Sustainable Computing: Informatics and Systems xxx (2015) xxx–xxx

Contents lists available at ScienceDirect

Sustainable Computing: Informatics and Systems

journal homepage: www.elsevier.com/locate/suscom

A fast, hybrid, power-efficient high-precision solver for large linear systems based on low-precision hardware

C.M. Angerer*, R. Polig, D. Zegarac, H. Giefers, C. Hagleitner, C. Bekas, A. Curioni

IBM Research – Zurich, Säumerstrasse 4, Rüschlikon, Switzerland

ARTICLE INFO

Article history:
Received 25 June 2014
Received in revised form
11 September 2015
Accepted 8 October 2015
Available online xxx

Keywords:
Hybrid computing
Heterogeneous computing
Hardware acceleration
Power-efficient linear system solving
GPU
Field programmable gate arrays

ABSTRACT

In recent years, the amount of data produced has been exploding at a rate far greater than the increase in computing power of even large supercomputers. As a result, modern computer systems are unable to analyze all the available data – a situation that will become even worse in the foreseeable future. We follow an approach to data analytics where the computational complexity is fundamentally reduced by performing the majority of the computation in an approximated or even stochastic framework while the high precision solution is guaranteed by an iterative refinement process.

This paper presents a parallel heterogeneous system implementing a mixed-precision iterative refinement solver for large linear systems, which is a building block for many other complex algorithms. In our solver, the backward step is implemented as a novel variant of the conjugate gradient (CG) method running on an FPGA using fixed point data types. The low precision of the backward step is compensated for by the forward step running in high precision on a GPU, which iteratively updates the current solution until a given working precision has been reached.

We have implemented our CG solver using Altera's OpenCL SDK for FPGAs and use NVIDIA's CUBLAS library for the forward step on the GPU. Through the combination of GPU and FPGA we were able to achieve a speedup of 3.7× for large dense 24,064 × 24,064 matrices and require 3.5× less energy per solved right-hand side compared to a tuned multi-threaded CPU solver based on the ATLAS linear algebra library.

1. Introduction

Modern applications produce data at previously unimaginable rates. The growth in available data significantly exceeds the rate at which we improve even our best supercomputers. In effect, solely relying on power and performance improvements of the hardware is not enough to catch up with the growing amount of data.

In order to address the increasing gap between workload and available compute power, we follow an approach to data analytics where the majority of the computation is performed in an approximated or even stochastic way while the final result is achieved in a high working precision through an iterative process.

Many critical data analytics applications in key disciplines such as engineering, medicine, physics, machine learning, statistics, finance, and big data can be formulated as an iterative refinement problem with three basic steps:

1. In a solving step, compute an approximate solution. This is the computationally expensive part.
2. In a subsequent update step, compute the remaining error and adapt the current solution. Compared to the backward step, computing the error is often a relatively inexpensive operation.
3. Repeat with Step 1 until the desired high precision is achieved.

While Step 1 affects the convergence speed of the overall solving process, the precision of the final result is achieved only by Step 2 and Step 3. The insight underlying our research is that for many problems the iterative refinement process converges quickly even if Step 1 produces perturbed inexact solutions with a relatively large error [2]. By exploring ways to trade the precision of Step 1 for an improvement in power and performance, we can tune the overall system performance to a given problem.

In this paper we present a parallel heterogeneous system implementing a mixed-precision iterative refinement solver. In our system, the solving Step 1 is implemented as a novel bounded

* Corresponding author. Tel.: +49 15118904142.
E-mail addresses: han@zurich.ibm.com (C.M. Angerer), pol@zurich.ibm.com (R. Polig), dze@zurich.ibm.com (D. Zegarac), hgi@zurich.ibm.com (H. Giefers), hle@zurich.ibm.com (C. Hagleitner), bek@zurich.ibm.com (C. Bekas), cur@zurich.ibm.com (A. Curioni).

http://dx.doi.org/10.1016/j.suscom.2015.10.001
2210-5379/

variant of the Conjugate Gradient (CG) method running on a custom FPGA design. Further, the update and error computation of Step 2 is done on a GPU, and a CPU-based host is managing the overall flow of computation associated with Step 3. Our system design allows for relaxing the accuracy of the inner solver in a controlled way:

Small number of CG iterations. In the inner CG solver, we use a (compared to the matrix size n) small number of iterations $i \ll n$. The overall cost of one run of the inner solver is then $O(i * n^2) \equiv O(n^2)$ because the cost of each CG step is dominated by a matrix–vector product.

Fixed point data types with configurable bit widths. All vector and matrix elements in our CG solver have tight bounds in the interval $[-1,1]$. This interval allows us to use a fixed point number representation with no integer bits and a per-vector/matrix configurable number of fractional bits: Each bit added increases memory consumption but also improves the precision.

Sparsification. When transforming the matrix A from floating point to low-precision fixed point, we divide A into two (or more) slices containing the most significant bits A_{msb} and least significant bits A_{lsb}. By combining the corresponding elements in the different slices, we can step-wise adapt the precision of the matrix multiplication at runtime. For a preconditioned matrix A, A_{msb} tends to become sparse, meaning that most of its elements are zero. We exploit this fact by applying a sparse matrix multiplication instead of a dense matrix multiplication during the low-precision rounds.

The main contributions of this paper are:

- A variant of the Conjugate Gradient method with mostly bounded values that can be implemented using fixed point data types or a combination of fixed point and floating point types to get the advantages of both representations.
- A heterogeneous hardware design and architecture implementing a mixed-precision iterative refinement method around our bounded CG algorithm. The host can dynamically adapt the precision of the CG solver in each round, enabling the system to trade precision for performance and power consumption.
- A thorough evaluation of power and performance characteristics for large dense linear systems. We evaluate multiple variations of our hardware design for different combinations of float/fixed point representations and different matrix.

The remainder of the paper is organized as follows. Section 2 presents some mathematical background on the Conjugate Gradient Method and mixed-precision iterative refinement. The changes we apply to the CG algorithm and the data types used are discussed in Section 3. Section 4 describes the architecture of our hardware design, which we evaluate in Section 5. Finally, Section 6 presents related work before we conclude in 7.

2. Background

This section presents some mathematical background for the Conjugate Gradient Method and mixed-precision iterative refinement.

2.1. Conjugate Gradient method

The Conjugate Gradient method [13] is an iterative method for solving large systems of linear equations of the form $Ax = b$, where b and x are vectors of length n and the $n \times n$ square matrix A is symmetric (i.e., $A^T = A$) and positive definite (i.e., for every nonzero vector x, $x^T Ax > 0$).

Algorithm 1 shows the CG algorithm. Most of the work of the CG solver is done in the loop between lines 5 and 14. The computationally most expensive operation is by far the matrix–vector multiplication on line 6 at the beginning of the loop. The tail of the loop consists of two vector dot products on lines 7 and 11 as well as vector additions with scaling on lines 8, 9, and 12. The remaining operations are simple scalar divisions.

Algorithm 1. The Conjugate Gradient Algorithm

Input: Matrix A, vector b, error tolerance ϵ
Output: x such that $||Ax - b||_2 \leq \epsilon ||b||_2$

1: $d \leftarrow b$
2: $r \leftarrow b$
3: $\delta_0 \leftarrow r^T r$
4: $\delta_{new} \leftarrow \delta_0$
5: **repeat**
6: $q \leftarrow Ad$
7: $\alpha \leftarrow \frac{\delta_{new}}{d^T q}$
8: $x \leftarrow x + \alpha d$
9: $r \leftarrow r - \alpha q$
10: $\delta_{old} \leftarrow \delta_{new}$
11: $\delta_{new} \leftarrow r^T r$
12: $\beta \leftarrow \frac{\delta_{new}}{\delta_{old}}$
13: $d \leftarrow r + \beta d$
14: **until** $\delta_{new} \leq \epsilon^2 \delta_0$

CG can be run as a stand-alone solver or it can be used, as done in this work, as the inner solver in an iterative refinement context. We chose the CG method due to a number of valuable computational properties:

1. Unlike other dense methods, such as Cholesky factorization, CG does not alter the original matrix A and it does not require additional storage for matrix factors such as triangular matrices.
2. For a given matrix size n, the memory requirement of CG is constant.[1] As can be seen from Algorithm 1, the CG solver only needs to keep the 4 vectors of length n: d, q, x, and r.
3. In exact arithmetic, CG would converge in at most n steps. In practice, rounding errors may require CG to run in $\gg n$ steps before it finds a solution with the requested precision. However, an approximation that allows the outer iterative refinement to progress towards the solution is usually found in $\ll n$ CG steps.
4. CG is based on matrix–vector products and vector dot-products which are operations that can be well parallelized in hardware.

2.2. Mixed-precision iterative refinement

Mixed-precision iterative refinement is a well-known method [20,10] that exploits the higher performance of modern hardware for single precision operations while still guaranteeing a high precision result. In fact, provided that the system is not too ill-conditioned mixed-precision iterative refinement produces a solution correct to the working precision.

The underlying idea of mixed-precision iterative refinement is to iteratively update an approximated high precision solution with a low-precision error correction term. The error correction term is hereby computed by an *inner correction solver*, such as a CG solver presented in the previous section.

[1] This is in fact true for all Krylov subspace methods, which build an orthonormal basis through a simple three term recurrence of the basis vectors.

Please cite this article in press as: C.M. Angerer, et al., A fast, hybrid, power-efficient high-precision solver for large linear systems based on low-precision hardware, Sustain. Comput.: Inform. Syst. (2015), http://dx.doi.org/10.1016/j.suscom.2015.10.001