Universidade do Minho
Escola de Engenharia

Pedro Filipe Araújo Costa

Efficient Computation of the Matrix Square Root in Heterogeneous Platforms

Setembro de 2013

Universidade do Minho
Escola de Engenharia
Departamento de Informática

Pedro Filipe Araújo Costa

Efficient Computation of the Matrix Square Root in Heterogeneous Platforms

Dissertação de Mestrado
Mestrado em Engenharia Informática

Trabalho realizado sob orientação de
Professor Alberto Proença
Professor Rui Ralha

Setembro de 2013

Anexo 3

DECLARAÇÃO

Nome

Pedro Filipe Araújo Costa

Endereço electrónico: pg19830@alunos.uminho.pt Telefone: 913 725 975 / 924 189 979

Número do Bilhete de Identidade: 13775276

Título dissertação □/tese □

Efficient Computation of the Matrix Square Root in Heterogeneous Platforms

Orientador(es):

Professor Alberto Proença

Professor Rui Ralha Ano de conclusão: 2013

Designação do Mestrado ou do Ramo de Conhecimento do Doutoramento:

Mestrado em Engenharia Informática

Nos exemplares das teses de doutoramento ou de mestrado ou de outros trabalhos entregues para prestação de provas públicas nas universidades ou outros estabelecimentos de ensino, e dos quais é obrigatoriamente enviado um exemplar para depósito legal na Biblioteca Nacional e, pelo menos outro para a biblioteca da universidade respectiva, deve constar uma das seguintes declarações:

1. É AUTORIZADA A REPRODUÇÃO INTEGRAL DESTA TESE/TRABALHO APENAS PARA EFEITOS DE INVESTIGAÇÃO, MEDIANTE DECLARAÇÃO ESCRITA DO INTERESSADO, QUE A TAL SE COMPROMETE;

2. É AUTORIZADA A REPRODUÇÃO PARCIAL DESTA TESE/TRABALHO (indicar, caso tal seja necessário, nº máximo de páginas, ilustrações, gráficos, etc.), APENAS PARA EFEITOS DE INVESTIGAÇÃO, , MEDIANTE DECLARAÇÃO ESCRITA DO INTERESSADO, QUE A TAL SE COMPROMETE;

3. DE ACORDO COM A LEGISLAÇÃO EM VIGOR, NÃO É PERMITIDA A REPRODUÇÃO DE QUALQUER PARTE DESTA TESE/TRABALHO

Universidade do Minho, ____/____/_______

Assinatura: ____

Acknowledgments

put your bigass heart out there

and don’t forget the money you were given

Abstract

an awesome abstract

Resumo

Computação Eficiente da Raíz Quadrada de Matrizes em Plataformas Heterogéneas

Um resumo espetacular

Contents

List of Figures

Page

List of Tables

Listings

Introduction

Case Study: The Matrix Square Root on Heterogeneous Platforms

The square root of a matrix A is any matrix X that satisfies the equation

$$A = X^2 \ , \tag{1}$$

When it exists, it is not unique, but when A does not have any real negative eigenvalues it has a unique square root whose eigenvalues all lie in the open right half plane (i.e. have non-negative real parts) [1]. This is the so-called principal square root $A^{1/2}$ and since it is the one usually needed in applications, it is the one the algorithm and respective implementations in this document are focused in computing.

The Schur method of Björck and Hammarling [2] is the most numerically stable method for computing the square root of a matrix. It starts by reducing the matrix A to upper triangular form T. By computing the square root of T (let U be such a matrix), also upper triangular, the same recurrence relation allows to compute X, thus solving Equation (1). The matrix U is computed by solving

$$U_{ii}^2 = T_{ii} \ , \tag{2}$$

$$U_{ii}U_{ij} + U_{ij}U_{jj} = T_{ij} - \sum_{k=i+1}^{j-1} U_{ik}U_{kj} \ , \tag{3}$$

This method is implemented in MATLAB as the `sqrtm` and `sqrtm_real` functions [3].

The scope of this document focus on expanding the work of Deadman et al. by solving Equations (2) and (3) using the resources available in modern heterogeneous platforms.

1 Strategies

Equations (2) and (3) describe an algorithm where each element depends on those at its left in the same row and those below in the same column. Consequently, the algorithm can be implemented either a column/row or a superdiagonal at a time.

While the first strategy (column/row) is preferred for any serial implementation due to a more efficient use of cache memory (better locality), it presents almost[1] no opportunities for parallelism since no more than one element is ready to be computed at any given time.

On the other hand, computing a superdiagonal at a time allows for several elements to be computed in parallel, as all the dependencies were computed in the previous superdiagonals.

2 Methods

In the previous work of Deadman et al. [4], the authors devised a blocked algorithm to compute the square root of a matrix. Similar to the original, the blocked algorithm lets U_{ij} and T_{ij} in Equations (2) and (3) refer to square blocks of dimension $m \ll n$ (n being the dimension of the full matrix).

The blocks U_{ii} (in the main diagonal) are computed using a non-blocked implementation as described previously. The remaining blocks are computed by solving the Sylvester equations (3). The dependencies can be solved with matrix multiplications and sums. Where available, these operations can be performed using the LAPACK `xTRSYL` and Level 3 BLAS `xGEMM` calls.

Two blocked methods were devised in [4]: a standard blocking method, where the matrix is divided once in a set of well defined blocks; and a recursive blocking method, where blocks are recursively divided into smaller blocks, until a threshold

[1] It is possible for this strategy to solve its dependencies in parallel. The following chapters will show this with more detail.

is reached. This allows for larger calls to `xGEMM`, and the Sylvester equation can be solved using a recursive algorithm by [5].

While the recursive method achieved better results in the serial implementations, using explicit parallelism the multiple synchronization points at each level of the recursion decreased the performance, favouring the standard blocking method. Given the devices targeted by this document's work, where explicit parallelism is required to take full advantage of the architecture, the recursive method is ignored.

Using the same terminology as in [4], the non-blocked and standard blocked methods will be referred to hereafter as the point and block method, respectively.

Multicore

Intel MIC

CUDA

3 Implementation

Unlike Many Integrated Core (MIC) devices, Graphics Processing Units (GPUs) differ greatly from Central Processing Units (CPUs). As such, this kind of devices have a distinct programming model which require a shift in the way the programmer thinks about the algorithm. Consequently, little of the code implemented in the previous chapters is reusable in a CUDA implementation of the matrix square root algorithm.

First of all, the NVidia compiler proved to be incompatible with the Armadillo library. While the usage of this library was reduced to loading the matrix file and outputting the result in the previous chapter, it had to be completely removed from the CUDA implementation. The reason for this is an incompatibility of the NVidia compiler with recent versions of the GNU compiler. Since older versions of the GNU compiler are required, some of the more recent features of the C++ language (used by Armadillo) are rejected. While the incompatibility was isolated and found not to be related to the IO operations, the library is prepared to have all its headers used simultaneously, which resulted in very tight coupling.

Removing Armadillo implied that the code to load the matrix files had to be ported to a compatible implementation. To ease the task, `ARMA_ASCII` format was selected as the reference format. This is the simplest text format in Armadillo, with the files having a small header (meant to identify the data type and the dimensions of the matrix) immediately followed by the matrix content.

Contrary to the implementations in the previous chapters, a CUDA implementation of this algorithm can not take advantage of an optimized BLAS library. The existing libraries assume that its kernels will have the entire device available. As seen in the previous implementations, this is not the case. Both the point and block methods contain independent parallel calls to BLAS functions. All these functions have to be reimplemented in the scope of this algorithm.

Conclusions

Future Work

Bibliography

[1] Nicholas J. Higham. *Functions of Matrices: Theory and Computation*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2008. ISBN: 978-0-898716-46-7.

[2] Åke Björck et al. "A Schur method for the square root of a matrix". In: *Linear Algebra and its Applications* 52–53 (1983), pp. 127–140.

[3] Nicholas J. Higham. *The Matrix Function Toolbox*. URL: `http://www.ma.man.ac.uk/~higham/mftoolbox`.

[4] Edvin Deadman et al. *A Recursive Blocked Schur Algorithm for Computing the Matrix Square Root*. MIMS EPrint 2012.26. Manchester, UK: Manchester Institute for Mathematical Sciences, University of Manchester, Jan. 2012. URL: `http://eprints.ma.man.ac.uk/1775`.

[5] Isak Jonsson et al. "Recursive blocked algorithms for solving triangular systems – Part I: one-sided and coupled Sylvester-type matrix equations". In: *ACM Trans. Math. Softw.* 28.4 (Dec. 2002), pp. 392–415.