Parallel Numerical Algorithms

Chapter 6 – Structured and Low Rank Matrices
Section 6.3 – Numerical Optimization

Michael T. Heath and Edgar Solomonik

Department of Computer Science
University of Illinois at Urbana-Champaign

CS 554 / CSE 512

Outline

1. General Nonlinear Optimization
   - Nonlinear Equations
   - Optimization
2. Matrix Completion
   - Alternating Least Squares
   - Coordinate Descent
   - Gradient Descent

Nonlinear Equations

Potential sources of parallelism in solving nonlinear equation $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$ include

- Evaluation of function $\boldsymbol{f}$ and its derivatives in parallel
- Parallel implementation of linear algebra computations (e.g., solving linear system in Newton-like methods)
- Simultaneous exploration of different regions via multiple starting points (e.g., if many solutions are sought or convergence is difficult to achieve)

Optimization

Sources of parallelism in optimization problems include

- Evaluation of objective and constraint functions and their derivatives in parallel
- Parallel implementation of linear algebra computations (e.g., solving linear system in Newton-like methods)
- Simultaneous exploration of different regions via multiple starting points (e.g., if global optimum is sought or convergence is difficult to achieve)
- Multi-directional searches in direct search methods
- Decomposition methods for structured problems, such as linear, quadratic, or separable programming

Nonlinear Optimization Methods

- Goal is to minimize objective function $f(\boldsymbol{x})$
- Gradient-based (first-order) methods compute

$$\boldsymbol{x}^{(s+1)} = \boldsymbol{x}^{(s)} - \alpha \nabla f(\boldsymbol{x}^{(s)})$$

- Newton's method (second-order) computes

$$\boldsymbol{x}^{(s+1)} = \boldsymbol{x}^{(s)} - \boldsymbol{H}_f(\boldsymbol{x}^{(s)})^{-1} \nabla f(\boldsymbol{x}^{(s)})$$

- Alternating methods fix a subset of variables $\boldsymbol{x}_1$ at a time and minimize (via one of above two methods)

$$g^{(s)}(\boldsymbol{x}_2) = f\left(\begin{bmatrix} \boldsymbol{x}_1^{(s)} \\ \boldsymbol{x}_2 \end{bmatrix}\right)$$

- Subgradient methods such as stochastic gradient descent, assume $f(\boldsymbol{x}^{(s)}) = \sum_{i=1}^{n} f_i(\boldsymbol{x}^{(s)})$ and compute

$$\boldsymbol{x}^{(s+1)} = \boldsymbol{x}^{(s)} - \eta \nabla f_i(\boldsymbol{x}^{(s)}) \quad \text{for} \quad i \in \{1, \cdots, n\}$$

Parallelism in Nonlinear Optimization

- In gradient-based methods, parallelism is generally found within calculation of $\nabla f(\boldsymbol{x}^{(s)})$, line optimization (if any) to compute α, and the vector sum $\boldsymbol{x}^{(s)} - \alpha \nabla f(\boldsymbol{x}^{(s)})$
- Newton's method main source of parallelism is linear solve
- Alternating methods often fix $\boldsymbol{x}_1$ so that $g^{(s)}(\boldsymbol{x}_2)$ may be decomposed into multiple independent problems

$$g^{(s)}(\boldsymbol{w}) = g_1^{(s)}(\boldsymbol{w}_1) + \cdots + g_k^{(s)}(\boldsymbol{w}_k)$$

- Subgradient methods exploit the fact that subgradients may be independent, since $\nabla f_i(\boldsymbol{x}^{(s)})$ is generally mostly zero and depends on subset of elements in $\boldsymbol{x}^{(s)}$
- Approximate/randomized nature of subgradient methods can permit chaotic/asynchronous optimization

Optimization Case-Study: Matrix Completion

Given a subset of entries

$$\Omega \subseteq \{1, \ldots, m\} \times \{1, \ldots, n\}$$

of the entries of matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, seek rank-$k$ approximation

$$\underset{\boldsymbol{W} \in \mathbb{R}^{m \times k}, \boldsymbol{H} \in \mathbb{R}^{n \times k}}{\operatorname{argmin}} \sum_{(i,j) \in \Omega} \underbrace{\left(a_{ij} - \sum_{l} w_{il} h_{jl} \right)}_{(\boldsymbol{A} - \boldsymbol{W}\boldsymbol{H}^T)_{ij}}^2 + \lambda(||\boldsymbol{W}||_F^2 + ||\boldsymbol{H}||_F^2)$$

- Problems of these type studied in sparse approximation
- Ω may be randomly selected sample subset
- Methods for this problem are typical of numerical optimization and machine learning

Alternating Least Squares

Alternating least squares (ALS) fixes $\boldsymbol{W}$ and solves for $\boldsymbol{H}$ then vice versa until convergence

- Each step improves approximation, convergence to a minimum expected given satisfactory starting guess
- We have a quadratic optimization problem

$$\underset{\boldsymbol{W}\in\mathbb{R}^{m\times k}}{\operatorname{argmin}} \sum_{(i,j)\in\Omega} \left(a_{ij} - \sum_{l} w_{il}h_{jl}\right)^2 + \lambda||\boldsymbol{W}||_F^2$$

- The optimization problem is independent for rows of $\boldsymbol{W}$
- Letting $\boldsymbol{w}_i = w_{i\star}$, $\boldsymbol{h}_i = h_{i\star}$, $\Omega_i = \{j : (i,j) \in \Omega\}$, seek

$$\underset{\boldsymbol{w}_i\in\mathbb{R}^{k}}{\operatorname{argmin}} \sum_{j\in\Omega_i} \left(a_{ij} - \boldsymbol{w}_i\boldsymbol{h}_j^T\right)^2 + \lambda||\boldsymbol{w}_i||_2^2$$

ALS: Quadratic Optimization

Seek minimizer $\boldsymbol{w}_i$ for quadratic vector equation

$$f(\boldsymbol{w}_i) = \sum_{j \in \Omega_i} \left(a_{ij} - \boldsymbol{w}_i \boldsymbol{h}_j^T \right)^2 + \lambda ||\boldsymbol{w}_i||^2$$

- Differentiating with respect to $\boldsymbol{w}_i$ gives

$$\frac{\partial f(\boldsymbol{w}_i)}{\partial \boldsymbol{w}_i} = 2 \sum_{j \in \Omega_i} \boldsymbol{h}_j^T \left(\boldsymbol{w}_i \boldsymbol{h}_j^T - a_{ij} \right) + 2\lambda \boldsymbol{w}_i = 0$$

- Rotating $\boldsymbol{w}_i \boldsymbol{h}_j^T = \boldsymbol{h}_j \boldsymbol{w}_i^T$ and defining $\boldsymbol{G}^{(i)} = \sum_{j \in \Omega_i} \boldsymbol{h}_j^T \boldsymbol{h}_j$,

$$(\boldsymbol{G}^{(i)} + \lambda \boldsymbol{I}) \boldsymbol{w}_i^T = \sum_{j \in \Omega_i} \boldsymbol{h}_j^T a_{ij}$$

which is a $k \times k$ symmetric linear system of equations

ALS: Iteration Cost

For updating each $\boldsymbol{w}_i$, ALS is dominated in cost by two steps

1. $\boldsymbol{G}^{(i)} = \sum_{j \in \Omega_i} \boldsymbol{h}_j^T \boldsymbol{h}_j$
   - dense matrix-matrix product
   - $O(|\Omega_i| k^2)$ work
   - logarithmic depth
2. Solve linear system with $\boldsymbol{G}^{(i)} + \lambda \boldsymbol{I}$
   - dense symmetric $k \times k$ linear solve
   - $O(k^3)$ work
   - typically $O(k)$ depth

Can do these for all m rows of $\boldsymbol{W}$ independently

Parallel ALS

Let each task optimize a row $\boldsymbol{w}_i$ of $\boldsymbol{W}$

- Need to compute $\boldsymbol{G}^{(i)}$ for each task
- Specific subset of rows of $\boldsymbol{H}$ needed for each $\boldsymbol{G}^{(i)}$
- Task execution is embarassingly parallel if all of $\boldsymbol{H}$ stored on each processor

Memory-Constrained Parallel ALS

May not have enough memory to replicate $\boldsymbol{H}$ on all processors

- Communication required and pattern is data-dependent
- Could rotate rows of $\boldsymbol{H}$ along a ring of processors
- Each processor computes contributions to the $\boldsymbol{G}^{(i)}$ it owns
- Requires $\Theta(p)$ latency cost for each iteration of ALS

Updating a Single Variable

Rather than whole rows $\boldsymbol{w}_i$ solve for elements of $\boldsymbol{W}$, recall

$$\underset{\boldsymbol{W}\in\mathbb{R}^{m\times k}}{\operatorname{argmin}} \sum_{(i,j)\in\Omega} \left(a_{ij} - \sum_{l} w_{il}h_{jl}\right)^2 + \lambda||\boldsymbol{W}||_F^2$$

- *Coordinate descent* finds the best replacement μ for w_{it}

$$\mu = \underset{\mu}{\operatorname{argmin}} \sum_{j\in\Omega_i} \left(a_{ij} - \mu h_{jt} - \sum_{l\neq t} w_{il}h_{jl}\right)^2 + \lambda\mu^2$$

- The solution is given by

$$\mu = \frac{\sum_{j\in\Omega_i} h_{jt}\left(a_{ij} - \sum_{l\neq t} w_{il}h_{jl}\right)}{\lambda + \sum_{j\in\Omega_i} h_{jt}^2}$$

Coordinate Descent

For $\forall(i,j) \in \Omega$ compute elements r_{ij} of

$$\boldsymbol{R} = \boldsymbol{A} - \boldsymbol{W}\boldsymbol{H}^T$$

so that we can optimize via

$$\mu = \frac{\sum_{j\in\Omega_i} h_{jt}\left(a_{ij} - \sum_{l\neq t} w_{il}h_{jl}\right)}{\lambda + \sum_{j\in\Omega_i} h_{jt}^2} = \frac{\sum_{j\in\Omega_i} h_{jt}\left(r_{ij} + w_{it}h_{jt}\right)}{\lambda + \sum_{j\in\Omega_i} h_{jt}^2}$$

after which we can update $\boldsymbol{R}$ via

$$r_{ij} \leftarrow r_{ij} - (\mu - w_{it})h_{jt} \quad \forall j \in \Omega_i$$

both using $O(|\Omega_i|)$ operations

Cyclic Coordinate Descent (CCD)

- Updating $\boldsymbol{w}_i$ costs $O(|\Omega_i|k)$ operations with coordinate descent rather than $O(|\Omega_i|k^2 + k^3)$ operations with ALS
- By solving for all of $\boldsymbol{w}_i$ at once, ALS obtains a more accurate solution than coordinate descent
- Coordinate descent with different update orderings:
  - *Cyclic coordinate descent (CCD)* updates all columns of $\boldsymbol{W}$ then all columns of $\boldsymbol{H}$ (ALS-like ordering)
  - *CCD++* alternates between columns of $\boldsymbol{W}$ and $\boldsymbol{H}$
  - All entries within a column can be updated concurrently

Parallel CCD++

Yu, Hsieh, Si, and Dhillon 2013 propose using a row-blocked layout of $\boldsymbol{H}$ and $\boldsymbol{W}$

- They keep track of a corresponding m/p and n/p rows and columns of $\boldsymbol{A}$ and $\boldsymbol{R}$ on each processor (using twice the minimal amount of memory)
- Every column update in CCD++ is then fully parallelized, but an allgather of each column is required to update $\boldsymbol{R}$
- The complexity of updating all of $\boldsymbol{W}$ and all of $\boldsymbol{H}$ is then

$$\begin{aligned} T_p(m,n,k) &= \Theta(kT_p^{\text{allgather}}(m+n) + \gamma Q_1(m,n,k)/p) \\ &= O(\alpha k \log p + \beta(m+n)k + \gamma|\Omega|k/p) \end{aligned}$$

Gradient-Based Update

ALS minimizes $\boldsymbol{w}_i$, gradient descent methods only improve it

- Recall that we seek to minimize

$$f(\boldsymbol{w}_i) = \sum_{j \in \Omega_i} \left(a_{ij} - \boldsymbol{w}_i \boldsymbol{h}_j^T \right)^2 + \lambda ||\boldsymbol{w}_i||^2$$

and use the partial derivative

$$\frac{\partial f(\boldsymbol{w}_i)}{\partial \boldsymbol{w}_i} = 2 \sum_{j \in \Omega_i} \boldsymbol{h}_j^T \left(\boldsymbol{w}_i \boldsymbol{h}_j^T - a_{ij} \right) + 2\lambda \boldsymbol{w}_i = 2 \left(\lambda \boldsymbol{w}_i - \sum_{j \in \Omega_i} r_{ij} \boldsymbol{h}_j \right)$$

- *Gradient descent* method updates

$$\boldsymbol{w}_i = \boldsymbol{w}_i - \eta \frac{\partial f(\boldsymbol{w}_i)}{\partial \boldsymbol{w}_i}$$

where parameter η is our step-size

Stochastic Gradient Descent (SGD)

Stochastic gradient descent (SGD) performs fine-grained updates based on a component of the gradient

- Again the full gradient is

$$\frac{\partial f(\boldsymbol{w}_i)}{\partial \boldsymbol{w}_i} = 2\left(\lambda \boldsymbol{w}_i - \sum_{j \in \Omega_i} r_{ij} \boldsymbol{h}_j\right) = 2 \sum_{j \in \Omega_i} \lambda \boldsymbol{w}_i / |\Omega_i| - r_{ij} \boldsymbol{h}_j$$

- SGD selects random $(i, j) \in \Omega$ and updates $\boldsymbol{w}_i$ using $\boldsymbol{h}_j$

$$\boldsymbol{w}_i \leftarrow \boldsymbol{w}_i - \eta(\lambda \boldsymbol{w}_i / |\Omega_i| - r_{ij} \boldsymbol{h}_j)$$

- SGD then updates $r_{ij} = a_{ij} - \boldsymbol{w}_i^T \boldsymbol{h}_j$
- Each update costs $O(k)$ operations

Asynchronous SGD

Parallelizing SGD is easy aside from ensuring concurrent updates do not conflict

- Asynchronous shared-memory implementations of SGD are popular and achieve high performance
- For sufficiently small step-size, inconsistencies among updates (e.g. duplication) are not problematic statistically
- Asynchronicity can slow down convergence

Blocked SGD

Distributed blocking SGD introduces further considerations

- Associate a task with updates on a block
- Can define $p \times p$ grid of blocks of dimension $m/p \times n/p$
- Diagonal/superdiagonals/subdiagonals of blocks updated independently, so p tasks can execute concurrently
- Assuming $\Theta(|\Omega|/p^2)$ updates are performed on each block, the execution time for $|\Omega|$ updates is

$$T_p(m, n, k) = \Theta(\alpha p \log p + \beta \min(m, n)k + \gamma|\Omega|k/p)$$

References

- Candés, Emmanuel J., and Benjamin Recht. "Exact matrix completion via convex optimization." *Foundations of computational mathematics* 9.6 (2009): 717.
- Jain, Prateek, Praneeth Netrapalli, and Sujay Sanghavi. "Low-rank matrix completion using alternating minimization." *Proceedings of the forty-fifth annual ACM Symposium on Theory of Computing*. ACM, 2013.
- Yu, H.F., Hsieh, C.J., Si, S. and Dhillon, I., 2012, December. Scalable coordinate descent approaches to parallel matrix factorization for recommender systems. In 2012 *IEEE 12th International Conference on Data Mining* (pp. 765-774).

References

- Recht, Benjamin, Christopher Re, Stephen Wright, and Feng Niu. "Hogwild: A lock-free approach to parallelizing stochastic gradient descent." In *Advances in neural information processing systems*, pp. 693-701. 2011.
- Gemulla, Rainer, Erik Nijkamp, Peter J. Haas, and Yannis Sismanis. "Large-scale matrix factorization with distributed stochastic gradient descent." In Proceedings of the *17th ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 69-77. ACM, 2011.
- Karlsson, Lars, Daniel Kressner, and André Uschmajew. "Parallel algorithms for tensor completion in the CP format." *Parallel computing* 57 (2016): 222-234.

References – Parallel Optimization

- J. E. Dennis and V. Torczon, Direct search methods on parallel machines, *SIAM J. Optimization* 1:448-474, 1991
- J. E. Dennis and Z. Wu, Parallel continuous optimization, J. Dongarra et al., eds., *Sourcebook of Parallel Computing*, pp. 649-670, Morgan Kauffman, 2003
- F. A. Lootsma and K. M. Ragsdell, State-of-the-art in parallel nonlinear optimization, *Parallel Computing* 6:133-155, 1988
- R. Schnabel, Sequential and parallel methods for unconstrained optimization, M. Iri and K. Tanabe, eds., *Mathematical Programming: Recent Developments and Applications*, pp. 227-261, Kluwer, 1989
- S. A. Zenios, Parallel numerical optimization: current trends and an annotated bibliography, *ORSA J. Comput.* 1:20-43, 1989