Investigating Quasi-Newton Compact Dense Representations on GPUs

Alp Dener    Todd Munson

Mathematics and Computer Science Division
Argonne National Laboratory

LANS Seminar
February 5, 2020
Outline

Introduction and Background

Limited-Memory BFGS

Compact Dense Representation

Numerical Experiments

Observations
Outline

Introduction and Background

Limited-Memory BFGS

Compact Dense Representation

Numerical Experiments

Observations
The Good Old Days

Simulation-based Applications
- (Relatively) Small optimization problem
- (Relatively) Large simulation
- Computational cost dominated by governing equations (i.e. objective function and/or gradient evaluation)

CPU-based Architectures
- Homogeneous computing w/MPI
- Full instruction set
- Mature software stack
The Accelerator Takeover

- **Top500 Rank: 2** (125 PFlop/s)
  - CPU: IBM Power9 (2/node)
  - GPU: NVIDIA Volta V100 (4/node)

- **Top500 Rank: 1** (200 PFlop/s)
  - CPU: IBM Power9 (2/node)
  - GPU: NVIDIA Volta V100 (6/node)

- 1000+ Pflop/s
  - CPU: Intel Xeon (2/node)
  - GPU: Xe-arch based GP-GPU (6/node)

Over 95% of flops from GPUs
The Accelerator Takeover

- **Top500 Rank: 2**  
  (125 PFlop/s)  
  CPU: IBM Power9  
  (2/node)  
  GPU: NVIDIA Volta V100 (4/node)

- **Top500 Rank: 1**  
  (200 PFlop/s)  
  CPU: IBM Power9  
  (2/node)  
  GPU: NVIDIA Volta V100 (6/node)

- 1000+ Pflop/s  
  CPU: Intel Xeon  
  (2/node)  
  GPU: Xe-arch based GP-GPU (6/node)

Over 95% of flops from GPUs
Why does it matter?

Libraries and application codes are being ported to new architectures

- Necessity - CPUs amount to a small % compute power on the latest generation supercomputers
- SIAM PP20 Minisymposiums - FASTMath (MS3 & MS12) and PETSc (MS23 & MS34)

Emerging applications in data science, machine learning and artificial intelligence

- Perform a lot of tasks well suited to GPUs and/or heterogeneous systems
- Can generate (very) large optimization problems

Optimization (and other "outer loop" tools) must also run on GPUs
PETSc/TAO Overview

**PETSc** – Portable Extensible Toolkit for Scientific Computing

**TAO** – Toolkit for Advanced Optimization

▶ Parallelized with PETSc Vec and Mat data structures
▶ Provides gradient-based solvers for large-scale optimization
▶ Unconstrained and bound-constrained methods:
  ▶ Nonlinear Conjugate Gradient (BNCG)
  ▶ Quasi-Newton (BQNLS)
  ▶ Truncated Newton (BNLS, BNTR)
▶ Constrained methods:
  ▶ Alternating Directions Method of Multipliers (ADMM)
  ▶ More to come in 2020
PETSc/TAO Overview

**PETSc** – Portable Extensible Toolkit for Scientific Computing

**TAO** – Toolkit for Advanced Optimization

- Parallelized with PETSc Vec and Mat data structures
- Provides gradient-based solvers for large-scale optimization
- Unconstrained and bound-constrained methods:
  - Nonlinear Conjugate Gradient (BNCG)
  - Quasi-Newton (BQNLS)
  - Truncated Newton (BNLS, BNTR)

- Constrained methods:
  - Alternating Directions Method of Multipliers (ADMM)
  - More to come in 2020

**Today:** Investigating QN on GPUs
PETSc/TAO Overview

**PETSc** – Portable Extensible Toolkit for Scientific Computing

**TAO** – Toolkit for Advanced Optimization
- Parallelized with PETSc Vec and Mat data structures
- Provides gradient-based solvers for large-scale optimization
- Unconstrained and bound-constrained methods:
  - Nonlinear Conjugate Gradient (BNCG)
  - Quasi-Newton (BQNLS)
  - Truncated Newton (BNLS, BNTR)
- Constrained methods:
  - Alternating Directions Method of Multipliers (ADMM)
  - More to come in 2020

**Tomorrow:** Profiling ADMM on GPUs, Todd Munson (MS23)
Outline

Introduction and Background

Limited-Memory BFGS

Compact Dense Representation

Numerical Experiments

Observations
The Basics

\[ \min_x f(x) \]

**For** \(k=0,1,2,\ldots\)

\[ p_k = \arg \min_p \frac{1}{2} p_k^T \nabla^2_{xx} f(x_k) p_k + p_k^T \nabla_x f(x_k) \]

\[ x_{k+1} = x_k + \alpha p_k \]
The Basics

\[
\min_x f(x)
\]

\[
p_k = -\left[\nabla^2_{xx} f(x_{k+1})\right]^{-1} \nabla_x f(x_k)
\]

\[
x_{k+1} = x_k + \alpha p_k
\]

BFGS approximates the Hessian as

\[
\left[\nabla^2_{xx} f(x_{k+1})\right]^{-1} \approx H_{k+1} = \left( I - \frac{s_k y_k^T}{y_k^T s_k} \right) H_k \left( I - \frac{y_k s_k^T}{y_k^T s_k} \right) + \frac{s_k s_k^T}{y_k^T s_k}
\]

with \( s_k = x_k - x_{k-1} \) and \( y_k = g_k - g_{k-1} \) where \( g_k = \nabla_x f(x_k) \)
Why quasi-Newton?

\[
\left[ \nabla^2_{xx} f(x_{k+1}) \right]^{-1} \approx H_{k+1} = \left( I - \frac{s_k y_k^T}{y_k^T s_k} \right) H_k \left( I - \frac{y_k s_k^T}{y_k^T s_k} \right) + \frac{s_k s_k^T}{y_k^T s_k}
\]

- First-order method – computing Hessians is prohibitively expensive for many applications
- L-BFGS is one of the most popular general-purpose gradient-based optimization algorithms
- Recent focus on developing stochastic variants for emerging ML/AI applications
A Practical Implementation

- Limited-memory implementation stores only $m \ll 100$ iterations of $(s, y)$ pairs
- Matrix-free two-loop algorithm computes the action of the inverse Hessian on a vector
- Available as MATLMVMBFGS in PETSc/TAO

$$q \leftarrow g_k$$

**for** $i = k - 1, k - 2, \ldots, k - m$ **do**

- $\alpha_i \leftarrow \frac{s_i^T q}{y_i^T s_i}$
- $q \leftarrow q - \alpha_i y_i$

**end for**

$z \leftarrow H_0 q$

**for** $i=k-m,k-m+1,\ldots,k-1$ **do**

- $\beta_i \leftarrow \frac{y_i^T z}{y_i^T s_i}$
- $z \leftarrow z + (\alpha_i - \beta_i) s_i$

**end for**
A Practical Implementation

- Limited-memory implementation stores only $m \ll 100$ iterations of $(s, y)$ pairs
- Matrix-free two-loop algorithm computes the action of the inverse Hessian on a vector
- Available as MATLMVMBFGS in PETSc/TAO

$$q \leftarrow g_k$$

**for** $i = k-1, k-2, \ldots, k-m$ **do**

$$\alpha_i \leftarrow \frac{s_i^T q}{y_i^T s_i}$$

$$q \leftarrow q - \alpha_i y_i$$

**end for**

$$z \leftarrow H_0 q$$

**for** $i=k-m, k-m+1, \ldots, k-1$ **do**

$$\beta_i \leftarrow \frac{y_i^T z}{y_i^T s_i}$$

$$z \leftarrow z + (\alpha_i - \beta_i)s_i$$

**end for**

Collective vector operations do not leverage GPU capabilities
A Practical Implementation

- Limited-memory implementation stores only \( m \ll 100 \) iterations of \((s, y)\) pairs
- Matrix-free two-loop algorithm computes the action of the inverse Hessian on a vector
- Available as MATLMVMBFGS in PETSc/TAO

\[
q \leftarrow g_k \\
\text{for } i = k - 1, k - 2, \ldots, k - m \text{ do} \\
\quad \alpha_i \leftarrow \frac{s_i^T q}{y_i^T s_i} \\
\quad q \leftarrow q - \alpha_i y_i \\
\text{end for} \\
\]

\[
z \leftarrow H_0 q \\
\text{for } i=k-m,k-m+1,\ldots,k-1 \text{ do} \\
\quad \beta_i \leftarrow \frac{y_i^T z}{y_i^T s_i} \\
\quad z \leftarrow z + (\alpha_i - \beta_i) s_i \\
\text{end for}
\]

Collective vector operations do not leverage GPU capabilities

Can we trade-off storage for better performance?
Does it run on GPUs?

**Disclaimer:**
Preliminary investigation, no general conclusions!

- Intel Core i5-9400F (262 GF)
- NVIDIA GTX 1080 (277 GF)
- CUDA 10.2
- $m = 5$
- VECSEQ vs. VECSEQCUDA

Computing $H_k z$ with two-loop algorithm
Outline

Introduction and Background

Limited-Memory BFGS

Compact Dense Representation

Numerical Experiments

Observations
An Alternative View

BFGS can be reformulated as

\[ H_{k+1} = H_0 + [H_0 Y_k \quad S_k] \begin{bmatrix} 0 & -\tilde{R}_k^{-1} \\ -\tilde{R}_k^{-T} & \tilde{R}_k^{-T} (D_k + Y_k^T H_0 Y_k) \tilde{R}_k^{-1} \end{bmatrix} \begin{bmatrix} Y_k^T H_0 \\ S_k^T \end{bmatrix} \]

where \( S_k = [s_1 \quad s_2 \quad \ldots \quad s_k] \), \( Y_k = [y_1 \quad y_2 \quad \ldots \quad y_k] \),

\( S_k^T Y_k = L_k + D_k + R_k \) and \( \tilde{R}_k = D_k + R_k \)
An Alternative View

BFGS can be reformulated as

\[
H_{k+1} = H_0 + \begin{bmatrix} H_0 Y_k & S_k \end{bmatrix} \begin{bmatrix} 0 & -\bar{R}_k^{-1} \\ -\bar{R}_k^{-T} & \bar{R}_k^{-T} \left( D_k + Y_k^T H_0 Y_k \right) \bar{R}_k^{-1} \end{bmatrix} \begin{bmatrix} Y_k^T H_0 \\ S_k^T \end{bmatrix}
\]

where

\[ S_k = \begin{bmatrix} s_1 & s_2 & \ldots & s_k \end{bmatrix}, \quad Y_k = \begin{bmatrix} y_1 & y_2 & \ldots & y_k \end{bmatrix}, \]

\[ S_k^T Y_k = L_k + D_k + R_k \text{ and } \bar{R}_k = D_k + R_k \]

\[
S_k^T Y_k, L_k, D_k, R_k, \bar{R}_k \in \mathbb{R}^{(m \times m)}
\]

\[ S_k, Y_k \in \mathbb{R}^{(n \times m)} \text{ where } x \in \mathbb{R}^n \]
Implementation Notes

\[
\begin{bmatrix}
0 & -\bar{R}_k^{-1} \\
-\bar{R}_k^{-T} & \bar{R}_k^{-T} (D_k + Y_k^T H_0 Y_k) \bar{R}_k^{-1}
\end{bmatrix}
\] can be assembled efficiently (see Alg. 1 from Erway and Marcia, 2016)

- Leverage fast mat-vec on GPUs – products with \( S_k \) and \( Y_k \) instead of looping over sequence of update vectors

- Theoretically requires only \((4m^3 + 4m^2)\) more storage but a practical implementation can approach a \(2\times\) factor

- Approx. 40% savings on flop count compared to two-loop algorithm for \( n \gg 100 \)

<table>
<thead>
<tr>
<th>Method</th>
<th>Flop Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Two-loop</td>
<td>(4nm + 3m + n + 2m(2n - 1))</td>
</tr>
<tr>
<td></td>
<td>((2m + 1)(n + m + 1) + 2n)</td>
</tr>
<tr>
<td></td>
<td>(+ 13(40m^3 + 90m^2 + 122m))</td>
</tr>
<tr>
<td></td>
<td>(+ (2n - 1)(m + 1))</td>
</tr>
<tr>
<td>Compact dense</td>
<td>((2m + 1)(n + m + 1) + 2n)</td>
</tr>
<tr>
<td></td>
<td>(+ 13(40m^3 + 90m^2 + 122m))</td>
</tr>
<tr>
<td></td>
<td>(+ (2n - 1)(m + 1))</td>
</tr>
</tbody>
</table>
Outline

Introduction and Background

Limited-Memory BFGS

Compact Dense Representation

Numerical Experiments

Observations
Compact Dense L-BFGS

Computing $H_k z$ with the compact dense representation
Head-to-Head (CPU)

Compact dense vs. two-loop $H_{kz}$ calculation on the CPU
Head-to-Head (GPU)

Compact dense vs. two-loop $H_{kz}$ calculation on the GPU
What’s the catch?

*H_k z* calculation does not include the cost of updating *S_k* and *Y_k* matrices with new iterate information

**Matrix-free Two-loop BFGS:**
- Store *S_k* and *Y_k* as array of vectors
- Vectors indexes for *i ≥ k* never used
- Reassign pointers to shift vectors when *k = m*

**Compact Dense BFGS:**
- Resize *S_k* and *Y_k* matrices when *k < m*
- Shift matrix columns when *k = m*
- Avoiding resizing/shifting requires custom mat-vec kernel
$S_k$ and $Y_k$ updates are not trivial!

Compact dense vs. two-loop updates on the CPU
$S_k$ and $Y_k$ updates are NOT trivial!

Compact dense vs. two-loop updates on the GPU
Compact dense BFGS might take better advantage of GPUs than the matrix-free two-loop algorithm when computing $H_k z$

**Future Work:**
- Changing size of $S_k$ and $Y_k$ pose some implementation challenges – need to write dedicated kernel instead of using high-level interfaces
- Matrix algebra needs to be inspected carefully to avoid unnecessary CPU-GPU copy operations
- Lack of MATMPIDENSECUDA in PETSc means dense $S_k$ and $Y_k$ has to use MATMPIAIJCUSPARSE and incur overhead cost
- More profiling in HPC environments (ORNL Summit)
References:


MF-BFGS Updates

Updating matrix-free BFGS with new $s_k$ and $y_k$ vectors
CD-BFGS Updates

Updating the $S_k$ and $Y_k$ matrices for compact dense BFGS