基于 CPU-GPU 协同并行内点算法求解结构化非线性规划

杨林峰[1,3],胡桂莉[1,2],张 晨[1],张振荣[3]
(1. 广西大学计算机与电子信息学院,广西南宁 530004;2. 广西电网有限责任公司贵港供电局,广西贵港 537100;
3. 广西多媒体通信与网络技术重点实验室,广西南宁 530004)

摘 要: 大量工程应用问题可建模为结构化非线性规划,且这类问题的系数矩阵可分为稀疏型和稠密型两种类型. 利用原始-对偶内点法(primal dual interior point method,PD-IPM),并结合分布式并行技术可高效求解此类问题. 经典工程问题-机组组合(unit commitment,UC)为稀疏系数矩阵的结构化非线性规划,本文根据 PD-IPM 原理,对 UC 模型进行连续松弛预处理,结合快速解耦技术解耦牛顿修正方程并设计 CPU-GPU 协同并行算法求解子问题,最后将结果与带稠密型子问题的结构化非线性规划的求解结果进行比较和分析. 实验结果显示,本文所设计的算法对于两种不同类型的结构化非线性规划求解均能获得较好的加速比.

关键词: 非线性规划; 内点法; 机组组合; CPU-GPU 协同; 并行计算
中图分类号: TP3-0 **文献标识码:** A **文章编号:** 0372-2112(2019)02-0382-08
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2019.02.018

CPU-GPU Cooperative Parallel Interior Point Method for Structured Nonlinear Programming

YANG Lin-feng[1,3], HU Gui-li[2], ZHANG Chen[1], ZHANG Zhen-rong[3]
(1. *School of Computer, Electronics and Information, Guangxi University, Nanning, Guangxi* 530004, *China*;
2. *Guigang Power Supply Bureau, Guangxi Power Grid, Guigang, Guangxi* 537100, *China*;
3. *Guangxi Key Laboratory of Multimedia Communications and Network Technology, Nanning, Guangxi* 530004, *China*)

Abstract: A lot of practical application problems can be modeled as structured nonlinearprogramming, and these problems always have two type of coefficients matrices: sparse and dense. Combining the principle of primal dual interior point method (PD-IPM) and the distributed parallel technology can solve the problem efficiently. The unit commitment (UC) is a classical engineering problem which can be formulated as a structured nonlinear programming with sparse coefficients matrices. In this paper, according to the PD-IPM principle, the UC model is continuous relaxation preprocessed, and the Newton correction equations are decoupled by using the fast decoupling technique, which can be used to obtain the independent sub problems. Then, a CPU-GPU collaborative parallel method is proposed to solve the sub problems in parallel and the results are compared with the results of structured nonlinear programming with dense sub problem. The experimental results show that the proposed method for solving two different types of structured nonlinear programming has achieved a certain speedup.

Key words: nonlinear programming; interior point method; unit commitment; CPU-GPU cooperative; parallel computing

1 引言

原始-对偶内点法(primal dual interior point method, PD-IPM)是求解非线性规划(nonlinear programming, NLP)的有效方法[1]. 许多实际应用问题的数学模型往往具有特殊结构,基于内点算法框架,并结合问题的特

收稿日期:2017-08-28;修回日期:2018-09-04;责任编辑:马兰英
基金项目:国家自然科学基金(No. 51767003, No. 51407037, No. 61661004);广西自然科学基金(No. 2016GXNSFDA380019);广西电力系统最优化与节能技术重点实验室基金(No. 15-A-01-11)

殊结构,研究更为高效的内点算法是近年来应用领域的研究热点[2,3]. 近年来,随着并行计算硬件平台的快速崛起,并行计算[4]在众多领域均受到了广泛的关注. CPU 由于受到硬件结构的影响,其多核技术的发展受到极大的限制,计算速度提升空间有限[5]. GPU 是一种适用于密集型数据计算的多核并行处理器,其计算单元数量远超 CPU[6]. 因此,使用 GPU 计算提高 NLP 的计算效率逐渐成为应用领域研究热点[7]. GPU 加速在不同研究领域得到广泛应用与实践[8,9]. 文献[10,12]将 GPU 应用到电力系统最优潮流计算、暂态稳定分析以及机组组合经济调度中,均取得了一定的加速比. 文献[13]利用 CPU-GPU 混合并行计算实现稀疏矩阵的 LU 分解,大幅提高了计算效率.

机组组合(Unit Commitment,UC)问题是指在满足电力系统安全经济运行约束下合理地安排机组投入. 该问题常建模为混合整数二次规划,对于大规模问题,现有的串行算法和求解器大多难以快速求解,然而电力系统的运行、控制与维护却要求实时性,UC 问题的快速求解可以保障电力系统安全经济运行. 多年来,如何快速高效地求解 UC 问题,一直是电力系统研究的重点和难点之一[14]. 近年来,利用内点法(interior point method,IPM)求解 UC 组合优化问题得到了广泛的研究[15]. 为提高 IPM 求解 NLP 的效率,本文基于 PD-IPM 提出 CPU-GPU 协同并行求解计及爬坡约束的机组组合连续松弛优化问题算法. 第一步连续松弛 UC 模型;第二步解耦 IPM 的牛顿修正方程;第三步利用 CPU-GPU 分布式并行计算各子问题. 最后,对带稠密型子问题和稀疏型子问题的结构化非线性规划进行比较分析,实验结果表明,本文所设计的算法对于两种不同类型的结构化非线性规划求解均能获得较好的加速比.

2　原始-对偶内点法求解结构化非线性规划的算法框架

2.1　模型求解

文献[17]中的 UC 模型的连续松弛为 NLP 模型. 如下式所示:

$$\begin{aligned}&\min F=\sum_{i=1}^{N}\boldsymbol{f}_i(\boldsymbol{x}_i)\\&\text{s.t.}\begin{cases}\tilde{\boldsymbol{g}}_i(\boldsymbol{x}_i)\leqslant\bar{\boldsymbol{g}}_i\\\tilde{\boldsymbol{g}}_i(\boldsymbol{x}_i)\geqslant\underline{\boldsymbol{g}}_i\\\boldsymbol{h}(\tilde{\boldsymbol{X}})=0\\\boldsymbol{\varphi}(\tilde{\boldsymbol{X}})\geqslant\underline{\boldsymbol{Q}}\end{cases},i=1,2,\cdots,N\end{aligned}\tag{1}$$

式中:i 表示分块变量的序号;$\boldsymbol{x}_i\in\Re^{n_i}$,$n^i$ 表示 $\boldsymbol{x}_i$ 的维数;$\boldsymbol{f}_i(\boldsymbol{x}_i)$表示对应变量 i 的目标函数;N 表示子目标函数的个数;$\tilde{\boldsymbol{g}}_i(\boldsymbol{x}_i)\in\Re^{r_i}$表示上界约束集合;$\tilde{\boldsymbol{g}}_i(\boldsymbol{x}_i)\in\Re^{r_i}$表示下界约束集合;$\bar{\boldsymbol{g}}_i$ 表示约束上界集合;$\underline{\boldsymbol{g}}_i$ 表示约束下界集合;$\boldsymbol{h}(\tilde{\boldsymbol{X}})\in\Re^{r_t}$表示所有变量构成的等式约束集合,$\boldsymbol{\varphi}(\tilde{\boldsymbol{X}})\in\Re^{r_t}$表示所有变量构成的不等式约束集合,$r_i$,$r_t$ 分别表示 $\tilde{\boldsymbol{g}}_i(\boldsymbol{x}_i)$,$\tilde{\boldsymbol{g}}_i(\boldsymbol{x}_i)$,$\boldsymbol{h}(\tilde{\boldsymbol{X}})$,和 $\boldsymbol{\varphi}(\tilde{\boldsymbol{X}})$约束集合所对应的维数,$\tilde{\boldsymbol{X}}=((\boldsymbol{x}_1)^{\mathrm{T}},(\boldsymbol{x}_2)^{\mathrm{T}},\cdots,(\boldsymbol{x}_N)^{\mathrm{T}})^{\mathrm{T}}$ 表示变量的集合.

采用 CPU-GPU 协同并行障碍内点算法求解问题(1),引入松弛变量,将不等式约束转化成等式约束,(其中障碍因子 $\mu>0$)即得到:

$$\begin{aligned}\min F_c=&\sum_{i=1}^{N}f_i(x_i)-\mu\sum_{i=1}^{N}\sum_{j=1}^{r_1}(\ln(q_{i,j}))\\&-\mu\sum_{i=1}^{N}\sum_{j=1}^{r_2}(\ln(v_{i,j}))-\mu\sum_{j=1}^{r_3}\ln(d_j)\\\text{s.t.}&\begin{cases}\tilde{\boldsymbol{g}}_i(\boldsymbol{x}_i)+\boldsymbol{v}_i-\bar{\boldsymbol{g}}_i=0\\\tilde{\boldsymbol{g}}_i(\boldsymbol{x}_i)-\boldsymbol{q}_i-\underline{\boldsymbol{g}}_i=0\\\boldsymbol{h}(\tilde{\boldsymbol{X}})=0\\\boldsymbol{\varphi}(\tilde{\boldsymbol{X}})-\boldsymbol{d}-\underline{\boldsymbol{Q}}=0\\\boldsymbol{v}_i,\boldsymbol{q}_i,d\geqslant0\\i=1,2,\cdots,N\end{cases}\end{aligned}\tag{2}$$

式中:$v_i=(v_{i,1},v_{i,2},v_{i,3},\cdots,v_{i,j})^{\mathrm{T}}$,$q_i=(q_{i,1},q_{i,2},q_{i,3},\cdots,q_{i,j})^{\mathrm{T}}$,$d=(d_1,d_2,d_3,\cdots,d_j)^{\mathrm{T}}$ 为引入的松弛变量集合.

问题(2)对应的拉格朗日函数为:

$$\begin{aligned}L=&\sum_{i=1}^{N}\boldsymbol{f}_i(\boldsymbol{x}_i)-\sum_{i=1}^{N}(\tilde{\boldsymbol{w}}_i)^{\mathrm{T}}[\tilde{\boldsymbol{g}}_i(\boldsymbol{x}_i)+\boldsymbol{v}_i-\boldsymbol{g}_i]-y^{\mathrm{T}}[\boldsymbol{h}(\tilde{\boldsymbol{X}})]\\&-\sum_{i=1}^{N}(\tilde{\boldsymbol{z}}_i)^{\mathrm{T}}[\tilde{\boldsymbol{g}}_i(\boldsymbol{x}_i)-\boldsymbol{q}_i-\underline{\boldsymbol{g}}_i]-\boldsymbol{\varpi}^{\mathrm{T}}[\boldsymbol{\varphi}(\tilde{\boldsymbol{X}})-d-\underline{Q}]\\&-\mu\sum_{i=1}^{N}\sum_{j=1}^{r_1}(\ln(q_{i,j}))-\mu\sum_{i=1}^{N}\sum_{j=1}^{r_2}(\ln(v_{i,j}))-\mu\sum_{j=1}^{r_3}\ln(d_j)\end{aligned}\tag{3}$$

动态耦合约束的拉格朗日乘子 $\boldsymbol{\varpi}$,y 和松弛变量 d 为独立的一组,根据一阶最优性条件,导出式(3)的一阶 KKT 方程:

$$\boldsymbol{K}=((\boldsymbol{K}_1)^{\mathrm{T}},(\boldsymbol{K}_2)^{\mathrm{T}},\cdots,(\boldsymbol{K}_N)^{\mathrm{T}},(\boldsymbol{K}_c)^{\mathrm{T}})^{\mathrm{T}}=0\tag{4}$$

其中:

$$\boldsymbol{K}_i=((\boldsymbol{L}_{x_i})^{\mathrm{T}},(\boldsymbol{L}_{z_i})^{\mathrm{T}},(\boldsymbol{L}_{w_i})^{\mathrm{T}},(\boldsymbol{L}_{q_i})^{\mathrm{T}},(\boldsymbol{L}_{v_i})^{\mathrm{T}})^{\mathrm{T}}$$

$$\boldsymbol{L}_{x_i}=\nabla_{x_i}\boldsymbol{f}_i(\boldsymbol{x}_i)-\nabla_{x_i}\boldsymbol{h}(\tilde{\boldsymbol{X}})\boldsymbol{y}-\nabla_{x_i}\tilde{\boldsymbol{g}}_i(\boldsymbol{x}_i)\tilde{\boldsymbol{w}}_i-\nabla_{x_i}\tilde{\boldsymbol{g}}_i(\boldsymbol{x}_i)\tilde{\boldsymbol{z}}_i-\nabla_{x_i}\boldsymbol{\varphi}(\tilde{\boldsymbol{X}})\boldsymbol{\varpi}$$

$$\boldsymbol{L}_{\tilde{z}_i}=\tilde{\boldsymbol{g}}_i(\boldsymbol{x}_i)-\boldsymbol{q}_i-\underline{\boldsymbol{g}}_i\boldsymbol{L}_{\tilde{w}_i}=\tilde{\boldsymbol{g}}_i(\boldsymbol{x}_i)+\boldsymbol{v}_i-\bar{\boldsymbol{g}}_i$$

$$\boldsymbol{L}_{v_i}^{\mu}=\tilde{\boldsymbol{W}}_i\boldsymbol{V}_i\boldsymbol{e}+\boldsymbol{\mu e};\boldsymbol{L}_{q_i}^{\mu}=\tilde{\boldsymbol{Z}}_i\boldsymbol{Q}_i\boldsymbol{e}-\boldsymbol{\mu e}$$

$$\boldsymbol{K}_c=\begin{pmatrix}\boldsymbol{L}_y\\\boldsymbol{L}_{\boldsymbol{\varpi}}\\\boldsymbol{L}_d^{\mu}\end{pmatrix}=\begin{pmatrix}\boldsymbol{h}(\tilde{\boldsymbol{X}})\\\boldsymbol{\varphi}(\tilde{\boldsymbol{X}})-\boldsymbol{d}-\bar{\boldsymbol{X}}\\\bar{\boldsymbol{W}}\boldsymbol{D}\boldsymbol{e}-\boldsymbol{\mu e}\end{pmatrix}$$

采用牛顿迭代法求解式(4),为了方便讨论引入 M 符号代替原先的具体方程表达式,得到修正方程如下:

$$\begin{pmatrix} \boldsymbol{M}_1 & 0 & \cdots & 0 & (\boldsymbol{M}_{0,1})^{\mathrm{T}} \\ 0 & \boldsymbol{M}_2 & \cdots & 0 & (\boldsymbol{M}_{0,2})^{\mathrm{T}} \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \cdots & \boldsymbol{M}_N & (\boldsymbol{M}_{0,N})^{\mathrm{T}} \\ (\boldsymbol{M}_{0,1}) & (\boldsymbol{M}_{0,2}) & \cdots & (\boldsymbol{M}_{0,N}) & (\boldsymbol{M}_0) \end{pmatrix} \begin{pmatrix} \Delta\boldsymbol{\chi}_1 \\ \Delta\boldsymbol{\chi}_2 \\ \vdots \\ \Delta\boldsymbol{\chi}_N \\ \Delta\boldsymbol{\chi}_c \end{pmatrix} = -\begin{pmatrix} \boldsymbol{K}_1 \\ \boldsymbol{K}_2 \\ \vdots \\ K_c \end{pmatrix} \tag{5}$$

简记为：

$$\boldsymbol{M}(\cdot)\Delta\boldsymbol{X} = -\boldsymbol{K} \tag{6}$$

其中：$\tilde{\boldsymbol{w}}_i \leq 0, \tilde{\boldsymbol{z}}_i \geq 0, \boldsymbol{\varpi} \geq 0$，是对 UC 模型进行预处理之后重新构造的拉格朗日函数的乘子变量，$\boldsymbol{v}_i \geq 0, \boldsymbol{q}_i \geq 0, \boldsymbol{d} \geq 0$ 是对应的互补松弛项；$\boldsymbol{e} = (1,1,\cdots,1)^{\mathrm{T}}$；$\tilde{\boldsymbol{W}}_i, \tilde{\boldsymbol{Z}}_i, \overline{\boldsymbol{W}}, \boldsymbol{V}_i, \boldsymbol{Q}_i$ 和 D 分别是以 $\tilde{\boldsymbol{w}}_i, \tilde{\boldsymbol{z}}_i, \boldsymbol{\varpi}, \boldsymbol{v}_i, \boldsymbol{q}_i, \boldsymbol{d}$ 为对角线元素所形成的矩阵；$\boldsymbol{\chi}_c = (\boldsymbol{d}^{\mathrm{T}}, \boldsymbol{\varpi}^{\mathrm{T}}, \boldsymbol{y}^{\mathrm{T}})^{\mathrm{T}}, \boldsymbol{\chi}_i = ((\boldsymbol{x}_i)^{\mathrm{T}}, (\tilde{\boldsymbol{z}}_i)^{\mathrm{T}}, (\tilde{\boldsymbol{w}}_i)^{\mathrm{T}}, (\boldsymbol{q}_i)^{\mathrm{T}}, (\boldsymbol{v}_i)^{\mathrm{T}})^{\mathrm{T}}$. 其中变量的增量 $\Delta\hat{\boldsymbol{\chi}}_i$ 和 $\Delta\hat{\boldsymbol{\chi}}_c$ 可依次类推. 附录 A 详细描述了 2 个机组修正方程(6)的具体表达形式.

通过修正方程(6)得到变量下降方向，计算(7)可以获得每次迭代的原始和对偶步长，根据式(8)得到 IPM 的最大步长，最后根据式(9)更新变量，完成一次迭代.

$$\begin{cases} step_P = 0.9995\min\left\{ \left.\dfrac{-q_{i,j}}{\Delta q_{i,j}}\right|_{\Delta q_{i,j}<0}, \left.\dfrac{-v_{i,j}}{\Delta v_{i,j}}\right|_{\Delta v_{i,j}<0}, \left.\dfrac{-d_j}{\Delta d_j}\right|_{\Delta d_j<0}, 1 \right\} \\ step_d = 0.9995\min\left\{ \left.\dfrac{-\tilde{z}_{i,j}}{\Delta\tilde{z}_{i,j}}\right|_{\Delta\tilde{z}_{i,j}<0}, \left.\dfrac{-\tilde{w}_{i,j}}{\Delta\tilde{w}_{i,j}}\right|_{\Delta\tilde{w}_{i,j}>0}, \left.\dfrac{-\varpi_j}{\Delta\varpi_j}\right|_{\Delta\varpi_j<0}, 1 \right\} \end{cases} \tag{7}$$

$$step = \min\{step_P, step_d\} \tag{8}$$

$$\begin{pmatrix} \boldsymbol{x}_i \\ \tilde{\boldsymbol{z}}_i \\ \tilde{\boldsymbol{w}}_i \\ \boldsymbol{q}_i \\ \boldsymbol{v}_i \\ \boldsymbol{d} \\ \boldsymbol{\varpi} \\ \boldsymbol{y} \end{pmatrix}^{k+1} = \begin{pmatrix} \boldsymbol{x}_i \\ \tilde{\boldsymbol{z}}_i \\ \tilde{\boldsymbol{w}}_i \\ \boldsymbol{q}_i \\ \boldsymbol{v}_i \\ \boldsymbol{d} \\ \boldsymbol{\varpi} \\ \boldsymbol{y} \end{pmatrix}^{k} + step \begin{pmatrix} \Delta\boldsymbol{x}_i \\ \Delta\tilde{\boldsymbol{z}}_i \\ \Delta\tilde{\boldsymbol{w}}_i \\ \Delta\boldsymbol{q}_i \\ \Delta\boldsymbol{v}_i \\ \Delta\boldsymbol{d} \\ \Delta\boldsymbol{\varpi} \\ \Delta\boldsymbol{y} \end{pmatrix} \tag{9}$$

$$C_{Gap} = \tilde{\boldsymbol{z}}_i' \cdot \boldsymbol{q}_i - \tilde{\boldsymbol{w}}_i' \cdot \boldsymbol{v}_i + \boldsymbol{\varpi}' \cdot \boldsymbol{d} \tag{10}$$

最后，利用式(10)计算算法对偶间隙，根据所求的对偶间隙判断当前解的精度是否满足求解精度要求，若满足，则输出当前的解，停机；否则，继续迭代计算.

3 基于 CPU－GPU 协同并行内点求解 UC 问题

3.1 NLP 模型修正方程解耦

由 2.1 节模型求解可知，基于 PD-IMP 求解 UC 问题时，主要计算量集中于修正方程(5)的求解. 因此，本文设计 CPU-GPU 协同并行方法求解式(5). 将大规模牛顿修正方程分解为 N 个小分块并行求解，实现内点法并行求解结构化非线性规划. 基于文献[16]的快速解耦技术，将式(5)按机组进行解耦，得到如下式子：

$$\begin{aligned} \Delta\chi_1 &= -\boldsymbol{M}_1^{-1}[K_1 + M_{(0,1)}^{\mathrm{T}} \cdot \Delta\chi_c] \\ \Delta\chi_2 &= -\boldsymbol{M}_2^{-1}[K_2 + M_{(0,2)}^{\mathrm{T}} \cdot \Delta\chi_c] \\ \Delta\chi_3 &= -\boldsymbol{M}_3^{-1}[K_3 + M_{(0,3)}^{\mathrm{T}} \cdot \Delta\chi_c] \\ &\vdots \\ \Delta\chi_N &= -M_N^{-1}[K_N + M_{(0,N)}^{\mathrm{T}} \cdot \Delta\chi_c] \end{aligned} \tag{11}$$

展开修正方程(5)末行，并带入式(11)，化简得式(12)

$$\begin{aligned} &[-M_0 + M_{(0,1)} \cdot M_1^{-1} \cdot M_{(0,1)}^{\mathrm{T}} + M_{(0,2)} \cdot M_2^{-1} \cdot M_{(0,2)}^{\mathrm{T}} + \\ &\cdots + M_{(0,N)} \cdot M_N^{-1} \cdot M_{(0,N)}^{\mathrm{T}}] \cdot \Delta\chi_c - K_c + M_{(0,1)} \cdot M_1^{-1} \cdot K_1 \\ &+ M_{(0,2)} \cdot M_2^{-1} \cdot K_2 + \cdots + M_{(0,N)} \cdot M_N^{-1} \cdot K_N = 0 \end{aligned} \tag{12}$$

为了方便讨论，将式(12)简写为：

$$\begin{aligned} (-M_0 + JJ) \cdot \Delta\chi_c &= BB \\ JJ &= \sum_{i=1}^{N} M_{(0,i)} \cdot M_i^{-1} \cdot M_{(0,i)}^{\mathrm{T}} \\ BB &= K_c - CC \\ CC &= \sum_{i=1}^{N} M_{(0,i)} \cdot M_i^{-1} \cdot K_i \end{aligned} \tag{13}$$

从式(13)可知，解耦后的 JJ 和 CC 只与对应下标 i 有关，可并行求解，将计算结果回代到原方程求出 $\Delta\chi_c$，然后将 $\Delta\chi_c$ 回代到式(11)，即可并行求解 $\Delta\chi_i$，得到新迭代点.

3.2 CPU-GPU 协同并行计算

根据文献[16]解耦修正方程(5)，其对角线上各子问题相互独立，通过将并行计算式(13)得到的 $\Delta\chi_c$ 回代到各解耦子问题，并分配多个线程求解 $\Delta\chi_i$，即通过开启多个线程块同时求解该方程组，如图 1 所示.

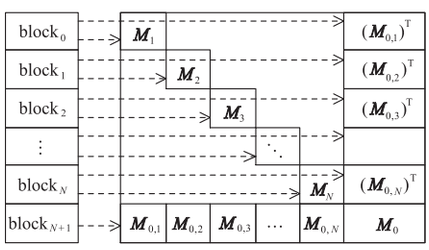


图1 GPU中的任务分配

图 1 显示，若修正方程解耦后计算都由 GPU 完成，(1)增加数据间传输通信，(2)闲置 CPU 宝贵计算资源. 本文使用 CPU-GPU 协同并行求解，即修正方程前 N 个求解任务，使用 CPU 并行操传输给 GPU，GPU 使用线程并行计算求解后再将求解结果并行传回 CPU 进行全局协调. 本文使用共享存储方式提升数据访问速度.

3.3 改进的高斯-约旦算法

本文基于文献[18]设计矩阵组的改进高斯-约旦算法. 第 1 步，根据输入数据创建二维单元矩阵，并构造单个矩阵的增广矩阵 $CC = (A|E)$；第 2 步，当迭代到第 R_i

($i=1,2,\cdots n$)行时:如果主元值 $a_{ii}=0$,则从矩阵中的第 i 列挑选出最大元素所在行,将其加到第 i 行上,如果 $a_{ii}\neq 0$ 执行第 3 步;第 3 步,执行式(14)将主元值化为 1;第 4 步:根据式(15)将主元列上非主元数消减为 0,得到增广矩阵 CC'(如式(16)所示). 依次执行 n 次步骤 2 至步骤 4,得到其逆矩阵. 附录 B 给出了利用 GPU 线程执行算法步骤 3 和 4 的详细描述.

$$R_i \leftarrow R_i / a_{ii} \tag{14}$$

$$R_i \leftarrow R_i - R_j \times a_{ij} \tag{15}$$

$$CC' = \left[\begin{array}{cccc|cccc} 1 & a_{12}/a_{11} & \cdots & a_{1n}/a_{11} & 1/a_{11} & 0 & \cdots & 0 \\ 0 & a_{22}-a_{12}\times a_{21}/a_{11} & \cdots & a_{2n}-a_{1n}\times a_{21}/a_{11} & -a_{21}/a_{11} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & a_{n2}-a_{12}\times a_{n1}/a_{11} & \cdots & a_{nn}-a_{1n}\times a_{n1}/a_{11} & -a_{n1}/a_{11} & 0 & \cdots & 1 \end{array}\right] \tag{16}$$

每一列的消元操作通过独立的线程来并行执行,共需要$(n-1)\times n$ 个线程进行并行处理,直到将原矩阵 A 变成单位矩阵为止,得到所求的逆矩阵 A^{-1}.

3.4　GPU 线程分配

本文利用 GPU 的高斯-约旦算法并行求解逆矩阵. 在算法每轮计算主元后将矩阵按维数划分分块,按照线程块的每个 ID 索引分布式并行消元. 如图 2 所示,展示了使用 GPU 的 N 个线程块并行处理 n 行矩阵情况下的任务分解:

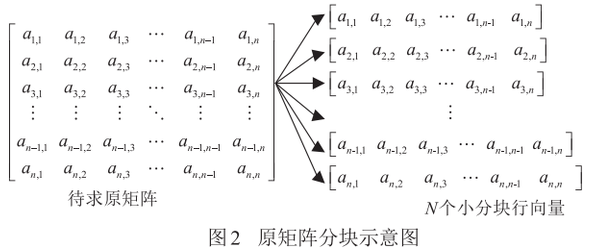


图2　原矩阵分块示意图

基于改进高斯-约旦消元法,循环进行 n 次消元,每次一列,线程 $T_{x,y}$(x 为线程块索引,y 为线程块内线程索引)并行消元. 为了提升该方法性能,仅 $x<y$ 时进行求解. 以消元第 i 列为例:

$$A_{x,y}=A_{i,y}\times p[i],I_{x,y}=I_{i,y}\times p[i]$$

计算时 x 使用 $x=\text{blockIdx.}\,x*\text{blockDim.}\,x+\text{threadIdx.}\,x$;$y=\text{blockIdx.}\,y*\text{blockDim.}\,y+\text{threadIdx.}\,y$.

4　数值实验

实验的硬件平台是:CPU 为 Intel Core 四核处理器,主频为 3.6GHz,内存 4GB;NVIDIA 显卡 GTX980Ti(显存 6GB,核心频率:1190MHz,显存位宽:384bit,显存频率:7010MHz,显存带宽:336.5GB/s). 实验的软件平台:Windows 7 旗舰版,matlab 2014a,CUDA 7.5.

4.1　机组组合求解

经预处理后 UC 模型约束为 $15TN+3T$,其中,T 表示时段数,N 表示机组数,那么以 10 机组 24 时段为例($T=24$,$N=10$),约束个数为 3672. 经过牛顿拉夫森迭代法求解得到的修正方程系数矩阵 $\boldsymbol{M}(\cdot)$ 的维数为 $34TN+3T$,其中每个对角线上的分块矩阵 $\boldsymbol{M}_i$ 均为方阵,$\boldsymbol{M}_i$ 矩阵的个数为 N,而它的维数是 $34T$,$M(\cdot)$ 的右、下边上的子矩阵维数可根据对角块矩阵维数计算. 为验证所提算法的有效性,本节通过在不同的机组数或时段数下分别进行 50 次迭代来比较它们各自的运行时间. 比较的三种方法为:串行计算,记为“串行”;单独使用 MATLAB 的 PCT 并行工具箱在 CPU 上并行计算,记为“PCT 并行”;本文设计的 CPU-GPU 协同并行计算,记为“CPU-GPU 协同”.

表 1　10 机组不同时段的运行结果比较

时段数	约束数量	$\boldsymbol{M}_i$ 维数	运行方式	耗时(s)	加速比
4	612	136	串行	0.66	1.0000
			PCT 并行	1.89	0.3492
			CPU-GPU 协同	0.65	1.0154
6	918	204	串行	0.98	1.0000
			PCT 并行	2.66	0.4537
			CPU-GPU 协同	0.90	1.0889
12	1836	408	串行	2.32	1.0000
			PCT 并行	2.83	0.8198
			CPU-GPU 协同	2.13	1.0892
24	3672	816	串行	6.23	1.0000
			PCT 并行	5.19	1.2004
			CPU-GPU 协同	4.28	1.4556
48	7344	1632	串行	18.75	1.0000
			PCT 并行	11.96	1.5677
			CPU-GPU 协同	10.23	**1.8328**

表 1 ~ 表 4 分别给出了 10 – 50 机 4 个系统 4-48 时段各算例的计算结果. 表 1 ~ 表 4 的实验结果表明,当时段 T 越大,则 $\boldsymbol{M}_i$ 的维数越大,同时非零元素所占据的比例也越小,即矩阵越稀疏. 对于这样带有结构化的稀疏型非线性优化问题,PCT 并行求解时,若问题规模较小,则并行计算所节省的时间开支相比于通信开支还要小. 例如,表 1 显示,当机组数 $N=10$,时段数 $T=4,6,12$ 时,就出现了 PCT 并行计算时间长于串行计算时间. 因此,出现了 PCT 并行计算加速比小于 1 的结

果. 当所求问题的规模增大时,PCT 的并行计算与串行执行之间的时间差逐渐增大,则加速比也慢慢增大. 与此同时,使用本节所设计的 CPU-GPU 协同并行计算时,当问题规模较小时,协同运行比串行执行节约的时间不明显. 但随着问题规模增大,或问题个数的增多,CPU-GPU 协同并行计算的加速比显著上升.

表 2 20 机组不同时段的运行结果比较

时段数	约束数量	M_i 维数	运行方式	耗时(s)	加速比
4	1212	136	串行	1.24	1.0000
			PCT 并行	2.39	0.5188
			CPU-GPU 协同	1.21	1.0248
6	1818	204	串行	1.89	1.0000
			PCT 并行	2.83	0.6678
			CPU-GPU 协同	1.78	1.0618
12	3636	408	串行	4.22	1.0000
			PCT 并行	4.07	1.0369
			CPU-GPU 协同	3.59	1.1754
24	7272	816	串行	11.66	1.0000
			PCT 并行	8.87	1.3145
			CPU-GPU 协同	7.33	1.5907
48	14544	1632	串行	36.55	1.0000
			PCT 并行	22.67	1.6123
			CPU-GPU 协同	19.88	**1.8385**

表 3 30 机组不同时段的运行结果比较

时段数	约束数量	M_i 维数	运行方式	耗时(s)	加速比
4	1812	136	串行	1.83	1.0000
			PCT 并行	2.98	0.6140
			CPU-GPU 协同	1.78	1.0281
6	2718	204	串行	2.83	1.0000
			PCT 并行	3.52	0.8040
			CPU-GPU 协同	2.73	1.0366
12	5436	408	串行	6.37	1.0000
			PCT 并行	5.34	1.1929
			CPU-GPU 协同	4.63	1.3758
24	10872	816	串行	17.96	1.0000
			PCT 并行	12.35	1.4543
			CPU-GPU 协同	9.43	1.9046
48	21744	1632	串行	55.11	1.0000
			PCT 并行	30.62	1.7998
			CPU-GPU 协同	25.63	**2.1502**

表表 4 50 机组的不同时段

时段数	约束数量	M_i 维数	运行方式	耗时(s)	加速比
4	3012	136	串行	3.09	1.0000
			PCT 并行	3.71	0.8329
			CPU-GPU 协同	3.00	1.0300
6	4518	204	串行	4.77	1.0000
			PCT 并行	4.68	1.0192
			CPU-GPU 协同	4.26	1.1197
12	9036	408	串行	10.71	1.0000
			PCT 并行	7.91	1.3540
			CPU-GPU 协同	6.83	1.5681
24	18072	816	串行	29.13	1.0000
			PCT 并行	19.12	1.5235
			CPU-GPU 协同	13.86	2.1018
48	36144	1632	串行	91.49	1.0000
			PCT 并行	50.69	1.8049
			CPU-GPU 协同	42.00	**2.1783**

4.2 具有稠密子块的问题

在机组组合问题中,约束集合所对应的系数矩阵是非常稀疏的对角带边型,属于结构化二次规划问题的典型代表. 而为进一步说明 CPU-GPU 协同并行求解与机组组合问题具有类似结构且带有稠密型分块子问题的其他应用问题求解效果,本节将分别利用 CPU 串行和 CPU-GPU 协同并行求解带稠密型子问题的结构化非线性规划. 其中,对于带稠密型子问题在不同问题规模下所对应的约束条件数量和修正方程对角线上的分块子矩阵 M_i 的维数,可以参照机组组合结构得到,当 T 取值比较小时,对应的 M_i 维数也较低. 在实际的其他应用问题中,对角线上的分块子矩阵 M_i 的维数不尽相同,而本节所进行的实验是采用与机组组合具有相同结构的带稠密型对角分块矩阵 $M_i=34T$. 串行求解和 CPU-GPU 协同并行求解带稠密型对角线上分块矩阵 M_i 的计算结果如表 5 和表 6 所示.

从表 5 和表 6 可以看出,在求解带稠密子矩阵的 NLP 问题时,随着问题规模增大,CPU-GPU 协同并行方法加速比显著提高. 实验结果显示,CPU-GPU 协同并行求解带稠密型对角线上分块矩阵 M_i 的 *NLP* 问题相比于求解带稀疏型子矩阵的问题所得到的加速效果更为明显,其主要原因是带稠密型子问题的数值计算量更大,更能发挥出 GPU 多计算单元的优势. 在 M_i 的个数为 20,M_i 的维数为 2448 时,加速比达到了 4.2570,取得了较好的加速比结果.

4.3 大规模子块的问题

为了测试改进高斯-约旦算法在子矩阵数量较大情况下,算法加速比是否会受通信开销影响而导致算法加速效果变差,本节主要针对不同数量规模和不同维

数的矩阵进行细粒度并行求解其逆矩阵的数值测试,其结果列于表 7 和表 8. 从表 7 与 8 可知,当矩阵维数为不超过 100 时,由于并行求解时产生的通信等时间开支相比并行运算所节省的时间要多,因此在矩阵的维数较小时加速比小于 1. 在矩阵的维数相同时,随着矩阵的个数增多,CPU 串行高斯-约旦算法求解多个矩阵的逆运算所需时间的增加的速度显著大于 GPU 并行高斯-约旦算法,所以加速比逐渐增大. 500 维 1000 个矩阵和 500 维 5000 个矩阵求逆计算的结果显示,算法加速比没有因此减弱,且由 3.31 增加到 3.94. 其主要原因有以下两点:(1)算法使用 GPU 共享内存提升了数据访问速度;(2)算法计算矩阵元素个数由原先$(2n)^2$减小为n^2. 实验结果说明,本文使用的矩阵求逆计算在矩阵数量较大情况下,其加速效果并未受通信开销影响,所以加速效果更加明显,该方法适合于大规模矩阵逆运算.

表 5　M_i 个数为 10 的带稠密型子问题不同规模下的运行结果

约束数量	M_i 维数	运行方式	耗时(s)	加速比
600	136	串行 CPU-GPU 协同	0.72 0.68	1.0000 1.0588
918	204	串行 CPU-GPU 协同	1.38 0.93	1.0000 1.4839
1836	408	串行 CPU-GPU 协同	5.43 2.72	1.0000 1.9853
3672	816	串行 CPU-GPU 协同	24.06 10.32	1.0000 2.3314
7344	1632	串行 CPU-GPU 协同	138.09 41.41	1.0000 3.3347
24696	2448	串行 CPU-GPU 协同	388.39 96.30	1.0000 **4.0331**

表 6　M_i 个数为 20 的带稠密型子问题不同规模下的运行结果

约束数量	M_i 维数	运行的方式	耗时(s)	加速比
1212	136	串行 CPU-GPU 协同	1.37 1.28	1.0000 1.0703
1818	204	串行 CPU-GPU 协同	2.61 1.59	1.0000 1.6415
3636	408	串行 CPU-GPU 协同	10.72 5.20	1.0000 2.0615
7272	816	串行 CPU-GPU 协同	42.90 17.64	1.0000 2.4320
14544	1632	串行 CPU-GPU 协同	272.46 81.13	1.0000 3.4002
49176	2448	串行 CPU-GPU 协同	758.17 178.10	1.0000 **4.2570**

表 7　1000 个矩阵基于并行与串行高斯-约旦法的结果比较

编号	矩阵维数	高斯-约旦法(s)	并行高斯约旦法(s)	加速比
1	50	0.14	0.64	0.2161
2	100	0.91	1.33	0.6874
3	200	7.13	3.60	1.9801
4	500	117.93	35.55	3.3168

表 8　5000 个矩阵基于并行与串行高斯-约旦法的结果比较

编号	矩阵维数	高斯-约旦法(s)	并行高斯约旦法(s)	加速比
1	50	0.61	3.08	0.1996
2	100	4.58	6.23	0.7352
3	200	35.72	18.93	1.8869
4	500	770.66	195.16	3.9489

4.4　改进高斯-约旦算法性能分析

本节从理论与数值结果两方面分析算法性能.

理论分析:(1)根据上述改进高斯-约旦算法与 GPU 线程分配两节内容可知,当 GPU 可用线程总数大于矩阵元素n^2时,算法时间复杂度为$O(n)$. (2)本文使用 GPU 共享内存,提高数据读取速度. GPU 默认共享内存大小为 48KB,共享内存小,单精度存储矩阵维数不能超过 12288. (3)本文默认每个 block 处理矩阵单行数据,GPU 单线程块默认线程数为 1024,因此,矩阵维数超过 1024 时需将矩阵进行垂直划分,影响算法求解效率.

数值结果对比:(1)文献[19]求解 1024 维逆矩阵时算法加速比为 4.88,本文在 20 个 2448 维矩阵并行求逆下,加速比达到了 4.25. 文献[19]求矩阵组逆时,必然增加算法通信时间,影响加速比. 另外,在同样矩阵规模下,本文处理增广矩阵元素与文献[19]相比减少了 1 半,提高了求解效率. (2)文献[11]的内点半定规划 GPU 并行算法,在 20 机组 24 时段时加速比为 1.1,在同等问题规模下本文获得的加速比为 1.59. 此外,由于文献[11]使用的分解本身具有串行性质,加速比有一定限制,实验结果显示在小规模如 10 机 24 时段时,求解速度反而比串行慢. 本文的 CPU-GPU 协同并行算法在小规模问题的求解上仍然具有加速效果. 当问题规模增加时,串行通信时间所占比相对减少,加速效果明显. (3)cublas 方法在矩阵组规模与矩阵维数(100 ~ 200 维)偏小时,求解效果略优于本文方法,但在计算超过 200 维 200 个矩阵组时,由于 getrfbatched( ) 和 getriBatched( ) 两个函数的启动开销大,消耗内存,导致无法计算,而本文方法仍然能高效求解.

5　结论

本文基于原始-对偶内点法原理,通过预处理结构

化非线性规划,利用快速解耦技术将原问题之间的耦合关系进行分离,从而得到相互独立的各个子问题,并对各子问题进行并行求解. 本文通过设计 CPU-GPU 协同并行内点算法对带稠密型和稀疏型子问题的结构化非线性规划分别进行求解,以验证所提方法在求解结构化非线性规划问题上的有效性和高效性. 实验结果表明,针对这两种不同类型的应用问题,本文所设计的算法均得到了一定的加速比,尤其是对于带稠密型子问题的结构化非线性规划取得了较好加速比. 因此,本文所设计的 CPU-GPU 协同并行内点算法为探索大规模结构化非线性规划问题的高效求解提供了新思路.

附录 A 两机组修正方程具体表达形式

下面以问题(1)的 NLP 模型为例,当 $N=2$,$\tilde{X}=(x_1,x_2)$时,对修正方程表达形式进行详细说明. 这里采用牛顿迭代法求解式(4),得到如式(5)的分块带边箭型结构的牛顿修正方程,其详细表达式如图 3 所示.

$$
\begin{bmatrix}
\boldsymbol{H}_1(\cdot) & -\nabla_{x_1}\underset{\sim}{\boldsymbol{g}}_i(\boldsymbol{x}_1) & -\nabla_{x_1}\tilde{\boldsymbol{g}}_1(\boldsymbol{x}_1) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\nabla_{x_1}h(\tilde{\boldsymbol{X}}) & -\nabla_{x_1}\boldsymbol{\varphi}(\tilde{\boldsymbol{X}}) & 0 \\
\nabla_{x_1}^{\mathrm{T}}\underset{\sim}{\boldsymbol{g}}_1(\boldsymbol{x}_1) & 0 & 0 & -\boldsymbol{E}_{q_1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\nabla_{x_1}^{\mathrm{T}}\tilde{\boldsymbol{g}}_1(\boldsymbol{x}_1) & 0 & 0 & 0 & \boldsymbol{E}_{v_1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \boldsymbol{Q}_1 & 0 & \tilde{\boldsymbol{Z}}_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \boldsymbol{V}_1 & 0 & \tilde{\boldsymbol{W}}_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \boldsymbol{H}_2(\cdot) & -\nabla_{x_2}\underset{\sim}{\boldsymbol{g}}_2(\boldsymbol{x}_2) & -\nabla_{x_2}\tilde{\boldsymbol{g}}_2(\boldsymbol{x}_2) & 0 & 0 & -\nabla_{x_2}h(\tilde{\boldsymbol{X}}) & -\nabla_{x_2}\boldsymbol{\varphi}(\tilde{\boldsymbol{X}}) & 0 \\
0 & 0 & 0 & 0 & 0 & \nabla_{x_2}^{\mathrm{T}}\tilde{\boldsymbol{g}}_2(\boldsymbol{x}_2) & 0 & 0 & -\boldsymbol{E}_{q_2} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \nabla_{x_2}^{\mathrm{T}}\tilde{\boldsymbol{g}}_2(\boldsymbol{x}_2) & 0 & 0 & 0 & -\boldsymbol{E}_{v_2} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \boldsymbol{Q}_2 & 0 & \tilde{\boldsymbol{Z}}_2 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \boldsymbol{V}_2 & 0 & \tilde{\boldsymbol{W}}_2 & 0 & 0 & 0 \\
\nabla_{x_1}^{\mathrm{T}}\boldsymbol{h}(\tilde{\boldsymbol{X}}) & 0 & 0 & 0 & 0 & \nabla_{x_2}^{\mathrm{T}}\boldsymbol{h}(\tilde{\boldsymbol{X}}) & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\nabla_{x_1}^{\mathrm{T}}\boldsymbol{\varphi}(\tilde{\boldsymbol{X}}) & 0 & 0 & 0 & 0 & \nabla_{x_2}^{\mathrm{T}}\boldsymbol{\varphi}(\tilde{\boldsymbol{X}}) & 0 & 0 & 0 & 0 & 0 & 0 & -\boldsymbol{E}_d \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \boldsymbol{D} & \overline{\boldsymbol{W}} & 0
\end{bmatrix}
\begin{bmatrix}
\Delta\boldsymbol{x}_1 \\ \Delta\tilde{\boldsymbol{z}}_1 \\ \Delta\tilde{\boldsymbol{w}}_1 \\ \Delta\boldsymbol{q}_1 \\ \Delta\boldsymbol{v}_1 \\ \Delta\boldsymbol{x}_2 \\ \Delta\tilde{\boldsymbol{z}}_2 \\ \Delta\tilde{\boldsymbol{w}}_2 \\ \Delta\boldsymbol{q}_2 \\ \Delta\boldsymbol{v}_2 \\ \Delta\boldsymbol{y} \\ \Delta\boldsymbol{\varpi} \\ \Delta\boldsymbol{d}
\end{bmatrix}
= -
\begin{bmatrix}
\boldsymbol{L}_{x_1} \\ \boldsymbol{L}_{z_1} \\ \boldsymbol{L}_{w_1} \\ \boldsymbol{L}_{q_1} \\ \boldsymbol{L}_{v_1} \\ \boldsymbol{L}_{x_2} \\ \boldsymbol{L}_{z_2} \\ \boldsymbol{L}_{w_2} \\ \boldsymbol{L}_{q_2} \\ \boldsymbol{L}_{v_2} \\ \boldsymbol{L}_{y} \\ \boldsymbol{L}_{\varpi} \\ \boldsymbol{L}_{d}
\end{bmatrix}
$$

图 3 两机组修正方程具体表达式

其中:$\boldsymbol{H}_1(\cdot)=\nabla_{\boldsymbol{x}_1}^2\boldsymbol{f}_1(\boldsymbol{x}_1)-\nabla_{\boldsymbol{x}_1}^2\boldsymbol{h}(\tilde{\boldsymbol{X}})y-\nabla_{\boldsymbol{x}_1}^2\tilde{\boldsymbol{g}}_1(\boldsymbol{x}_1)\tilde{\boldsymbol{w}}_1-\nabla_{x_1}^2\tilde{\boldsymbol{g}}_1(\boldsymbol{x}_1)\tilde{\boldsymbol{z}}_1$,$\boldsymbol{H}_2(\cdot)$表达形式可类似获得;$\nabla_{\boldsymbol{x}_1}^2\boldsymbol{f}_1(\boldsymbol{x}_1)$表示关于变量 $\boldsymbol{x}_1$ 的目标函数对 $\boldsymbol{x}_1$ 的二次偏导数,$\nabla_{x_1}\underset{\sim}{\boldsymbol{g}}_1(\boldsymbol{x}_1)$表示对应下界类约束方程对 $\boldsymbol{x}_1$ 的一阶偏导数;$\boldsymbol{E}_{v_1}$表示对 $\boldsymbol{v}_1$ 求偏导数得到单位矩阵,$\tilde{\boldsymbol{Z}}_1$ 表示以 $\tilde{\boldsymbol{z}}_1$ 变量为对角线上的元素形成的矩阵,其他符号可根据以上符号定义类似获得其意义.

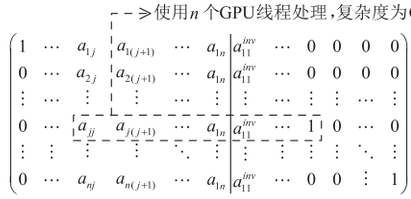


图4 第3步线程计算过程

附录 B GPU 线程计算第 3 步与第 4 步

改进高斯-约旦算法步骤 3 针对一行矩阵元素生成 n 个 GPU 线程并行处理该行中每个元素,如图 4 所示. 而步骤 4 则生成 $n\times(n-1)$ 个线程处理剩余矩阵元素,将 a_{jj} 列所对应的行元素利用行列式计算转换为 0,如图 5 所示. 由图 4 和图 5 可知,基于 GPU 的改进高斯-约旦算法单次循环计算的时间复杂度为 $O(1)$,一个 n 维的矩阵需要重复执行 n 次图 4 和图 5 所示的计算步骤,才能得到其逆矩阵,因此其时间复杂度为 $O(n)$.

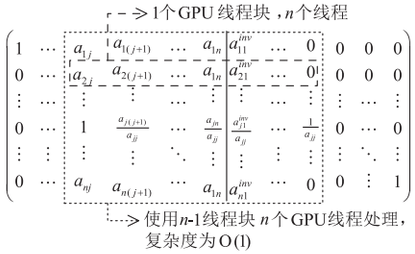


图 5 第 4 步线程计算过程

参考文献

[1] Byrd R H, Hribar M E, Nocedal J. An interior point algorithm for large-scale nonlinear programming[J]. SIAM Journal on Optimization, 1999, 9(4): 877 – 900.

[2] 张立峰,刘昭麟,田沛. 基于压缩感知的电容层析成像图像重建算法[J]. 电子学报,2017,45(2):353 – 358.
Zhang L F, Liu Z L, Tian P. Imagereconstruction algorithm for electrical capacitance tomography based on compressed sensing[J]. Acta Electronica Sinica, 2017, 45(2): 353 – 358. (in Chinese)

[3] 李旭超,宋博. 原始-对偶模型的牛顿迭代原理与图像恢

复[J]. 电子学报,2015,43(10):1984 – 1993.
Li X C, Song B. Newton iterative principle of primal-dual model and image restoration[J]. Acta Electronica Sinica, 2015,43(10):1984 – 1993. (in Chinese)

[4] 卢风顺,宋君强,银福康,等. CPU/GPU 协同并行计算研究综述[J]. 计算机科学,2011,38(3):5 – 9.
Lu F S, Song J B, Yin F K, et al. Survey of CPU/GPU synergetic parallel computing[J]. Computer Science,2011, 38(3):5 – 9. (in Chinese)

[5] 吴恩华. 图形处理器用于通用计算的技术、现状及其挑战[J]. 软件学报,2004,15(10):1493 – 1504.
Wu E H. State of the art and future challenge on general purpose computation by graphics processing unit[J]. Journal of Software,2004,15(10):1493 – 1504. (in Chinese)

[6] 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 C H, Liu J Q, Xu Q J. Analysis and application of the GPU parallel computing technology[J]. Information Technology,2009,(11):86 – 89. (in Chinese)

[7] 巨涛,等. 异构众核系统及其编程模型与性能优化技术研究综述[J]. 电子学报,2015,43(1):111 – 119.
Ju T, et al. The feature programming model and performance optimization strategy of heterogeneous many – core system: a review[J]. Acta Electronica Sinica, 2015, 43(1):111 – 119. (in Chinese)

[8] Chang Y S, Wei J Z, Zhao G Y, et al. A novel architecture of special arithmetic function unit for area-efficient programmable vertex shader[J]. Chinese Journal of Electronics,2013,22(3):483 – 488.

[9] Ravie C, Ali M. Enhancing the simulation of membrane system on the GPU for the N-Queens problem[J]. Chinese Journal of Electronics,2015,24(4):740 – 743.

[10] 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 D Y, Li Y L, Jiang H, 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)

[11] 张宁宇,高山,赵欣. 基于 GPU 的机电暂态仿真细粒度并行算法[J]. 电力系统自动化,2012,36(9):54 – 60.
Zhang N Y, Gao S, Zhao X. A fine granularity parallel algorithm for electromechanical transient stability simulation based on graphic processing unit[J]. Automation of Electric Power Systems. 2012,36(9):54 – 60. (in Chinese)

[12] Alili M V, Dinavahi V. SIMD – based large – scale transient stability simulation on the graphics processing unit [J]. IEEE Trans on Power Systems,2010,25(3):1589 – 1599.

[13] Yang G W. A highly efficient GPU-CPU hybrid parallel implementation of sparse LU factorization[J]. Chinese Journal of Electronics,2012,21(1):7 – 12.

[14] 李文沅. 电力系统安全经济运行——模型与方法[M]. 重庆:重庆大学出版社出版,1989. 147 – 169.

[15] Mehrtash M, et al. An interior point optimization method for stochastic security – constrained unit commitment in the presence of plug – in electric vehicles[J]. Journal of Applied Sciences,2016,16:189 – 200.

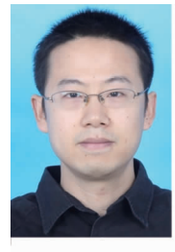
[16] 简金宝,杨林峰,全然. 基于改进多中心校正解耦内点法的动态最优潮流并行算法[J]. 电工技术学报,2012,27(6):232 – 241.
Jian J B, Yang L F, Quan R. Parallel algorithm of dynamic optimal power flow based on improved multiple centrality corrections decoupling interior point method[J]. Transactions of China Electrotechnical Society,2012,27(6):232 – 241. (in Chinese)

[17] Yang L F, Jian J B, Wang Y Y, et al. Projected mixed integer programming formulations for unit commitment problem[J]. International Journal of Electrical Power & Energy Systems,2015,68(68):195 – 202.

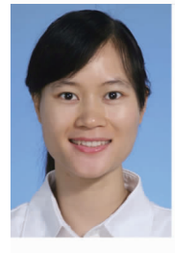
[18] Sharma G, Agarwala A, Bhattacharya B. A fast parallel Gauss Jordan algorithm for matrix inversion using CUDA [J]. Computers & Structures,2013,128:31 – 37.

[19] 刘丽,沈杰,李洪林. 基于 GPU 的矩阵求逆性能测试和分析[J]. 华东理工大学学报,2010,36(6):812 – 817.
Liu L, Shen J, Li H L. Performance testing and analysis for matrix inversion based on GPU[J]. Journal of East China University of Science and Technology, 2010, 36(6):812 – 817. (in Chinese)

作者简介



杨林峰(通信作者)　男,1979 年 12 月生于湖北省武汉市. 博士,广西大学计算机与电子信息学院教授,硕士生导师,主要研究方向为并行与分布式优化算法及其应用.
E-mail:ylf@ gxu. edu. cn



胡桂莉　女,1990 年 8 月生于广西贵港市,广西大学计算机与电子信息学院硕士研究生,主要研究方向为计算机网络与并行分布式计算.
E-mail:794453997@ qq. com