Using GPUs to Accelerate the Solution of Large-Scale Model Reduction Problems

Peter Benner\textsuperscript{1} Pablo Ezzatti\textsuperscript{2} Francisco Igual\textsuperscript{3} Daniel Kressner\textsuperscript{4} Enrique S. Quintana-Ortí\textsuperscript{3} Alfredo Remón\textsuperscript{3}

\textsuperscript{1}Fakultät für Mathematik, Chemnitz University of Technology, Chemnitz (Germany)
\textsuperscript{2}Centro de Cálculo–Instituto de la Computación, Universidad de la República, Montevideo (Uruguay)
\textsuperscript{3}Seminar für angewandte Mathematik, ETH Zürich, Zurich (Switzerland).
\textsuperscript{4}Depto. de Ingeniería y Ciencia de Computadores, Universidad Jaume I, Castellón (Spain).
Linear time-invariant systems:

\[ \dot{x}(t) = Ax(t) + Bu(t), \quad t > 0, \quad x(0) = x^0, \]
\[ y(t) = Cx(t) + Du(t), \quad t \geq 0, \]

- \( n \) state-space variables, i.e., \( n \) is the order of the system;
- \( m \) inputs,
- \( p \) outputs.

Corresponding TFM:

\[ G(s) = C(sI_n - A)^{-1}B + D. \]
Find a reduced-order model

\[
\dot{x}(t) = \hat{A} \hat{x}(t) + \hat{B} u(t), \quad t > 0, \quad \hat{x}(0) = \hat{x}^0,
\]

\[
\hat{y}(t) = \hat{C} \hat{x}(t) + \hat{D} u(t), \quad t \geq 0,
\]

of order \( r \ll n \) such that the output error

\[
y - \hat{y} = G u - \hat{G} u = (G - \hat{G}) u
\]

is “small”.

**Example**

**μ-mechanical Gyroscope**  
[The IMEGO Institute (Sweden) + Saab Bofors Dynamics AB]

- Commercial rate sensor with applications in inertial navigation systems.
- Simulation problem: Improve the design with respect to a number of parameters.
- \( n = 17,361 \) states.

Can we obtain a reduced-order model with similar behavior?
1. Truncation methods for model reduction
2. Solution of Lyapunov equations
3. GPU implementation
   - Matrix inversion
4. Iterative refinement
5. Conclusions
Outline

1. Truncation methods for model reduction

2. Solution of Lyapunov equations

3. GPU implementation
   - Matrix inversion

4. Iterative refinement

5. Conclusions
Balanced Truncation is an absolute error method, which aims at
\[
\min \| G - \hat{G} \|_\infty
\]
Composed of the following three steps:

**Step 1.** Solve the *coupled* Lyapunov matrix equations

\[
AW_c + W_c A^T + BB^T = 0,
\]
\[
A^T W_o + W_o A + C^T C = 0,
\]

for the observability and controllability Gramians, $W_c$ and $W_o$. Actually, we need the Cholesky factors $S$ and $R$ such that

\[
W_c = S^T S, \quad W_o = R^T R.
\]

$S$ and $R$ are dense.
Balanced Truncation is an absolute error method, which aims at

$$\min \| G - \hat{G} \|_\infty$$

Composed of the following three steps:

**Step 1.** Solve the *coupled* Lyapunov matrix equations

$$AW_c + W_c A^T + BB^T = 0,$$

$$A^T W_o + W_o A + C^T C = 0,$$

for the observability and controllability Gramians, $W_c$ and $W_o$. Actually, we need the Cholesky factors $S$ and $R$ such that

$$W_c = S^T S, \quad W_o = R^T R.$$  

$S$ and $R$ are dense.
Step 2. Compute the **Hankel singular values (HSV)** from

\[ SR^T = U\Sigma V^T = [U_1 \ U_2] \begin{bmatrix} \Sigma_1 & \Sigma_2 \\ \\ \end{bmatrix} \begin{bmatrix} V_1^T \\ V_2^T \end{bmatrix}, \]

with \( U, V, \) and \( \Sigma \) partitioned at a certain order \( r \).

The HSV in \( \Sigma = \text{diag}(\sigma_1, \ldots, \sigma_n) \), measure how much a state is involved in energy transfer from a given input to a certain output!
Step 3. In the square-root balance truncation (SRBT) method

\[ T_l = \Sigma_1^{-1/2} V_1^T R \quad \text{and} \quad T_r = S^T U_1 \Sigma_1^{-1/2}, \]

and \((\hat{A}, \hat{B}, \hat{C}, \hat{D}) = (T_l AT_r, T_l B, C T_r, D)\) for the TFM:

\[ \hat{G}(s) = C T_r (sI_n - T_l AT_r)^{-1} T_l B + D. \]

Computable error bound: \[ \| G - \hat{G} \|_\infty \leq 2 \sum_{k=r+1}^{n} \sigma_k. \]
Given \((A, B, C, D, x^0)\) with \(A\) large, and \(m, p \ll n\)...

How do we solve the previous numerical problems?

2. SVD of matrix product.
3. Application of the SRBT formulae to obtain the reduced-order model.
Outline

1. Truncation methods for model reduction
2. Solution of Lyapunov equations
3. GPU implementation
   - Matrix inversion
4. Iterative refinement
5. Conclusions
Solution of Lyapunov equations

Sign Function Method

Given $\alpha \in \mathbb{R}$,

$$
\text{sign}(\alpha) = \begin{cases} 
1 & \text{if } \alpha > 0, \\
-1 & \text{if } \alpha < 0, \\
\text{undefined} & \text{otherwise.}
\end{cases}
$$

For a matrix $A \in \mathbb{R}^{n \times n}$, $\text{sign}(A)$ is a function of the signs of its eigenvalues.

Given

$$
H = \begin{bmatrix} 
A & 0 \\
C^T C & -A^T 
\end{bmatrix}, \quad \text{sign}(H) = \begin{bmatrix} 
-I_n & 0 \\
2W_o & I_n 
\end{bmatrix},
$$

where $W_o$ is the observability Gramian.

So, how do we compute the sign function?
For $H = \begin{bmatrix} A & 0 \\ C^T C & -A^T \end{bmatrix}$ the classical Newton iteration boils down to

\[
A_{j+1} = \frac{1}{2} (A_j + A_j^{-1})/2, \quad A_0 = A,
\]

\[
R_{j+1} = \frac{1}{\sqrt{2}} \begin{bmatrix} R_j \\ R_j A_j^{-1} \end{bmatrix}, \quad R_0 = C,
\]

which converges to $R$, the Cholesky factor of $W_o$.

At each iteration $R_j$ is increased in $p$ rows ($p \Rightarrow$ number of outputs).

The computation of the inverse represents the main part of the computation ($O(2n^3)$ flops).
As in model reduction $R$ (and $S$) is usually rank-deficient the cost of the iteration and subsequent steps can be greatly reduced:

At the $j$th iteration, compute the rank-revealing QR (RRQR) factorization

$$\frac{1}{\sqrt{2}} \begin{bmatrix} R_j \\ R_j A_j^{-1} \end{bmatrix} = \bar{Q} \bar{R} \Pi$$

and then set

$$R_{j+1} = (\bar{R} \Pi)^T.$$ 

On convergence the iteration produces dense, full-rank $\hat{R}$ with $l \ll n$ columns, such that

$$\hat{R}^T \hat{R} \approx R^T R = W_o.$$
Outline

1. Truncation methods for model reduction
2. Solution of Lyapunov equations
3. **GPU implementation**
   - Matrix inversion
4. Iterative refinement
5. Conclusions
Hybrid approach for the sign function

Each step is performed in the most suitable device:

1. \[ A_{j+1} = \frac{1}{2} (A_j + A_j^{-1}) / 2, \quad A_0 = A \Rightarrow \text{Matrix inverse on GPU} \]

2. \[ R_{j+1} = \frac{1}{\sqrt{2}} \begin{bmatrix} R_j \\ R_j A_j^{-1} \end{bmatrix}, \quad R_0 = C \Rightarrow \text{GEMM on CPU or GPU} \]

3. RRQR \Rightarrow \text{Executed on CPU}
Matrix inversion

Via LU factorization

1. \( PA = LU \)
2. \( U \rightarrow U^{-1} \)
3. Solve the system \( XL = U^{-1} \) for \( X \)
4. Undo the permutations \( A^{-1} := XP \)

Implementation

- The algorithm sweeps through the matrix four times
- Presents a mild load imbalance, due to the work with triangular factors

Algorithm implemented by LAPACK
Matrix inversion

Via Gauss-Jordan elimination (GJE)
- Reordering of the computations of LU-based methods
- Requires the same arithmetic cost

Implementation
- The algorithm sweeps through the matrix once
- Most of the computations are highly parallel
**Algorithm:** \([A] := \text{GJE}_{\text{BLK}}(A)\)

Partition \(A \rightarrow \left( \begin{array}{c|c} \bar{A}_L & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)\)

where \(A_{TL}\) is 0 \(\times\) 0 and \(A_{BR}\) is \(n \times n\)

while \(m(A_{TL}) < m(A)\) do
  
  Determine block size \(b\)

  Repartition

  \[
  \left( \begin{array}{c|c} \bar{A}_L & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left( \begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{array} \right)
  \]

  where \(A_{11}\) is \(b \times b\)

  \[
  \begin{array}{l}
  \begin{bmatrix}
  A_{01} \\
  A_{11} \\
  A_{21}
  \end{bmatrix} := \text{GJE}_{\text{UNB}} \left( \begin{bmatrix}
  A_{01} \\
  A_{11} \\
  A_{21}
  \end{bmatrix} \right) \\
  A_{00} := A_{00} + A_{01}A_{10} \\
  A_{20} := A_{20} + A_{21}A_{10} \\
  A_{10} := A_{11}A_{10} \\
  A_{02} := A_{02} + A_{01}A_{12} \\
  A_{22} := A_{22} + A_{21}A_{12} \\
  A_{12} := A_{11}A_{12}
  \end{array}
  \]

  Unblocked Gauss-Jordan

  Matrix-matrix product

  Continue with

  \[
  \left( \begin{array}{c|c} \bar{A}_L & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left( \begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{array} \right)
  \]

endwhile

**Figure:** Blocked algorithm for matrix inversion via GJE without pivoting.
Matrix inversion (GJE)

**GPU implementation**
- The matrix is transferred to the GPU
- The inverse is computed **completely** on the GPU
- Result is transferred back to the CPU

**Hybrid implementation**
- GPU computes all the matrix-matrix products
- CPU computes the $GJE_{UNB}$
- Only small (column) panels are transferred
## Experimental Results

### Experimental setup

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU</strong></td>
<td>Dual Xeon QuadCore E5410</td>
</tr>
<tr>
<td><strong>CPU frequency</strong></td>
<td>2.33 Ghz</td>
</tr>
<tr>
<td><strong>RAM memory</strong></td>
<td>8 Gbytes</td>
</tr>
<tr>
<td><strong>GPU</strong></td>
<td>Tesla C1060</td>
</tr>
<tr>
<td><strong>Processor</strong></td>
<td>Nvidia GT200</td>
</tr>
<tr>
<td><strong>GPU frequency</strong></td>
<td>1.3 Ghz</td>
</tr>
<tr>
<td><strong>Video memory</strong></td>
<td>4 Gbytes DDR3</td>
</tr>
<tr>
<td><strong>Interconnection</strong></td>
<td>PCIExpress Gen2</td>
</tr>
<tr>
<td><strong>CUDA (CUBLAS) version</strong></td>
<td>2.1</td>
</tr>
<tr>
<td><strong>BLAS implementation</strong></td>
<td>GOTOBlas 1.26</td>
</tr>
<tr>
<td><strong>Driver version</strong></td>
<td>185.18</td>
</tr>
</tbody>
</table>

Results for matrices with $1000 \leq n \leq 8000$ and $b \leq 200$

Transfer times included in all results
Experimental Results

(Matrix inverse - GotoBLAS)

Matrix inversion on Caton2 + Goto 1.26

- LAPACK+CPU
- GJE+CPU
- GJE+GPU
- GJE+Hybrid

Model Reduction Problems on GPUs... Benner et al.
Experimental Results
(Single precision Matrix Sign Function - GotoBLAS)

![Graph showing performance comparison of different methods for computing the sign function on Caton2 + Goto 1.26](image)

- LAPACK
- GJE(CPU)
- GJE(GPU)
- GJE(Hybrid)

Model Reduction Problems on GPUs...
Experimental Results

(Double Precision Matrix Sign Function - GotoBLAS)

Sign Function on Caton2 + Goto 1.26

Matrix size vs. Time(s)

- LAPACK
- DGJE(CPU)
- DGJE(GPU)
- DGJE(Hybrid)

Model Reduction Problems on GPUs...
Outline

1. Truncation methods for model reduction
2. Solution of Lyapunov equations
3. GPU implementation
   - Matrix inversion
4. Iterative refinement
5. Conclusions
Given a Lyapunov equation:

\[ AX + XA^T = -BB^T \]

Goals

1. Exploit the single-precision capabilities of GPUs.
2. Get an approximation of the solution in low-precision arithmetic:
   \[ L = \text{ApproxLyap}(B), \quad X \approx LL^T \]
3. Refine the result to regain full accuracy.
Iterative refinement (II)

Let

\[ L_0 = \text{ApproxLyap}(B) \quad \text{SinglePrecision}(GPU) \]

to improve \( L_0 \) we construct a correction based on the residual

\[ \text{Res} = A L_0 L_0^T + L_0 L_0^T A^T + B B^T \quad \text{DoublePrecision} \]

and solve

\[ L_1 = \text{ApproxLyap}(\text{-Res}) \quad \text{SinglePrecision} \]

to get the correction term.

Problem

\[ \text{Res is usually indefinite} \]
Solution

Decompose Res into a positive definite and a negative definite part:

\[ \text{Res} = R_p R_p^T - R_n R_n^T \]

Each term corresponds to a Lyapunov equation (solved in SP):

\[ AX_p + X_p A^T = -R_p R_p^T \quad A(-X_n) + (-X_n) A^T = -R_n R_n^T \]

Then \( X_c = X_p + X_n \) solves the correction equation

\[ AX_c + X_c A^T = -\text{Res} \]

Corrected solution:

\[ X_1 = L_0 L_0^T + L_p L_p^T - L_n L_n^T \]
Numerical example (MATLAB)

- $A \rightarrow 900 \times 900$ symmetric negative definite matrix
- $\text{it} \rightarrow$ number of sign function iterations
- $\text{tol} \rightarrow$ tolerance for sign function
- $\text{res}_i \rightarrow$ residual after $i$ steps of iterative refinement

<table>
<thead>
<tr>
<th>tol</th>
<th>$10^{-2}$</th>
<th>$10^{-4}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>it</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>$\text{res}_0$</td>
<td>$7 \times 10^{-2}$</td>
<td>$7 \times 10^{-4}$</td>
</tr>
<tr>
<td>$\text{res}_1$</td>
<td>$5 \times 10^{-4}$</td>
<td>$7 \times 10^{-7}$</td>
</tr>
<tr>
<td>$\text{res}_2$</td>
<td>$3 \times 10^{-5}$</td>
<td>$7 \times 10^{-11}$</td>
</tr>
<tr>
<td>$\text{res}_3$</td>
<td>$6 \times 10^{-7}$</td>
<td></td>
</tr>
<tr>
<td>$\text{res}_4$</td>
<td>$1 \times 10^{-7}$</td>
<td></td>
</tr>
<tr>
<td>$\text{res}_5$</td>
<td>$1 \times 10^{-9}$</td>
<td></td>
</tr>
<tr>
<td>$\text{res}_6$</td>
<td>$1 \times 10^{-10}$</td>
<td></td>
</tr>
<tr>
<td>$\text{res}_7$</td>
<td>$1 \times 10^{-12}$</td>
<td></td>
</tr>
<tr>
<td>Time (s)</td>
<td>$6.5 + 1$</td>
<td>$8 + 0.5$</td>
</tr>
</tbody>
</table>

$\Rightarrow$ 15 seconds in double precision
1. Truncation methods for model reduction
2. Solution of Lyapunov equations
3. GPU implementation
   - Matrix inversion
4. Iterative refinement
5. Conclusions
Solution of (large) model reduction problems applying GPUs

Truncation Methods → Lyapunov equations

Hybrid approach for the Sign Function to solve Lyapunov equations

Iterative refinement approach to combine full-accuracy and high performance
Thank you!

More information...
- HPCA Group at University Jaume I (http://www.h pca.uji.es)
- {figual, quintana, remon}@icc.uji.es