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

Peter Benner[1] Pablo Ezzatti[2] Francisco Igual[3]
Daniel Kressner[4] Enrique S. Quintana-Ortí[3]
Alfredo Remón[3]

[1]Fakultät für Mathematik, Chemnitz University of Technology, Chemnitz (Germany)

[2]Centro de Cálculo–Instituto de la Computación, Universidad de la República, Montevideo (Uruguay)

[3]Seminar für angewandte Mathematik, ETH Zürich, Zurich (Switzerland).

[4]Depto. de Ingeniería y Ciencia de Computadores, Universidad Jaume I, Castellón (Spain).

UNIVERSITAT JAUME•I

Dynamical Linear Systems

Linear time-invariant systems:

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t), \qquad t > 0, \qquad x(0) = x^0, \\ y(t) &= Cx(t) + Du(t), \qquad t \geq 0, \end{aligned}$$

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

Corresponding TFM:

$$G(s) = C(sI_n - A)^{-1}B + D.$$

Goal for Model Reduction

Find a reduced-order model

$$\begin{aligned} \dot{\hat{x}}(t) &= \hat{A}\hat{x}(t) + \hat{B}u(t), \qquad t > 0, \qquad \hat{x}(0) = \hat{x}^0, \\ \hat{y}(t) &= \hat{C}\hat{x}(t) + \hat{D}u(t), \qquad t \geq 0, \end{aligned}$$

of order $r \ll n$ such that the output error

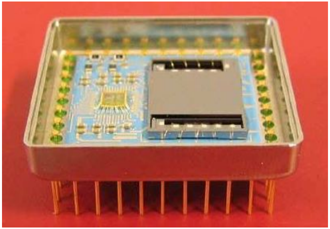
$$y - \hat{y} = Gu - \hat{G}u = (G - \hat{G})u$$

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?

Outline

Outline

Balanced Truncation (I)

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

$$\begin{aligned} AW_c + W_cA^T + BB^T &= 0, \\ A^TW_o + W_oA + C^TC &= 0, \end{aligned}$$

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

$$W_c = S^TS, \quad W_o = R^TR.$$

S and R are dense.

Balanced Truncation (I)

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

$$\begin{aligned} AW_c + W_cA^T + BB^T &= 0, \\ A^TW_o + W_oA + C^TC &= 0, \end{aligned}$$

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

$$W_c = S^TS, \quad W_o = R^TR.$$

S and R are dense.

Balanced Truncation (II)

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 Σ partitioned at a certain order r.

The HSV in $\Sigma = \operatorname{diag}(\sigma_1, \ldots, \sigma_n)$, measure how much a state is involved in energy transfer from a given input to a certain output!

Balanced Truncation (III)

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

$$\hat{G}(s) = C T_r (s I_n - T_l A T_r)^{-1} T_l B + D.$$

- Computable error bound: $\|G - \hat{G}\|_\infty \leq 2 \sum_{k=r+1}^{n} \sigma_k.$

Balanced Truncation (IV)

Given (A, B, C, D, x^0) with A large, and $m, p \ll n$...

How do we solve the previous numerical problems?

1. Coupled Lyapunov equations.
2. SVD of matrix product.
3. Application of the SRBT formulae to obtain the reduced-order model.

Outline

Sign Function Method

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

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

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

Given

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

where W_o is the observability Gramian.

So, how do we compute the sign function?

Sign Function Methods (I)

For $H = \begin{bmatrix} A & 0 \\ C^T C & -A^T \end{bmatrix}$ the classical Newton iteration boils down to

$$\begin{aligned} A_{j+1} &= \frac{1}{2}(A_j + A_j^{-1})/2, \quad A_0 = A, \\ R_{j+1} &= \frac{1}{\sqrt{2}} \begin{bmatrix} R_j \\ R_j A_j^{-1} \end{bmatrix}, \quad R_0 = C, \end{aligned}$$

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).

Sign Function Methods (II)

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 jth iteration, compute the rank-revealing QR (RRQR) factorization

$$\frac{1}{\sqrt{2}}\left[\begin{array}{c} R_j \\ R_j A_j^{-1} \end{array}\right] = \bar{Q}\bar{R}\Pi$$

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

Hybrid approach for the sign function

Hybrid approach

Each step is performed in the most suitable device:

1. $A_{j+1} = \frac{1}{2}(A_j + A_j^{-1})/2, \quad A_0 = A$ $\Rightarrow$ Matrix inverse on GPU
2. $R_{j+1} = \frac{1}{\sqrt{2}} \begin{bmatrix} R_j \\ R_j A_j^{-1} \end{bmatrix}, \quad R_0 = C$ $\Rightarrow$ GEMM on CPU or GPU
3. RRQR $\Rightarrow$ 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

Matrix inversion (GJE)

Algorithm: $[A] := \text{GJE}_{\mathsf{BLK}}(A)$

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

where A_{TL} is 0×0 and A_{BR} is $n \times n$

while $m(A_{TL}) < m(A)$ **do**

Determine block size b

Repartition

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

where A_{11} is $b \times b$

$\begin{bmatrix} A_{01} \\ A_{11} \\ A_{21} \end{bmatrix}$	$:=$	$GJE_{\mathsf{UNB}}\left(\begin{bmatrix} A_{01} \\ A_{11} \\ A_{21} \end{bmatrix}\right)$	Unblocked Gauss-Jordan
A_{00}	$:=$	$A_{00} + A_{01}A_{10}$	Matrix-matrix product
A_{20}	$:=$	$A_{20} + A_{21}A_{10}$	Matrix-matrix product
A_{10}	$:=$	$A_{11}A_{10}$	Matrix-matrix product
A_{02}	$:=$	$A_{02} + A_{01}A_{12}$	Matrix-matrix product
A_{22}	$:=$	$A_{22} + A_{21}A_{12}$	Matrix-matrix product
A_{12}	$:=$	$A_{11}A_{12}$	Matrix-matrix product

Continue with

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

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

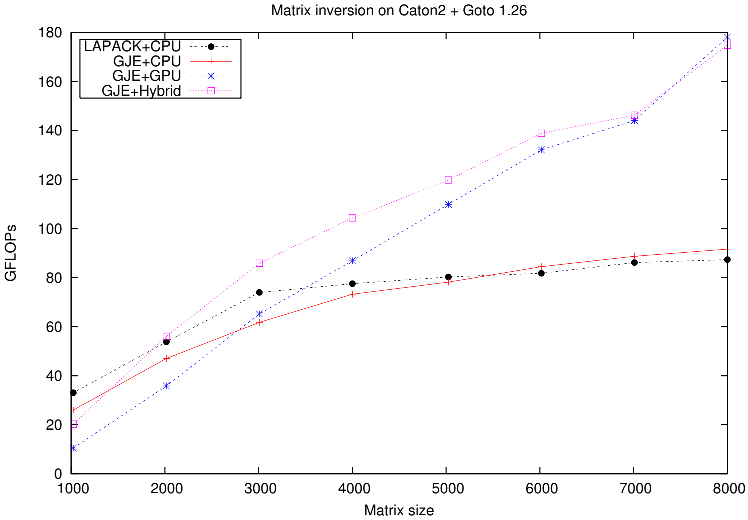
CPU	Dual Xeon QuadCore E5410
CPU frequency	2.33 Ghz
RAM memory	8 Gbytes
GPU	Tesla C1060
Processor	Nvidia GT200
GPU frequency	1.3 Ghz
Video memory	4 Gbytes DDR3
Interconnection	PCIExpress Gen2
CUDA (CUBLAS) version	2.1
BLAS implementation	GOTOBlas 1.26
Driver version	185.18

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

Transfer times included in all results

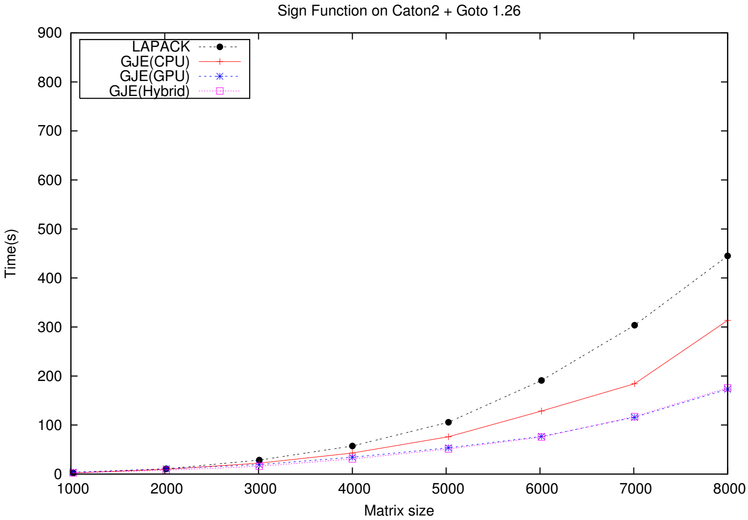
Experimental Results

(Matrix inverse - GotoBLAS)



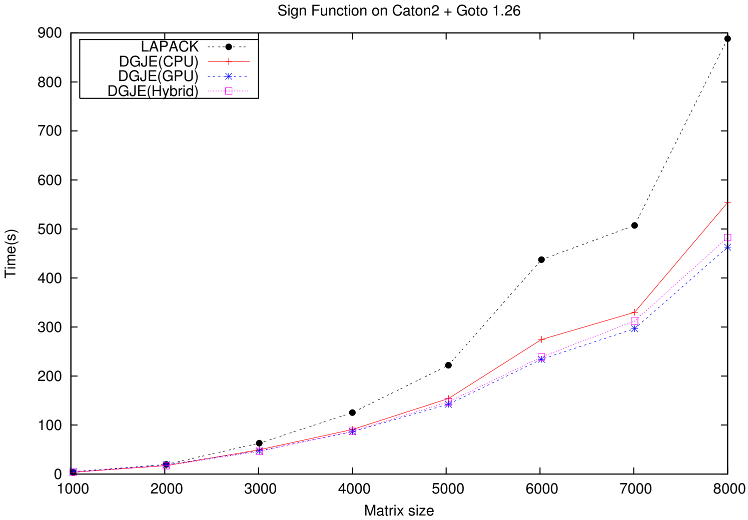
Experimental Results

(Single precision Matrix Sign Function - GotoBLAS)



Experimental Results

(Double Precision Matrix Sign Function - GotoBLAS)



Outline

Iterative refinement (I)

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:

$$L = \text{ApproxLyap}(B), \qquad X \approx LL^T$$

3. Refine the result to regain full accuracy.

Iterative refinement (II)

Let

$$L_0 = \text{ApproxLyap}(B) \qquad SinglePrecision(GPU)$$

to improve L_0 we construct a correction based on the residual

$$\text{Res} = AL_0L_0^T + L_0L_0^TA^T + BB^T \qquad DoublePrecision$$

and solve

$$L_1 = \text{ApproxLyap}(-\text{Res}) \qquad SinglePrecision$$

to get the correction term.

Problem

Res is usually indefinite

Iterative refinement (III)

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):

$$AX_p + X_p A^T = -R_p R_p^T \qquad A(-X_n) + (-X_n)A^T = -R_n R_n^T$$

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
- it $\rightarrow$ number of sign function iterations
- tol $\rightarrow$ tolerance for sign function
- res$_i$ $\rightarrow$ residual after *i* steps of iterative refinement

tol	10^{-2}	10^{-4}
it	4	5
res_0	7×10^{-2}	7×10^{-4}
res_1	5×10^{-4}	7×10^{-7}
res_2	3×10^{-5}	7×10^{-11}
res_3	6×10^{-7}	
res_4	1×10^{-7}	
res_5	1×10^{-9}	
res_6	1×10^{-10}	
res_7	1×10^{-12}	
Time (s)	$6.5 + 1$	$8 + 0.5$

Sign function

Iterative refinement

$\Rightarrow$ 15 seconds in double precision

Outline

Concluding remarks

- Solution of (large) model reduction problems applying GPUs
- Truncation Methods $\longrightarrow$ 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.hpca.uji.es)
- {figual, quintana, remon}@icc.uji.es