HyKKT: A Hybrid Direct and Iterative Method for Solving KKT Linear Systems

Shaked Regev, Nai-Yuan Chiang, Eric Darve, Cosmin Petra, Michael A. Saunders, Kasia Świrydowicz, Slaven Peleš

Stanford University; Pacific Northwest National Laboratory; Lawrence Livermore National Laboratory

sregev@stanford.edu

August 3, 2022
Approved for unlimited distribution PNNL-SA-164807
Acknowledgements

This research was supported by the Exascale Computing Project (ECP), Project Number: 17-SC-20-SC, a collaborative effort of two DOE organizations—the Office of Science and the National Nuclear Security Administration—responsible for the planning and preparation of a capable exascale ecosystem—including software, applications, hardware, advanced system engineering, and early testbed platforms—to support the nation’s exascale computing imperative.
Sparse NLP formulation supports sparse optimization problems, requires Hessians of objective and constraints in addition to gradients

\[
\begin{align*}
\min_{x \in \mathbb{R}^n} & \quad f(x) \\
\text{s.t.} & \quad c(x) = c_E \\
& \quad [y] \\
& \quad [y_{d,l}] \quad d_l \leq d(x) \leq d_u \quad [y_{d,u}] \\
& \quad [z_l] \quad x_l \leq x \leq x_u \quad [z_u]
\end{align*}
\]

- Assume gradients and sparse Hessians are available
- Quantities inside brackets are Lagrange multipliers for the constraints
- For infinite bounds, multiplier is 0
Model Requirements

D1  objective and constraint functions \( f(x), c(x), d(x) \)

D2  first derivatives: \( \nabla f(x), J_c(x) = \nabla c(x), J_d(x) = \nabla d(x) \)

D3  Hessian of the Lagrangian

\[
\nabla^2 L(x) = \nabla^2 f(x) + \sum_i y_{c,i} \nabla^2 c_i(x) + \sum_i y_{d,i} \nabla^2 d_i(x)
\]

D4  simple bounds \( x_l \) and \( x_u \), inequality bounds \( d_l \) and \( d_u \),
and rhs \( c_E \) of equality constraints
Motivation

- Out of the box GPU solvers do not work well on these problems
- KLU + cuSolver works but is proprietary, only works on NVIDIA GPUs
- Want a solver that allows substantial speedup on GPUs
- Using a Cholesky solver instead of $LBL^T$ would allow parallelization and GPU utilization, but the problem is indefinite

<table>
<thead>
<tr>
<th>Test case</th>
<th>Size</th>
<th>NNZ</th>
<th>MA57 reference CPU only</th>
<th>SuperLU (ECP – LBNL)</th>
<th>STRUMPACK (ECP – LBNL)</th>
<th>KLU + cuSolver (NVIDIA)</th>
<th>SSIDS (STFC, UK Gov.)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CPU</td>
<td>GPU</td>
<td>CPU</td>
<td>GPU</td>
</tr>
<tr>
<td>10k-bus</td>
<td>238,072</td>
<td>1,111,991</td>
<td>0.54s</td>
<td>4.06s</td>
<td>4.95s</td>
<td>2.82s</td>
<td>3.71s</td>
</tr>
<tr>
<td>70k-bus</td>
<td>1,640,411</td>
<td>7,671,693</td>
<td>5.30s</td>
<td>30.46s</td>
<td>35.58s</td>
<td>24.4s</td>
<td>26.8s</td>
</tr>
</tbody>
</table>

* Total computational cost of symbolic factorization amortized over 5 solves, i.e. assumed 1 symbolic factorization can be reused over 5 optimization solver iterations.

K. Świrydowicz et al. (2021) Linear solvers for power grid optimization problems: a review of GPU-accelerated linear solvers
Problem Setup

Interior method, used to solve KKT systems, generates series of linear systems $K_k \Delta x_k = r_k$:

$$
\begin{bmatrix}
H + D_x & 0 & J_c^T & J_d^T \\
0 & D_s & 0 & -I \\
J_c & 0 & 0 & 0 \\
J_d & -I & 0 & 0
\end{bmatrix}
\begin{bmatrix}
\Delta x \\
\Delta s \\
\Delta y \\
\Delta y_d
\end{bmatrix}
= 
\begin{bmatrix}
\tilde{r}_x \\
r_s \\
r_{yc} \\
r_{yd}
\end{bmatrix}
$$

- $J_c$ and $J_d$ - sparse Jacobians for equality and inequality constraints
- $H$ - sparse Hessian matrix
- Diagonal $D_x$ arises from primal variables $x$ in log-barrier function
- Diagonal $D_s$ arises from slack variables $s$ in log-barrier function
Simplifying the Problem

Eliminating $\Delta s = J_d \Delta x - r_{yd}$ and $\Delta y_d = D_s \Delta s - r_d$ gives

$$
\begin{bmatrix}
\tilde{H} & J_c^T \\
J_c & 0
\end{bmatrix}
\begin{bmatrix}
\Delta x \\
\Delta y
\end{bmatrix} =
\begin{bmatrix}
r_x \\
r_{yc}
\end{bmatrix}, \quad \tilde{H} \equiv H + D_x + J_d^T D_s J_d,
$$

where $r_x = \tilde{r}_x + J_d^T (D_s r_{yd} + r_d)$. Gaussian elimination with pivot $\begin{bmatrix} D_s & -I \\ -I & -I \end{bmatrix}$.

- We need to perform Ruiz Scaling on $\tilde{H}$ and $J_c$ so we can judge the sizes of the entries in the blocks

C. Petra et al. (2009) A computational study of the use of an optimization-based method for simulating large multibody systems
D. Ruiz (2001) A scaling algorithm to equilibrate both rows and columns norms in matrices
The KKT system is equivalent to

\[
\begin{bmatrix}
H_\gamma & J_c^T \\
J_c & 0
\end{bmatrix}
\begin{bmatrix}
\Delta x \\
\Delta y
\end{bmatrix} =
\begin{bmatrix}
\hat{r}_x \\
r_{yc}
\end{bmatrix},
\quad H_\gamma = \tilde{H} + \gamma J_c^T J_c,
\quad \hat{r}_x = r_x + \gamma J_c^T r_{yc}
\]

- \( \gamma > 0 \) makes the system more SPD (increases the eigenvalues)
- If \( H_\gamma \) or whole system are poorly conditioned, only option may be to ignore the block structure and use an LBL^T factorization
- Sparse Cholesky on \( H_\gamma \) and CG on its Schur complement \( S \):

\[
S \Delta y = J_c H_\gamma^{-1} \hat{r}_x - r_{yc},
\quad S = J_c H_\gamma^{-1} J_c^T
\]

\[
H_\gamma \Delta x = \hat{r}_x - J_c^T \Delta y
\]

Reminder: \( \tilde{H} \equiv H + D_x + J_d^T D_s J_d, \quad H_\gamma = \tilde{H} + \gamma J_c^T J_c \)

- For large \( \gamma \) and full-rank \( J_c \), \( \tilde{H} \) is PD on null(\( J_c \)) iff \( H_\gamma \) is uniformly PD (required at optimization problem solution).
- HyKKT provides descent direction to interior method (for large \( \gamma \))
- If \( \tilde{H} \) is positive definite on null(\( J_c^T J_c \)), \( \exists \gamma_{\text{max}} \) such that for \( \gamma \geq \gamma_{\text{max}} \), \( \kappa(H_\gamma) \) increases linearly with \( \gamma \).
- For large enough \( \gamma \) and \( C \equiv 1/\gamma \left( J_c \tilde{H}^{-1} J_c^T \right)^{-1} \), \( S_\gamma \equiv \gamma S = \gamma J_c H_\gamma^{-1} J_c^T = \sum_{k=0}^{\infty} (-1)^k C^k = I - C + O \left( \frac{1}{\gamma^2} \right) = I + O \left( \frac{1}{\gamma} \right) \).
HyKKT Workflow

1. Transform 4X4 system to 2X2
2. Perform Ruiz scaling to normalize entry magnitudes in $\tilde{H}$, $j_c$, $j_E$
3. Compute augmented 2X2 system with $H_y$, $\hat{r}_x$
4. First optimization solver iteration?
   - Yes: Compute symamd ordering of $H_y$ to minimize fill-in
   - No: Permuate the augmented 2X2 system
5. Perform symbolic analysis on $H_y$
6. First optimization solver iteration?
   - Yes:
     - Regularize $H_y$ to $H_\delta$
     - Is $\delta$ too large?
       - No: Exit and call LBL solver
       - Yes: Solve schur complement system with $S_\delta = J_y H_\delta^{-1} J_E$
         via conjugate gradient using Chol($H_y$)
   - No: Regularize $S_\delta$
   - Small dot products encountered?
     - Yes: Recover solution to original system via matrix vector products, solves with Chol($H_y$), reverse permutation, and reverse scaling
     - No:
Preliminary Results With Solver Prototype

- **RR** for $Ax = b$: $\frac{\|Ax-b\|}{\|b\|}$

- **BE**: $\frac{\|Ax-b\|}{\|A\|\|x\|+\|b\|}$, with $\|A\|_\infty \approx \|A\|$

- 5/6 matrix series (NLPs at different iterations of interior method) solved efficiently and accurately (other needed $O(1)$ regularization)

- BE consistently $< 10^{-8}$

- Average CG iterations < 20.

- $\delta_{\text{min}}$ in the range $10^{-8}$ down to $10^{-10}$ is reasonable for any $\gamma \leq 10^8$

- (1) refers to the $4 \times 4$ system, (2) refers to the $2 \times 2$ system

Figure: (Left) CG iterations on $S$ with $\gamma = 10^6$. Mean number of iterations is 13.1. (Right) Various errors for $\gamma = 10^6$. BE $< 10^{-14}$. 
Comparison with $\text{LBL}^\top$: Factorization Density

**Table:** Dimensions, number of nonzeros, and factorization densities (average number of nonzeros in the factors per row) for solving full system directly with $\text{LBL}^\top$ via MA57 with pivot tolerance 0.01 ($n_L$, nnz$_L$, $\rho_L$) and solving systems with $H_\gamma$ with Cholesky ($n_C$, nnz$_C$, $\rho_C$)

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>$n_L$</th>
<th>nnz$_L$</th>
<th>$\rho_L$</th>
<th>$n_C$</th>
<th>nnz$_C$</th>
<th>$\rho_C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Illinois</td>
<td>4.64K</td>
<td>94.7K</td>
<td><strong>20.4</strong></td>
<td>2.28K</td>
<td>34.9K</td>
<td><strong>15.3</strong></td>
</tr>
<tr>
<td>Texas</td>
<td>55.7K</td>
<td>2.95M</td>
<td><strong>52.9</strong></td>
<td>25.9K</td>
<td>645K</td>
<td><strong>24.9</strong></td>
</tr>
<tr>
<td>Western US</td>
<td>238K</td>
<td>10.7M</td>
<td><strong>44.8</strong></td>
<td>116K</td>
<td>2.23M</td>
<td><strong>19.2</strong></td>
</tr>
<tr>
<td>Eastern US</td>
<td>1.64M</td>
<td>85.4M</td>
<td><strong>52.1</strong></td>
<td>794K</td>
<td>17.7M</td>
<td><strong>22.3</strong></td>
</tr>
</tbody>
</table>
Comparison with LBL$^T$: Run Time (Preliminary)

**Table:** Times (in seconds) for solving full system directly on a CPU with LBL$^T$ (via MA57) or HyKKT on a GPU. CG is solved to tolerance of $10^{-12}$. All runs are on x86_64 CPUs and A100 GPUs. As the problems grow larger, HyKKT outperforms MA57 by an increasing factor.

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>MA57</th>
<th>HyKKT</th>
<th>Relative size</th>
<th>MA57/HyKKT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Illinois</td>
<td>$6.24 \cdot 10^{-3}$</td>
<td>$1.01 \cdot 10^{-2}$</td>
<td>1</td>
<td>0.62</td>
</tr>
<tr>
<td>Texas</td>
<td>$1.00 \cdot 10^{-1}$</td>
<td>$1.04 \cdot 10^{-1}$</td>
<td>10</td>
<td>1.04</td>
</tr>
<tr>
<td>Western US</td>
<td>$3.38 \cdot 10^{-1}$</td>
<td>$1.46 \cdot 10^{-1}$</td>
<td>50</td>
<td>2.32</td>
</tr>
<tr>
<td>Eastern US</td>
<td>$3.48 \cdot 10^0$</td>
<td>$3.31 \cdot 10^{-1}$</td>
<td>350</td>
<td>10.5</td>
</tr>
</tbody>
</table>

Duff (2004)
Chen, Davis, Hager, and Rajamanickam (2008)
Summary

- We designed a linear solver strategy suitable for fine-grain parallelization and deployment on hardware accelerators.
- We prove fast CG, holds in practice.
- The iterative nature of the solver provides more flexibility to balance trade-offs between accuracy and performance.
- Cholesky instead of $LBL^T$ allows for better GPU utilization.
- Non-optimized HyKKT outperforms MA57 by $10^x$ on largest problems tested.