解的并行处理，以及一些传统迭代方法的并行化处理如高斯–塞德尔法、共轭梯度法的并行等，对于新的迭代法理论和预处理技术在电力系统潮流计算中的并行化实现还未深入研究。除方程组求解外，交流潮流计算的剩余计算时间主要集中于雅可比矩阵的形成，目前利用 GPU 对雅可比矩阵的形成实现并行加速的研究成果比较有限。

结合以上问题，本文的主要工作包括：

1）在保证计算实时性需求的基础上，用牛顿–拉夫逊交流潮流计算方法提高计算精度，提出一种 GPU-CPU 异构运算框架加速的实时 *N*–1 交流潮流计算方法，对于电网中输电线路、变压器开断的 *N*–1 潮流问题进行快速并行求解。

2）提出一种 *N*–1 潮流问题的拼接求解算法，使得一个系统的所有 *N*–1 潮流问题拼接为一个较大规模的潮流问题，从而只需对这一较大规模的潮流问题进行一次牛顿–拉夫逊法迭代计算即可，有效提高计算效率。

3）本文提出的 *N*–1 潮流问题的拼接求解算法包括了雅可比矩阵的拼接处理和拼接后线性方程组求解的两个关键子步骤。本文利用 GPU 加速设计了一种雅可比矩阵的并行拼接生成方法；同时，根据 Krylov 子空间理论，提出基于不完全 LU 分解预处理迭代法针对拼接后线性方程组规模较大的情形求解，并利用 GPU 实现并行加速处理。

算例测试表明，本文提出的雅可比矩阵的并行拼接生成方法、大规模线性方程组并行迭代求解算法具有显著的加速效果，从而使得基于 GPU-CPU 异构运算框架和潮流问题的拼接求解算法的实时 *N*–1 交流潮流计算方法具有较高的计算效率，满足安全分析和辅助决策的实时性和精确性需求。

1 *N*–1 交流潮流计算

1.1 基本数学模型

本文提出的 *N*–1 交流潮流计算算法基于牛顿–拉夫逊交流潮流法。牛顿–拉夫逊法是常用的交流潮流计算方法。该方法通过逐次线性化，把非线性方程组的求解过程变成了反复求解其相对应的修正线性方程组的过程。每一次迭代过程中，需要求解如下形式的修正方程组：

$$\begin{bmatrix}\Delta \boldsymbol{P}\\ \Delta \boldsymbol{Q}\end{bmatrix}=\boldsymbol{J}\begin{bmatrix}\Delta \boldsymbol{\delta}\\ \Delta \boldsymbol{U}/\boldsymbol{U}\end{bmatrix} \tag{1}$$

式中：$\Delta \boldsymbol{P}$ 和 $\Delta \boldsymbol{Q}$ 为节点有功功率注入和节点无功功率注入修正向量；$\boldsymbol{J}$ 为系统的雅可比矩阵；$\Delta \boldsymbol{\delta}$ 和 $\Delta \boldsymbol{U}$ 为电压相角和电压幅值修正向量；$\boldsymbol{U}$ 为各节点电压的对角矩阵。通过修正方程组求解得到修正量

$\Delta\boldsymbol{\delta}$ 和 $\Delta\boldsymbol{U}$ 后，即可得到新的解，反复迭代直至收敛即可。

1.2　计算加速分析

在 *N*–1 交流潮流计算中，对于一个具有 *N* 个独立元件的系统，*N*–1 全校验需求解 *N* 个交流潮流问题，即需要运用 *N* 次牛顿–拉夫逊法迭代求解。当系统规模较大时，需要做大量次数的牛顿–拉夫逊法迭代，计算效率较低。因此，可以考虑利用适当手段减少运用牛顿–拉夫逊法的次数以提高计算速度。

另一方面，在每一次的牛顿–拉夫逊法计算过程中，运算量最大的步骤主要是修正方程组的求解及雅可比矩阵形成两部分。其中，修正方程组中的雅可比矩阵一般为稀疏矩阵，稀疏线性方程组的求解方法主要包括直接法[15-16]和迭代法[17-18]两种，直接法针对小规模方程组求解具有优势，迭代法则在大规模稀疏线性方程组的求解中计算效率更高[19]，且适合并行加速处理，因此根据矩阵规模选取合适的求解方法非常重要。雅可比矩阵的生成是另一个耗时较长的步骤，可以结合雅可比矩阵结构特点设计并行加速方法减少雅可比矩阵生成的耗时。

2　GPU-CPU 异构运算架构与 CUDA

2.1　GPU-CPU 异构运算架构数据流

如图 1 左侧所示，GPU-CPU 异构运算架构一般由 1 个 CPU 和多个 GPU 组成，CPU 及每个 GPU 有各自独立的内存体系，CPU 无法与各 GPU 显存直接通讯，每个 GPU 也无法与 CPU 内存或其他 GPU 显存直接通讯，CPU 和 GPU 之间的数据传输通过 PCI-Express 总线进行通信。在这一运算框架下，GPU 通用计算程序需要包括三部分[20]：CPU 计算程序、GPU 计算程序和 GPU-CPU 通信程序。

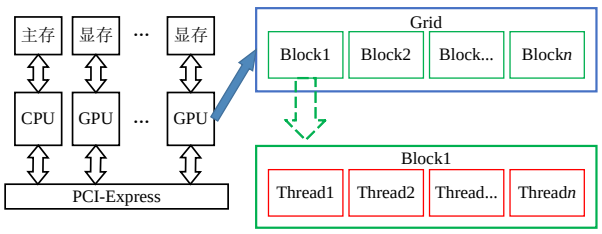

图 1　GPU-CPU 异构运算架构数据流及 CUDA 模型

Fig. 1　Data flow of GPU-CPU heterogeneous computing framework and model of CUDA

单 CPU、多 GPU 的架构能够扩展并行线程数，用于加速处理计算规模非常庞大的问题，在机器学习[21-22]、图像分析[23]等领域得到广泛应用。但是，多 GPU 架构的成本昂贵，数据划分、数据同步、通讯等操作也相对繁琐。

2.2　CUDA 模型及原理

CUDA(compute unified devices architecture)是 NVIDIA 公司推出的一种新的编程模型和指令集架构，用于实现 GPU 通用计算。CUDA 对原先的 C 语言进行扩展，基于这一框架既可以定义普通的 C 语言函数，也可以定义特别的 C 语言函数称为“内核”，即在调用时，并行执行 *n* 次 *n* 个不同的并行线程，而不是像普通的 C 语言函数只执行一次[24]。因此，CUDA 框架为 GPU-CPU 异构运算架构的实现提供了条件。

CUDA 的编程模型可以用图 1 右侧框图刻画。每一个内核函数在执行时占用一个线程网络(grid)，每个线程网络由多个线程块(block)组成，各线程块之间并行执行，线程块之间无法通信。进一步地，每一个线程块由多个线程(thread)组成，各线程之间并行执行[25]。调用内核函数时，通过设定启用的线程块数量及每个线程块中的线程数量，可以使得每个线程对不同数据并行执行相同指令。

3　*N*–1 潮流问题的拼接求解算法

由于牛顿–拉夫逊法是一个具有逻辑性和顺序性的算法，而 GPU 只适合计算量密集但逻辑简单的场合，因此直接使用 GPU 对各独立的潮流问题用牛顿–拉夫逊法进行并行计算是很难行得通的。此外，GPU 的显存有限，GPU 和 CPU 不能直接通讯，将完整的牛顿–拉夫逊法在 GPU 上运行需要开辟大量空间，这是不现实的。基于此，提出 *N*–1 潮流问题的拼接求解算法。

3.1　数学引理

若对于矩阵 $\boldsymbol{A}_1$、$\boldsymbol{A}_2$、……、$\boldsymbol{A}_{n-1}$、$\boldsymbol{A}_n$，向量 $\boldsymbol{x}_1$、$\boldsymbol{x}_2$、……、$\boldsymbol{x}_{n-1}$、$\boldsymbol{x}_n$ 及 $\boldsymbol{b}_1$、$\boldsymbol{b}_2$、……、$\boldsymbol{b}_{n-1}$、$\boldsymbol{b}_n$ 满足 $\boldsymbol{A}_1\boldsymbol{x}_1=\boldsymbol{b}_1$、$\boldsymbol{A}_2\boldsymbol{x}_2=\boldsymbol{b}_2$、……、$\boldsymbol{A}_{n-1}\boldsymbol{x}_{n-1}=\boldsymbol{b}_{n-1}$、$\boldsymbol{A}_n\boldsymbol{x}_n=\boldsymbol{b}_n$，则有：

$$\begin{bmatrix} \boldsymbol{A}_1 & & & & \\ & \boldsymbol{A}_2 & & & \\ & & \ddots & & \\ & & & \boldsymbol{A}_{n-1} & \\ & & & & \boldsymbol{A}_n \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_{n-1} \\ \boldsymbol{x}_n \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \\ \vdots \\ \boldsymbol{b}_{n-1} \\ \boldsymbol{b}_n \end{bmatrix} \tag{2}$$

3.2　算法加速手段

N–1 交流潮流计算需要对于各输电线路、变压器开断情形下分别形成雅可比矩阵、反复迭代求解潮流，在节点和支路规模较大时计算效率低，故提出一种 *N*–1 潮流问题的拼接求解算法，通过减少牛

顿–拉夫逊法运算次数减少计算耗时。

假设一个系统经连通性检查后有 n 条输电线路或变压器支路需要进行 N–1 安全校验。若对于每条支路开断的情形分别进行牛顿–拉夫逊法潮流求解，则对于支路 k 开断的情形做 N–1 牛顿–拉夫逊交流潮流计算时，需要求解线性方程组

$$\begin{bmatrix}\Delta \boldsymbol{P}_{ki}\\ \Delta \boldsymbol{Q}_{ki}\end{bmatrix}=\boldsymbol{J}_{ki}\boldsymbol{X}_{ki},\quad 1\le i\le p_k \tag{3}$$

式中$\Delta \boldsymbol{P}_{ki}$、$\Delta \boldsymbol{Q}_{ki}$、$\boldsymbol{J}_{ki}$、$\boldsymbol{X}_{ki}$分别表示支路 k 开断计算潮流时第 i 次迭代的节点有功功率注入修正向量、节点无功功率注入修正向量、雅可比矩阵、修正方程组的解。潮流问题收敛且迭代次数小于预设的最大迭代次数时，p_k 表示支路 k 开断的情形下牛顿–拉夫逊法计算收敛所需的迭代次数，否则 p_k 等于预设的最大迭代次数。

考虑到牛顿—拉夫逊法计算结果的精度随迭代次数逐渐提高，若对于任意支路开断的情形下的牛顿–拉夫逊法迭代，都设定其迭代次数为 P 为

$$P=\max\{p_1,p_2,\cdots,p_{k-1},p_k\} \tag{4}$$

则对于支路 k 开断的情形下，需要求解线性方程组：

$$\begin{bmatrix}\Delta \boldsymbol{P}_{ki}\\ \Delta \boldsymbol{Q}_{ki}\end{bmatrix}=\boldsymbol{J}_{ki}\boldsymbol{X}_{ki},\quad 1\le i\le P \tag{5}$$

假设这些线性方程组的解为：

$$\boldsymbol{X}_{ki}=\begin{bmatrix}\Delta \boldsymbol{\delta}_{ki}\\ \Delta \boldsymbol{U}_{ki}/\boldsymbol{U}_{ki}\end{bmatrix},\quad 1\le i\le P \tag{6}$$

为了减少 N–1 安全校验过程中牛顿—拉夫逊法的使用次数，考虑将节点功率注入修正向量和雅可比矩阵分别进行拼接。现考虑各支路开断情形下对应的潮流问题的第一次迭代，拼接它们的节点功率注入修正向量、雅可比矩阵，可以形成如下线性方程组：

$$\begin{bmatrix}\Delta \boldsymbol{P}_{11}\\ \Delta \boldsymbol{Q}_{11}\\ \Delta \boldsymbol{P}_{21}\\ \Delta \boldsymbol{Q}_{21}\\ \cdots\\ \cdots\\ \Delta \boldsymbol{P}_{n1}\\ \Delta \boldsymbol{Q}_{n1}\end{bmatrix}=\begin{bmatrix}\boldsymbol{J}_{11} & & & \\ & \boldsymbol{J}_{21} & & \\ & & \cdots & \\ & & & \boldsymbol{J}_{n1}\end{bmatrix}\begin{bmatrix}\boldsymbol{Y}_{11}\\ \boldsymbol{Y}_{21}\\ \cdots\\ \boldsymbol{Y}_{n1}\end{bmatrix} \tag{7}$$

由 3.1 节的数学引理可知，线性方程组的解应为：

$$\boldsymbol{Y}_{k1}=\boldsymbol{X}_{k1}=\begin{bmatrix}\Delta \boldsymbol{\delta}_{k1}\\ \Delta \boldsymbol{U}_{k1}/\boldsymbol{U}_{k1}\end{bmatrix},\quad 1\le k\le n \tag{8}$$

可见，经拼接处理后的线性方程组的解的每一个子向量都与各独立潮流问题中线性方程组的解对应相等。依次类推，拼接处理后的潮流问题只需要使用一次牛顿—拉夫逊法，迭代过程中求解如下线性方程组：

$$\begin{bmatrix}\Delta \boldsymbol{P}_{1i}\\ \Delta \boldsymbol{Q}_{1i}\\ \Delta \boldsymbol{P}_{2i}\\ \Delta \boldsymbol{Q}_{2i}\\ \cdots\\ \cdots\\ \Delta \boldsymbol{P}_{ni}\\ \Delta \boldsymbol{Q}_{ni}\end{bmatrix}=\begin{bmatrix}\boldsymbol{J}_{1i} & & & \\ & \boldsymbol{J}_{2i} & & \\ & & \cdots & \\ & & & \boldsymbol{J}_{ni}\end{bmatrix}\begin{bmatrix}\boldsymbol{Y}_{1i}\\ \boldsymbol{Y}_{2i}\\ \cdots\\ \boldsymbol{Y}_{ni}\end{bmatrix},\ 1\le i\le P \tag{9}$$

由 3.1 节的数学引理可知，线性方程组的解应为

$$\boldsymbol{Y}_{ki}=\begin{bmatrix}\Delta \boldsymbol{\delta}_{ki}\\ \Delta \boldsymbol{U}_{ki}/\boldsymbol{U}_{ki}\end{bmatrix},\quad 1\le i\le P,1\le k\le n \tag{10}$$

由此可见，拼接后的潮流问题仅通过一次牛顿—拉夫逊法求解，可得到与各潮流问题独立使用牛顿—拉夫逊法相同的潮流结果。当然，拼接后的潮流问题的线性方程组规模远大于原来各个独立的潮流问题，当大到一定程度时，如果使用传统的直接法求解如 LU 分解等，会需要耗费较长时间，故考虑结合矩阵预处理、迭代法、并行加速等手段提供支持，在本文第 5 节将详细叙述。6.3 节的算例分析将进一步验证拼接算法较原本独立求解各个独立的潮流问题具有更高的计算效率。

3.3 算法流程

本文所提出的 N–1 潮流问题的拼接求解算法结合 3.2 中的加速手段，基于 CUDA 采用 GPU-CPU 异构运算架构，分为 CPU 处理部分和 GPU 处理部分，CPU 处理迭代初值的设定、节点导纳矩阵的形成、校验集合的形成、迭代值的修正、收敛性判断等步骤，GPU 处理雅可比矩阵的拼接生成等步骤，修正方程组的求解根据其规模选择 CPU 求解或 GPU 求解，以达到快速求解的目的。主要流程如下：

1）CPU：输入系统数据；设定牛顿—拉夫逊法计算精度、最大迭代次数。

2）CPU：形成原始潮流问题的导纳矩阵和第一次迭代时的雅可比矩阵，记录雅可比矩阵中的非零元位置。

3）CPU：对系统中各支路进行连通性检验，形成校验集合 *S*，集合元素个数为 *n*。

4）CPU：设置牛顿–拉夫逊法迭代变量的初值，设置迭代次数 *i*=0。

5）GPU：根据 2）、3）步骤的结果，结合 3.2 节中所提到的方法，根据牛顿–拉夫逊法生成当前迭代步骤下的拼接后的雅可比矩阵、节点功率注入向量，进而得到修正方程组。令

$$\boldsymbol{J}=\mathrm{diag}(\boldsymbol{J}_{1i},\boldsymbol{J}_{2i},\cdots\boldsymbol{J}_{ni}) \tag{11}$$

$$\boldsymbol{X}=[\boldsymbol{Y}_{1i}\quad\boldsymbol{Y}_{2i}\quad\cdots\quad\boldsymbol{Y}_{ni}]^{\mathrm{T}} \tag{12}$$

$$\boldsymbol{b}=[\Delta\boldsymbol{P}_{1i}\ \Delta\boldsymbol{Q}_{1i}\ \Delta\boldsymbol{P}_{2i}\ \Delta\boldsymbol{Q}_{2i}\ \cdots\ \cdots\ \Delta\boldsymbol{P}_{ni}\ \Delta\boldsymbol{Q}_{ni}]^{\mathrm{T}} \tag{13}$$

6）CPU 或 GPU：根据线性方程组规模，利用合适的方式求解。

$$\boldsymbol{JX}=\boldsymbol{b} \tag{14}$$

7）CPU：根据 6）步骤得到的 $\boldsymbol{X}$ 进行牛顿—拉夫逊法的收敛性判断，如满足精度要求，则退出潮流算法，认为算法收敛；否则，对于逐个独立的潮流问题分别进行迭代变量的修正。

8）CPU：令迭代次数 *i*=*i*+1，若已达到预设的最大迭代次数，则退出潮流算法，认为算法不收敛；否则，转至步骤 5）。

4　雅可比矩阵的并行拼接生成

如第 3 节所述，整个系统的 *N*–1 潮流计算只需要进行一次牛顿—拉夫逊法，每一次迭代的过程中，都需要拼接生成雅可比矩阵。雅可比矩阵的生成在潮流计算中是除线性方程组求解外最耗时的步骤，故考虑利用 GPU 并行加速雅可比矩阵的拼接生成。

4.1　可并行性分析

4.1.1　雅可比矩阵的自然并行性

在潮流计算中，极坐标和直角坐标是表示潮流方程的两种基本形式。以极坐标为例，雅可比矩阵可以表示为：

$$\boldsymbol{J}=\begin{bmatrix}\boldsymbol{H} & \boldsymbol{N}\\ \boldsymbol{F} & \boldsymbol{L}\end{bmatrix} \tag{15}$$

其中，雅可比矩阵非对角元素的表达式为：

$$\begin{cases}H_{ij}=-U_iU_j(G_{ij}\sin\delta_{ij}-B_{ij}\cos\delta_{ij})\\ N_{ij}=-U_iU_j(G_{ij}\cos\delta_{ij}+B_{ij}\cos\delta_{ij})\\ F_{ij}=U_iU_j(G_{ij}\cos\delta_{ij}+B_{ij}\cos\delta_{ij})\\ L_{ij}=-U_iU_j(G_{ij}\sin\delta_{ij}-B_{ij}\cos\delta_{ij})\end{cases} \tag{16}$$

对角元素的计算表达式为：

$$\begin{cases}H_{ii}=Q_i+B_{ii}U_i^2\\ N_{ii}=-P_i-G_{ii}U_i^2\\ F_{ii}=-P_i+G_{ii}U_i^2\\ L_{ii}=-Q_i+B_{ii}U_i^2\end{cases} \tag{17}$$

式中：P_i 表示节点 *i* 的有功注入量；Q_i 表示节点 *i* 的无功注入量；U_i 和 U_j 表示节点 *i* 和 *j* 的电压幅值；δ_{ij} 表示节点 *i* 和 *j* 的相角差；G_{ij}、B_{ij} 表示节点 *i* 和 *j* 的互电导、互电纳；G_{ii}、B_{ii} 表示节点 *i* 的自电导、自电纳。

由式(16)、(17)可见，雅可比矩阵中的每个元素都只与导纳矩阵、节点间相角差、节点电压、节点功率注入等参量有关系，任意两个元素之间相互独立，不存在依赖关系，因此雅可比矩阵的生成是自然可并行的。

当节点电压以直流坐标形式表示时，与极坐标形式类似，通过列写雅可比矩阵元素的表达式不难看出，雅可比矩阵的任意两个元素的计算相互独立，不存在依赖关系，故雅可比矩阵的生成同样具有自然可并行性。但是，相比直角坐标形式，极坐标形式表示的潮流计算可以简化误差向量，计算过程中形成的雅可比矩阵阶数更小，拼接处理后的雅可比矩阵阶数相应更小，有利于优化计算效率，故本文算法选用极坐标作为节点电压的表示形式。

4.1.2　不同潮流问题对应雅可比矩阵的可并行性

经连通性检验后形成的校验集合中的任意一条支路(连接节点 *i* 和 *j*)开断时，导纳矩阵节点 *i* 和 *j* 的互电导、互电纳变为 0，节点 *i* 和 *j* 各自的自电导、自电纳发生改变，其余矩阵元素值不变。相应地，在雅可比矩阵中，仅有下标为 *ij*、*ji*、*ii*、*jj* 的元素值发生改变，而整个雅可比矩阵的结构未发生变化，与没有支路开断的原潮流问题的雅可比矩阵结构完全一致。因此不同潮流问题对应的雅可比矩阵之间也可以并行生成。

4.2　并行拼接生成方法

4.2.1　稀疏技术与雅可比矩阵的存储

考虑到雅可比矩阵一般为稀疏矩阵，拼接后的雅可比矩阵则更加稀疏，为了节省存储空间，可以采用稀疏存储技术存储雅可比矩阵。以 IEEE 标准算例 CASE4 为例，原始潮流问题的雅可比矩阵结构如图 2 左侧所示，若以压缩稀疏行(compressed sparse row，CSR)格式[26]存储，则如图 2 右侧所示。

4.2.2　并行生成方法

结合 4.1 节的分析，对拼接后的雅可比矩阵的

行偏移数组与列号、数值数组分别并行生成。

1）行偏移数组的并行生成。

每个线程块计算多个独立潮流问题在拼接后雅可比矩阵的行偏移数组值，每个线程块中连续编号的线程计算不同独立潮流问题同一个位置元素的行偏移值，具体如图 3 所示。设置启用的线块数量等于原始潮流问题雅可比矩阵的阶数，每个线程块启用的线程数量等于校验集合中的元素个数。

2）列号数组和数值数组的并行生成。

每个线程块计算多个独立潮流问题在拼接后雅可比矩阵的列号数组和数值数组，每个线程块中连续编号的线程计算不同独立潮流问题同一个位置元素的数值，具体如图 4 所示。设置启用的线程块数量等于原始潮流问题雅可比矩阵的非零元个数，每个线程块启用的线程数量等于校验集合中的元素个数。在计算过程中，利用开断情形下的雅可比矩阵与原潮流问题的雅可比矩阵具有完全相同的结构这一结论，同时利用原潮流问题雅可比矩阵生成时记录下的非零元位置，可以方便地对开断情形下的列号数组进行推算。数值数组则根据开断支路对导纳矩阵中的相关值进行修正，再利用雅可比矩阵的计算公式可得。

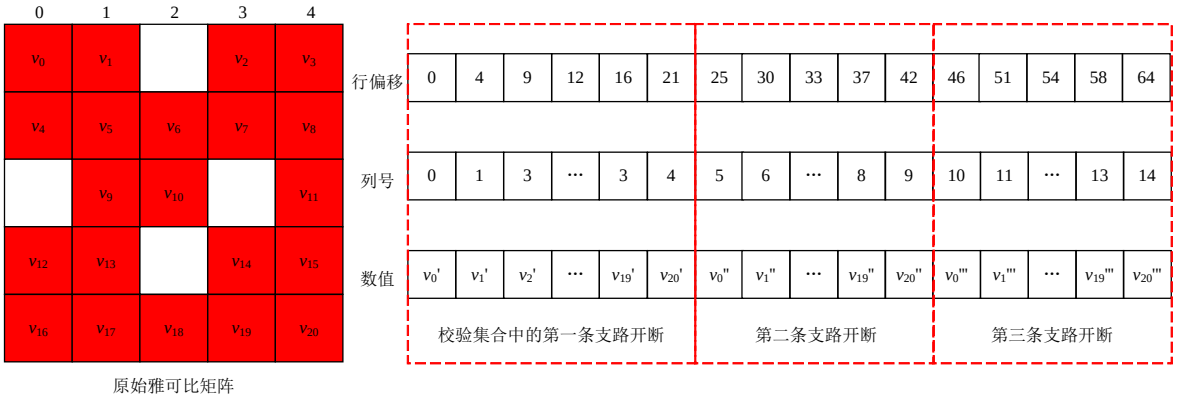

图 2 雅可比矩阵的拼接生成与稀疏矩阵格式存储

Fig. 2 Concatenation and sparse matrix format storage of Jacobi matrix

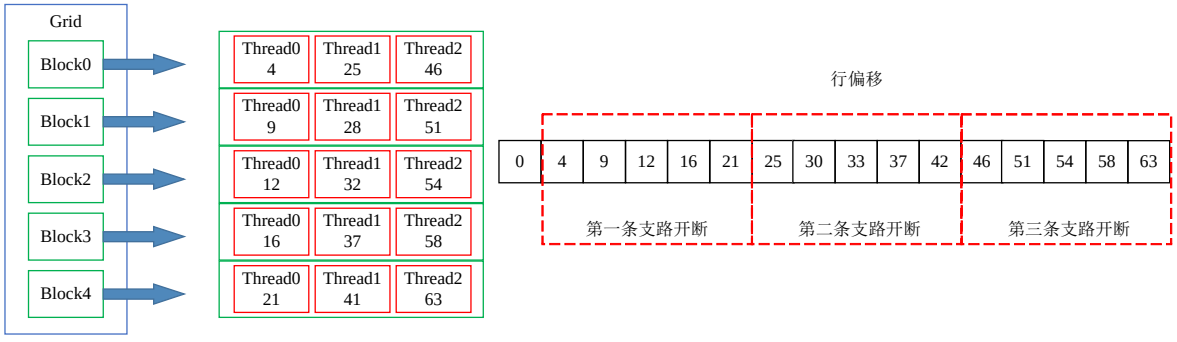

图 3 雅可比矩阵行偏移数组的并行生成

Fig. 3 Parallel generation of row offset array of Jacobi matrix

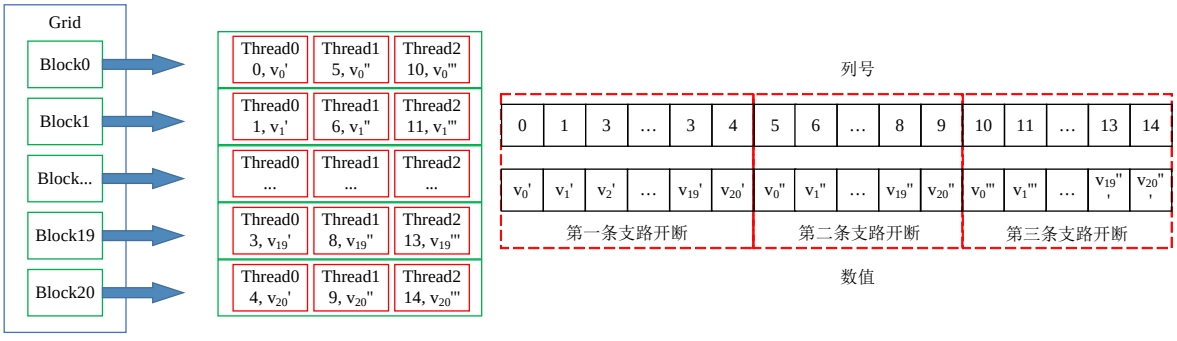

图 4 雅可比矩阵列号数组和数值数组的并行生成

Fig. 4 Parallel generation of column index array and value array of Jacobi matrix

5　线性方程组的加速求解

5.1　线性方程组求解方法的确定

拼接后的潮流问题的线性方程组规模大于原来各个独立的潮流问题。根据拼接后线性方程组的规模，应选用合适的求解方法。线性方程组的求解一般包括直接法和迭代法。

文献[19]根据 Krylov 子空间思想提出一种基于迭代法求解线性方程组的潮流算法，该算法利用不完全 LU 分解作为预处理，采用 GPU-CPU 异构运算架构实现快速求解。这一算法收敛性能稳定、收敛速度快、算法效率高，在系统规模较大时，与传统基于 LU 分解的潮流算法相比具有明显优势。

基于此，本文的算法中根据矩阵规模大小分别利用不同的求解方法进行求解。对于矩阵规模较小的情形，利用基于 SuperLU 求解库的直接法[27]进行求解；对于矩阵规模较大的情况，利用文献[19]中所提出的迭代法进行求解。直接法和迭代法的分界点则需要由算例进一步确定。

5.2　可并行性分析

矩阵规模较大时，除使用预处理技术和迭代法外，还可以结合迭代法特点利用 GPU 进行并行化处理，进一步提高计算效率。基于不完全 LU 分解预处理方法和 Krylov 子空间理论的 BICGSTAB 迭代法中，主要计算形式为矩阵—向量乘法、内积运算等，均具有自然并行性，可以基于 GPU 实现 $\boldsymbol{JX}=\boldsymbol{b}$ 的求解，CPU 向 GPU 移植 $\boldsymbol{J}$、$\boldsymbol{b}$ 矩阵数据，GPU 向 CPU 移植计算得到的 $\boldsymbol{X}$ 矩阵。

6　算例分析

本文中设计的算法在单 GPU 和多 GPU 架构下均能够得到实现。在本文的算例分析部分，以单 CPU、单 GPU 架构为例验证算法的有效性。若要将本文算法在多 GPU 架构下进行实现，需要额外进行数据划分和数据同步工作，即将数据均衡划分到不同 GPU 上进行并行处理，并在合理的时间在 CPU 上进行数据同步，达到进一步加速的目的。

为了分析算法的有效性和优越性，利用 IEEE 标准算例 CASE4 至 CASE300 及 PSS\E 软件自带算例 BENCH(1648 节点)进行测试。编译程序所使用的编译器为 Microsoft Visual Studio 2013 Update 3 与 NVIDIA Nsight Visual Studio Edition，运行在 64bit 的 windows10 操作系统上。测试使用的 CPU 型号为 Intel Core i7-7700K，运行主频 4.20GHz；内存为 32GB；GPU 为 NVIDIA GeForce GTX1080，支持 CUDA8.0 标准。牛顿—拉夫逊法设定最大迭代次数为 10 次，精度为 0.01。线性方程组求解的迭代法精度要求取 1e-6。

6.1　雅可比矩阵并行生成的加速效果分析

为了验证本文提出的雅可比矩阵并行拼接生成方法的效率，与各独立潮流问题的雅可比矩阵依次串行生成作对比，利用 CASE4 至 CASE300 测试两种方法的计算用时。表 1 中的数据由两种方法分别进行 10 次取计算用时的平均值得到。

表 1　雅可比矩阵并行生成的加速效果分析

Tab. 1　Acceleration effect analysis of parallel generation of Jacobi matrix

算例	故障集支路数	串行生成用时/ms	并行生成用时/ms	加速比
CASE4	3	1	<1	—
CASE39	35	2	<1	—
CASE57	79	3	<1	—
CASE118	177	7	1	7
CASE300	320	40	1	40
BENCH	762	531	13	41

由表 1 可知，本文提出的并行生成方法具有显著的加速效果，特别是系统规模较大时，如 CASE300、BENCH，加速比达到 40 倍左右。在 *N*–1 安全校验中，需要多次生成雅可比矩阵，并行生成的加速效果能够产生累积效应，有效提高算法的整体效率。

6.2　线性方程组求解方法的确定及计算用时对比

拼接后线性方程组求解根据其规模选用基于 SuperLU 求解库的直接法或基于不完全 LU 分解预处理方法、Krylov 子空间理论，并利用 GPU 并行化处理的 BICGSTAB 迭代法。直接法和迭代法的分界点结合算例进行确定。图 5 的数据由直接法和迭代法分别对拼接后的潮流问题进行 10 次完整的牛

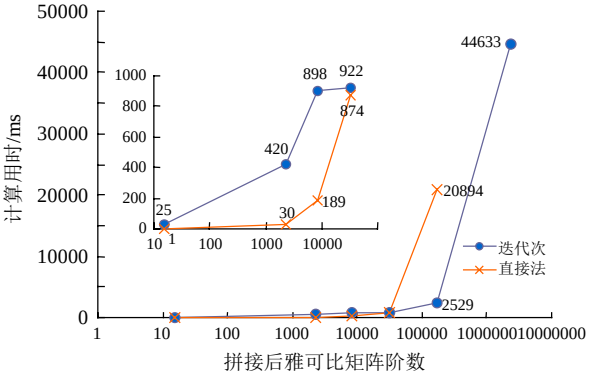

图 5　两种线性方程组求解方法计算用时对比

Fig. 5　Comparison of time consuming of two methods for solving linear equations

顿—拉夫逊法求解并取计算用时的平均值得到。

由图 5 可以看出，在拼接后雅可比矩阵阶数较小(小于 35000 阶)时，使用直接法在运算速度上有优势；当拼接后雅可比矩阵规模较大(大于 35000 阶)时，使用并行化迭代法计算速度更快。并且，拼接后雅可比矩阵规模越大，并行化迭代法优势越明显。因此，不妨取 35000 阶作为使用直接法或并行化迭代法的分界点，可以使得本文算法以更有效率的方式进行拼接后潮流问题的求解。

6.3 算法总体计算效率分析

本文所提的算法对各独立潮流问题进行拼接处理，利用雅可比矩阵并行生成、直接法和迭代法的组合使用提高计算效率。为了验证本文算法总体计算效率，与传统的 *N*–1 潮流计算方法进行计算效率的对比。传统的 *N*–1 潮流计算，即对校验集合中的每一条支路分别开断的情形依次进行潮流计算，传统的潮流计算方法又包括牛顿—拉夫逊法、PQ 分解法、直流潮流法。这里利用 MATPOWER 工具箱分别使用上述三种潮流算法进行对比计算。MATPOWER 工具箱运行稳定、计算速度快、准确度高，是一款非常成熟的电力系统潮流计算工具。

表 2 将本文算法和基于 MATPOWER 牛顿—拉夫逊法、PQ 分解法、直流潮流法的 *N*–1 潮流计算方法进行计算用时、收敛性和精度的对比。其中，精度对比主要针对直流法与牛顿—拉夫逊法进行对比，包括节点电压幅值最大误差Δ*U*、节点电压幅值最大相对误差δ*U*、支路有功功率最大绝对误差Δ*P*。收敛性对比主要针对 PQ 分解法，统计整个 *N*–1 校验过程中牛顿—拉夫逊法收敛但 PQ 分解法不收敛的总次数。表 2 中的计算用时数据由相应方法分别进行 10 次计算取平均值得到。

由表 2 可以看出，本文算法相较于传统的基于 MATPOWER 工具箱中牛顿—拉夫逊法和 PQ 分解法的方法能够实现有效加速，特别是在系统规模较大时，如 BENCH，本文算法的加速效果更加明显，比牛顿—拉夫逊法计算用时节省近 100s，比 PQ 分解法计算用时节省 30 余秒。牛顿—拉夫逊法、PQ 分解法和本文算法可以认为在给定的精度要求下不存在误差。但是，对于部分算例，PQ 分解法会出现不收敛的现象。

表 2 本文算法与传统算法计算用时、收敛性及精度对比

Tab. 2 Comparison of time consuming, convergence and accuracy of traditional methods and the algorithm in this paper

算例	本文算法	牛拉法	PQ 分解法		直流法			
	用时/ms	用时/ms	用时/ms	不收敛次数	用时/ms	Δ*U*/pu	δ*U*/%	Δ*P*/MW
CASE4	1	17	11	0	12	0.10	10.8	11.17
CASE39	30	124	147	0	38	0.08	7.54	62.77
CASE57	189	214	209	1	67	0.39	65.0	20.93
CASE118	874	1122	633	0	187	0.10	10.6	90.07
CASE300	2529	4211	2832	0	668	0.21	26.6	272.15
BENCH	44633	140175	76964	1	15678	0.20	25.2	1010.10

直流法由于对计算模型做了较大简化，故计算用时较少，但是计算精度差，节点电压幅值最大相对误差在 7%~65%，支路有功功率最大绝对误差在系统规模较大时能达到 1000MW 左右，不能够满足 *N*–1 安全校验的精度要求。

由此可见，综合考虑计算用时、收敛性和精度，本文算法相对于传统的 3 类方法具有显著优势。

6.4 内存占用测试

考虑到拼接后的雅可比矩阵的稀疏性，本算法中使用了 CSR 格式存储，且在选用迭代法求解时，迭代计算过程中只需开辟若干一维空间，整体算法内存占用率低。

从表 3 可以看出，系统规模在 1648 节点以内，算法的 RAM 存储占用峰值仅为 1.73GB，GPU 缓存占用峰值仅为 2.48GB，4G 显存的 GPU 即可支持。

表 3 算法内存占用

Tab. 3 Memory footprint of the algorithm

算例	拼接后雅可比矩阵阶数	RAM 存储占用峰值/MB	GPU 缓存占用峰值/MB
CASE4	15	354.1	1606.0
CASE39	2345	354.3	1606.0
CASE57	8374	367.7	1606.0
CASE118	32037	390.0	1606.0
CASE300	169600	590.3	1606.0
BENCH(1648 节点)	2272284	1730.9	2478.0

7 结论

本文提出了一种基于 GPU-CPU 异构运算框架和潮流问题的拼接求解算法的实时 *N*–1 交流潮流

计算方法，所提方法具有以下特点：

1）算法在保证实时性要求的基础上，采用交流潮流法进行 *N*–1 潮流计算，较常用的直流法、灵敏度分析法能取得更高的精度，使得校验分析更具有精确性。

2）算法采用的拼接方法将各独立潮流问题组合求解，求解过程及结果与组合前完全等价，将原本需要多次进行的牛顿—拉夫逊法运算减少为一次，有效提高计算效率。

3）算法采用的雅可比矩阵并行拼接生成方法能够有效实现加速，提高总体算法的计算效率。算法采用的直接法和迭代法组合使用的方式能够根据矩阵规模，选择效率更高的计算方法。

4）算法 RAM 存储占用和 GPU 缓存占用小，普通 PC 机配备的 CPU 和 GPU 即可支持。

算例测试表明，本文所提方法能够显著提高系统 *N*–1 安全校验的速度和精度，能够满足电网实时 *N*–1 潮流计算的需求，具有工程应用的价值。

在未来的研究过程中，本文所提的 GPU-CPU 异构运算框架可以尝试应用到电力系统计算的其他领域中，包括连续开断潮流、最优潮流、状态估计等。同时，在省地县一体化电网背景下，可以考虑利用单 CPU、多 GPU 架构处理计算规模更加庞大的问题，进一步提高计算效率。

参考文献

[1] 赵晋泉，叶君玲，邓勇．直流潮流与交流潮流的对比分析[J]．电网技术，2012，36(10)：147-152．
Zhao Jinquan，Ye Junling，Deng Yong．Comparative analysis on DC power flow and AC power flow[J]．Power System Technology，2012，36(10)：147-152(in Chinese)．

[2] 袁智强，陈宇晨，刘涌，等．改进直流法在静态安全分析中的应用[J]．继电器，2003，31(7)：23-27，80．
Yuan Zhiqiang，Chen Yuchen，Liu Yong，et al．Application of the improved direct current load flow algorithm on static security analysis[J]．Relay，2003，31(7)：23-27，80(in Chinese)．

[3] 王虹富，王毅，高崇，等．基于网损等值负荷模型的直流潮流迭代算法[J]．电力系统自动化，2015，39(1)：99-103．
Wang Hongfu，Wang Yi，Gao Chong，et al．Iterative algorithm of DC power flow based on network loss equivalent load model[J]．Automation of Electric Power Systems，2015，39(1)：99-103(in Chinese)．

[4] 孙鸣，单永梅．补偿法在继电保护整定计算中的应用[J]．电网技术，2003，27(7)：28-31．
Sun Ming，Shan Yongmei．Application of compensation algorithm in setting software for line protection[J]．Power System Technology，2003，27(7)：28-31(in Chinese)．

[5] 容文光，吴政球，匡文凯，等．基于泰勒级数展开的 *N*–1 牛顿拉夫逊法快速潮流修正计算[J]．电网技术，2007，31(2)：42-46．
Rong Wenguang，Wu Zhengqiu，Kuang Wenkai，et al．Taylor series expansion based *N*–1 fast power flow revision calculation using Newton-Raphson method [J]．Power System Technology，2007，31(2)：42-46(in Chinese)．

[6] 杜正旺，哈恒旭，宋扬，等．基于灵敏度–补偿法的电力网络开断潮流新算法[J]．电力系统保护与控制，2010，38(16)：103-107．
Du Zhengwang，Ha Hengxu，Song Yang，et al．New algorithm based on the sensitivity and the compensation methods for line-outage problem of power network [J]．Power System Protection and Control，2010，38(16)：103-107(in Chinese)．

[7] 王海峰，陈庆奎．图形处理器通用计算关键技术研究综述[J]．计算机学报，2013，36(4)：757-772．
Wang Haifeng，Chen Qingkui．General purpose computing of graphics processing unit：A survey[J]．Chinese Journal of Computers，2013，36(4)：757-772(in Chinese)．

[8] Jalili-Marandi V，Zhou Zhiyin，Dinavahi V．Large-scale transient stability simulation of electrical power systems on parallel GPUs[J]．IEEE Transactions on Parallel and Distributed Systems，2012，23(7)：1255-1266．

[9] 陈德扬，李亚楼，江涵，等．基于道路树分层的大电网潮流并行算法及其 GPU 优化实现[J]．电力系统自动化，2014，38(22)：63-69．
Chen Deyang，Li Yalou，Jiang Han，et al．A parallel power flow algorithm for large-scale grid based on stratified path trees and its implementation on GPU[J]．Automation of Electric Power Systems，2014，38(22)：63-69(in Chinese)．

[10] Li Xue，Li Fangxing．GPU-based power flow analysis with Chebyshev preconditioner and conjugate gradient method[J]．Electric Power Systems Research，2014，116：87-93．

[11] Li Xue，Li Fangxing，Clark J M．Exploration of multifrontal method with GPU in power flow computation[C]//Proceedings of the 2013 IEEE Power & Energy Society General Meeting．Vancouver，BC，Canada：IEEE，2013：1-5．

[12] Byun J H，Ravindran A，Mukherjee A，et al．Accelerating the gauss-Seidel power flow solver on a high performance reconfigurable computer[C]//Proceedings of the 17th IEEE Symposium on Field Programmable Custom Computing Machines．Napa，CA，USA：IEEE，2009：227-230．

[13] 杨罡，喻乐，刘明光．基于多轻量线程的电力系统潮流计算加速方法[J]．电网技术，2013，37(6)：1666-1671．
Yang Gang，Yu Le，Liu Mingguang．A method for accelerating power flow calculation based on multiple light threads[J]．Power System Technology，2013，37(6)：1666-1671(in Chinese)．

[14] Singh J，Aruni I．Accelerating power flow studies on graphics processing unit[C]//Proceedings of the 2010 Annual IEEE India Conference．Kolkata，India：IEEE，2011：1-5．

[15] 张永杰，孙秦．大型稀疏线性方程组符号 LU 分解法[C]//2006 航空宇航科学与技术全国博士生学术论坛．北京：国务院学位委员会，2006．
Zhang Yongjie，Sun Qin．Symbol LU decomposition method of large scale sparse linear equations[C]//Aeronautics & Astronautics Aircraft Design & Mechanics Branch Forum．Beijing，China：The Academic Degrees Committee of the State Council，2006(in Chinese)．

[16] 邓兴升，孙虹虹．自适应谱修正 LU 分解法解算高病态法方程[J]．大地测量与地球动力学，2014，34(6)：135-139．
Deng Xingsheng，Sun Honghong．Self-adaptive spectrum correction LU decomposition algorithm for solving a normal equation with severely ill-conditioned matrix [J]．Journal of Geodesy and Geodynamics，2014，34(6)：135-139(in Chinese)．

[17] Hou Guoliang，Wang Li．A generalized iterative method and comparison results using projection techniques for solving linear systems[J]．Applied Mathematics and Computation，2009，215(2)：806-817．

[18] Ahamed A K C，Magoulès F．Iterative methods for sparse linear systems on graphics processing unit[C]//Proceedings of the 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems．Liverpool，UK：IEEE，2012：836-842．

[19] 唐坤杰，董树锋，宋永华．基于不完全 LU 分解预处理迭代法的电力系统潮流算法[J]．中国电机工程学报，2017，37(S1)：55-62．
Tang Kunjie，Dong Shufeng，Song Yonghua．Power flow algorithm based on an iterative method with incomplete LU decomposition preconditioning[J]．Proceedings of the CSEE，2017，37(S1)：55-62(in Chinese)．

[20] 张朝晖，刘俊起，徐勤建．GPU 并行计算技术分析与应用[J]．信息技术，2009(11)：86-89．
Zhang Zhaohui，Liu Junqi，Xu Qinjian．Analysis and application of the GPU parallel computing technology [J]．Information Technology，2009(11)：86-89(in Chinese)．

[21] Li Jinjing，Chen Qingkui，Liu Bocheng．Classification and disease probability prediction via machine learning programming based on multi-GPU cluster MapReduce system[J]．The Journal of Supercomputing，2017，73(5)：1782-1809．

[22] Li Qi，Kecman V，Salman R．A chunking method for Euclidean distance matrix calculation on large dataset using multi-GPU[C]//Proceedings of 9th International Conference on Machine Learning and Applications．Washington，DC，USA：IEEE，2011：208-213．

[23] Pan Yuechao，Wang Yangzihao，Wu Yuduo，et al．Multi-GPU graph analytics[C]//Proceedings of the 2017 IEEE International Parallel and Distributed Processing Symposium．Orlando，FL，USA：IEEE，2017：479-490．

[24] NVIDIA．NVIDIA CUDA compute unified device architecture-Programming guide[M]．Santa Clara：NVIDIA，2007．

[25] Cook S．CUDA 并行程序设计[M]．苏统华，李东，李松泽，等译．北京：机械工业出版社，2014．
Cook S．CUDA programming[M]．Su Tonghua，Li Dong，Li Songze，et al，Trans．Beijing：China Machine Press，2014(in Chinese)．

[26] Greathouse J L，Daga M．Efficient sparse matrix-vector multiplication on GPUs using the CSR storage format[C]//Proceedings of the International Conference for High Performance Computing，Networking，Storage and Analysis．New Orleans，LA，USA：IEEE，2015：769-780．

[27] Li X S，Demmel J，Gilbert J，et al．SuperLU[M]．Boston，MA：Springer，2011．

唐坤杰

收稿日期：2017-10-11。

作者简介：

唐坤杰(1994)，男，博士研究生，研究方向为电力系统高性能计算方法，tangkunjie1994@163.com；

通讯作者：董树锋(1982)，男，博士，副教授，研究方向为状态估计和有源配电网分析等，dongshufeng@zju.edu.cn；

宋永华(1964)，男，博士，教授，研究方向为电网运行与控制和电力市场。

(编辑　邱丽萍)

 DOI：10.13334/j.0258-8013.pcsee.172148

A Real-time *N*–1 AC Power Flow Calculation Method Based on GPU-CPU Heterogeneous Computing Framework

TANG Kunjie, DONG Shufeng, SONG Yonghua

(Zhejiang University)

KEY WORDS: *N*–1 power flow calculation; GPU-CPU heterogeneous computing framework; parallel processing; concatenation algorithm; iterative method

With the expansion of scale of power system, the static security problem is becoming more and more severe. Contingency simulation, especially *N*–1 security is an important means for stability analysis. Since *N*–1 security is used for real-time security analysis and decision-making support, *N*–1 power flow calculation has a high demand for computing accuracy and speed, especially in large-scale system.

In order to satisfy the increasing real-time and accuracy requirements of *N*–1 security verification, a real-time *N*–1 AC power flow calculation algorithm based on GPU-CPU heterogeneous computing framework is proposed.

In the algorithm, a method for solving the *N*–1 power flow problem is designed, which concatenates several independent power flow problems into one. In specific, the node power injection correction vectors and Jacobi matrices can be concatenated respectively. Such concatenated equations need to be solved.

$$\begin{bmatrix} \Delta \boldsymbol{P}_{1i} \\ \Delta \boldsymbol{Q}_{1i} \\ \Delta \boldsymbol{P}_{2i} \\ \Delta \boldsymbol{Q}_{2i} \\ \cdots \\ \cdots \\ \Delta \boldsymbol{P}_{ni} \\ \Delta \boldsymbol{Q}_{ni} \end{bmatrix} = \begin{bmatrix} \boldsymbol{J}_{1i} & & & \\ & \boldsymbol{J}_{2i} & & \\ & & \cdots & \\ & & & \boldsymbol{J}_{ni} \end{bmatrix} \begin{bmatrix} \boldsymbol{Y}_{1i} \\ \boldsymbol{Y}_{2i} \\ \cdots \\ \boldsymbol{Y}_{ni} \end{bmatrix}, \quad 1 \le i \le P \tag{1}$$

According to the linear algebra theory, these equations share the same solution with the independent power flow problems. The concatenated power flow problem needs N-R method solving for only once. Considering that the concatenated Jacobi matrix is a sparse matrix, the sparsity storage technology is used to store the Jacobi matrix to save storage space. In addition, the concatenated Jacobi matrix is formed by parallel processing.

When the scale of the concatenated problem is large, matrix preprocessing technique, iterative method and parallel processing are used to accelerate equations solving. In this paper, an iterative method based on Krylov subspace theory and incomplete LU decomposition preprocessing is used to solve large-scale equations.

The overall algorithm is divided into the CPU processing part and the GPU processing part. CPU processes iterative initial value set, node admittance matrix formation, check set formation, iterative value correction and convergence judgement while GPU processes generation of concatenated Jacobi matrix. Correction equations solution is processed by CPU or GPU depended on the scale of equations, in order to achieve fast solution.

The case analysis shows that, the proposed algorithm has high efficiency, high accuracy and small storage space. It has advantages compared with traditional *N*–1 power flow algorithms, which can meet the real-time requirement of N-1 power flow calculation and have engineering application values.

Tab. 1 Comparison of time consuming and accuracy of traditional methods and the algorithm proposed in this paper

CASE	Algorithm in this paper	Newton-Raphson method	PQ decomposition method				DC method			
	Time/ms	Time/ms	Time/ms	Δ*U*/pu	δ*U*/%	Δ*P*/MW	Time/ms	Δ*U*/pu	δ*U*/%	Δ*P*/MW
CASE4	1	17	11	0	0	0	12	0.10	10.8	11.17
CASE39	30	124	147	0	0	0	38	0.08	7.54	62.77
CASE57	189	214	209	0	0	0	67	0.39	65.0	20.93
CASE118	874	1122	633	0	0	0	187	0.10	10.6	90.07
CASE300	2529	4211	2832	0	0	0	668	0.21	26.6	272.15
BENCH	44633	140175	76964	0.005	0.52	32.68	15678	0.20	25.2	1010.10