Parallelizing the Computation of Exponential of a Matrix

Kaushik Kulkarni
&
Suryansh Bhargava

Under the Guidance of
Prof. Shivasubramanian Gopalakrishnan

Department of Mechanical Engineering
Indian Institute of Technology, Bombay

April 30, 2016

Overview

1. Matrix Exponential
2. Pade Approximation
3. Current Project
4. Serial Code
5. OpenMP Code
6. CUDA Code
7. Conclusion
8. Future Work Proposed
9. References

Matrix Exponential

- Matrix Exponential forms a very important part of Control Engineering involving high degrees of Freedom.
- The solution of $\dot{x}(t) = Ax$ is given by $x(t) = e^{At}$.
- But, when the system involves higher degrees of freedom the equation of motion becomes $\dot{\vec{x}}(t) = A\vec{x}$, A being a matrix of size $n \times n$ is given by $\vec{x}(t) = e^{At}$, and the RHS is found out using the **Matrix Exponential**.

- The matrix exponential is defined as follows:

$$e^A = I + A + \frac{A^2}{2!} + \frac{A^3}{3!} + \dots \quad (1)$$

$$A \rightarrow N \times N \text{ Matrix} \quad (2)$$

- But direct calculation of the Matrix Exponential is computationally unstable process as it includes the addition of numbers which are very separated in magnitudes, hence we need to look for different algorithms.

Pade Approximation

- The Pade Approximant is a better approximation to the Taylor's series.
- As by calculating the third order terms, we can get the results with precision upto 7^{th} order.
- The Pade Approximant of 3^{rd} order is given by:

$$\left(e^A\right)_3 = \frac{120I + 60A + 12A^2 + A^3}{120I - 60A + 12A^2 - A^3} \tag{3}$$

$$\left(e^A\right)_5 = \frac{30240I + 15120A + 3360A^2 + 420A^3 + 30A^4 + A^5}{30240I - 15120A + 3360A^2 - 420A^3 + 30A^4 - A^5} \tag{4}$$

Modification to the Pade Approximant

- Even this method turns out to be computationally unstable as N increase and the norm of the matrix increases.
- Hence, to deal with this problem **Scaling and Squaring Algorithm** has been used:
  1. Scales down a matrix by an order of power of 2.
  2. Then compute the Pade Approximant of the new Matrix.
  3. Compensate the scale down by appropriately squaring it.

Modification to the Pade Approximant(*Contd.*)

Let's say we need to find the exponential of the matrix A. We choose a σ, such that

$$2^{-\sigma}||A||_\infty < 1$$

Then, we define another Matrix B such that:

$$B = 2^{-\sigma}A$$

Eventually the Matrix B being a normalized matrix, the Pade Approximant is computationally stable. Then to compensate the scaling down the Matrix e^B is sqaure σ times.

$$e^A = \left(e^B\right)^{2^\sigma}$$

Current Project

In the current implementation, we have focused on 3 implementations:

- Serial Code
- OpenMP Code
- CUDA Code

The details of the implementation have explained in the following slides. In the current study, comparison of the data might not be a good method to compare the methods as all of them are a bit different in implementations.

Serial Code

There have been two implementations of the Serial Code:

- Normal Serial Version
  - The sub-routines to calculate the Matrix Multiplication and Gaussian Elimination were defined by us.
  - Result: The time taken for calculating the exponential of 1024×1024 was **214.227** seconds.
  - Hence, this implementation is "useless".
- BLAS and LAPACK Version:
  - Third level BLAS and LAPACK sub-routines were used.
  - Result: The time taken for calculating the exponential of 1024×1024 was **9.42** seconds.
  - Hence, this implementation is very efficient and hence we must continue to use these.

OpenMP Code

- Here, again to parallelize the code directly working on the crude user-built functions would not be an efficient method.
- The problem is looked by the perspective of $N - 2^{nd}$ level BLAS functions.
- But owing to the efficiency of Third-Level BLAS functions the speedup obtained using 4 processors was not sufficient to meet the implementation of the Serial Version.
- Result: Code Parallelized but could not match the time of third level BLAS functions used. Execution time: **15.04** seconds.

CUDA Code

- User built functions were built as kernels.
- But again these were not efficient functions, compared to the earlier crude functions quite high speed-up is obtained but still it cannot be compared to third level BLAS functions.
- Result: Code Parallelized but could not match the time of third level BLAS functions used. Execution time: **19.46** seconds.

Conclusion

- The main challenge of implementing the **Scaling and Squaring Algorithm** was successful, and the results have been verified with those of the `expm(A)` function of MATLAB, and the results were found to be close upto the 6^{th} decimal place.
- The code has been parallelized using OpenMP(using 4 processors) and CUDA.
- Still, both of them could not "compete" with the Serial Version.
- Hence, BLAS and LAPACK functions are very powerful and must be used appropriately.

Future Work

- In future to get this code accepted as a standard code the CUDA implementation should be better, as it involves lot of capital investment.
- One major improvement could be obtained by changing the layout of the matrix which could help remove the cache misses, which could significantly boost the results.

References

1. Cleve Moler and Charles Van Loan, *Nineteen Dubious Ways to Compute the Exponential of a Matrix*, SIAM REVIEW Vol. 45, No. 1, pp. 349.
2. Nicholas J. Higham, *The Scaling and Squaring Method for the Matrix Exponential Revisited*, SIAM J. MATRIX ANAL. APPL. Vol. 26, No. 4, pp. 1179 – 1193.