HOKKAIDO UNIVERSITY

Title	Sparse Linear Solvers for High-Performance Computing Devices [an abstract of entire text]
Author(s)	鈴木, 謙吾
Description	この博士論文全文の閲覧方法については、以下のサイトをご参照ください。 https://www.lib.hokudai.ac.jp/dissertations/copy-guides/
Degree Grantor	北海道大学
Degree Name	博士(情報科学)
Dissertation Number	甲第16172号
Issue Date	2024-09-25
Doc URL	https://hdl.handle.net/2115/93445
Type	doctoral thesis
File Information	Kengo_Suzuki_summary.pdf

Hokkaido University Collection of Scholarly and Academic Papers : HUSCAP

学 位 論 文 内 容 の 要 約

博士の専攻分野の名称　博士（情報科学）　氏名　鈴木 謙吾

学 位 論 文 題 名

Sparse Linear Solvers for High-Performance Computing Devices

(高性能計算デバイスのための疎行列線形ソルバ)

Over the decades, significant improvements in computer performance have made numerical analysis an essential tool across many academic disciplines, including science, engineering, and medicine. Even today, there is an ongoing need for enhanced numerical analysis techniques to conduct more detailed and larger-scale analyses. In many numerical analysis applications, the objects of the analysis are formulated as systems of linear equations with large, sparse coefficient matrices (sparse linear systems). Therefore, the computational algorithms that solve them, sparse linear solvers, often determine the performance and accuracy of the entire application. Thus, enhancing sparse linear solvers is an important research issue that will contribute to the advancement of numerical analysis and, in turn, various academic disciplines. Hence, I consider several approaches to improve sparse linear solvers in this research.

The simplest way to enhance sparse linear solvers is to implement the same algorithm on a new, higher-performance computer, but this requires the algorithm to take full advantage of the computing resources of the new computer. In other words, the renewed computer must have the same architecture as the old one. Unfortunately, for more than the past ten years, the increase in clock frequency of computing devices has stagnated mainly due to challenges in power consumption. Therefore, many modern computing devices include distinctive architectures to achieve high performance without relying on increased clock frequencies. A clear example is the architectures of CPUs supporting single instruction multiple data (SIMD) instructions and GPUs, which are suitable for vector processing; in vector processing, a single instruction executes multiple operations, reducing the number of instructions issued and achieving high computing performance without increasing the clock frequency. Inversely, many conventional algorithms involving many sequential computations cannot exploit this architecture, resulting in less performance gains. Thus, in light of the current trend of the development of high-performance computing devices, a significant increase in the performance of the sparse linear solver by the simple approach described above cannot be expected. Therefore, it is necessary to realize new algorithms that can fully utilize current and future high-performance computing devices.

To this end, I consider two approaches based on the structural and numerical properties of algorithms of sparse linear solvers, respectively. The former focuses, in particular, on the structures to utilize vector processing; in addition to exploiting relatively small-size vector processing structures, such as those for CPU SIMD instructions, I also consider exploiting structures for more massive parallelism, such as those for GPUs. As mentioned above, vector processing is an appreciated approach for achieving high-performance computation at low clock frequencies. Hence, it will be a significant processing architecture for future devices, not only for existing CPUs and GPUs. Since many existing computing devices support vector processing and their architectures vary, to deeply study each method, I have specifically considered two types of computing devices, Intel CPUs and NVIDIA GPUs, and developed algorithms for each device. Note that although they

are evaluated only on the targeted devices, the methods and algorithms investigated in this work may also apply to other devices. The latter numerical approach considers the use of integer arithmetic. In particular, it examines methods to reduce the amount of data while maintaining the accuracy of the solution. This approach aims to increase the number of elements processed in a single vector operation and use memory bandwidth efficiently by reducing the amount of data transfer. Furthermore, it will be advantageous in environments with limited circuit resources, such as reconfigurable devices like field-programmable gate arrays (FPGAs).

In these approaches, this research mainly makes the following contributions:

- Chapter 3 shows that SIMD instructions can be effectively utilized in Krylov solvers by controlling fill-ins in ILU preconditioners; this can significantly reduce the number of iterations and the solution time.
- Chapter 4 demonstrates that matrix reordering based on multi-coloring increases the concurrency of the ILU(0) preconditioner significantly, leading to the efficient use of the massive parallelism of GPUs; furthermore, it shows that grouping several nodes in the coloring process can reduce the performance degradation in the convergence rate of the ILU(0) preconditioner.
- Chapter 4 also clarifies that approximate inverse preconditioning can be a strong alternative to ILU preconditioning with matrix reordering, significantly when solving problems with many nonzero elements on GPUs; along with this, this chapter presents a new algorithm to approximate the inverse of the coefficient matrix quickly.
- Chapter 5 shows that Krylov solvers implemented primarily with integer arithmetic can find solutions with a high accuracy comparable to double-precision floating-point solvers by using the iterative refinement technique; moreover, it also confirms that the algebraic multi-grid preconditioner and the approximate inverse preconditioner can also be implemented with integer arithmetic without degrading their performance.

The following describes the contents of the three chapters in more detail.

Chapter 3 considers using CPU SIMD instructions efficiently in Krylov solvers. To this end, the chapter first evaluates the performance of the sparse matrix-vector multiplication kernels implemented with the block compressed row storage (BCRS) format and SIMD instructions. Numerical results show that the unnecessary computations (e.g., multiplication by zero elements) introduced by the BCRS format to reduce random memory accesses hinder the full exploitation of the SIMD instructions. This result indicates that reducing such unnecessary computations can lead to better usage of SIMD instructions. Hence, the chapter focuses, in particular, on fill-ins in the ILU factorization to achieve this. Specifically, it presents a new technique called ILUB that gives the LU factors a block structure corresponding to the SIMD width by controlling fill-ins. ILUB can approach the exact LU factors while keeping the structure suited for SIMD instructions, reducing the meaningless computations due to the BCSR format. While the ILUB technique can apply to any ILU preconditioner if only it primarily uses a sparsity pattern to control fill-ins, the numerical tests choose the simplest one, ILU(0), which controls fill-ins so that the resulting sparsity pattern will be identical to the coefficient matrix. The numerical results show that ILUB is over seven times faster than the conventional ILU(0). The results indicate that significant improvements in convergence rate and suppression of the increase in computational time due to increased fill-ins by SIMD instructions result in higher performance of ILU preconditioning. Consequently, the chapter demonstrates that controlling the sparsity pattern is a rational way to utilize SIMD instructions in Krylov solvers.

Chapter 4 considers the effective use of GPUs in Krylov solvers. The above method is effective, especially considering relatively small-size vector processing such as SIMD, but includes some sequential computations. Although this would be fine in multithreading on CPUs, considering GPU implementations, the sequential nature can hinder the effective use of the massive parallelism of GPUs. Hence, increasing the concurrency of (the forward and backward subsection kernels of) ILU preconditioners will be one of the critical issues. For this reason, Chapter 4 investigates methods to increase the concurrency of preconditioners. First, Section 4.1 considers the ILU(0) preconditioner again and demonstrates that reordering the coefficient matrix via graph coloring, or multi-coloring (MC), can provide sufficient concurrency to utilize the GPU parallelism. However, it also clarifies that the reordering method degrades ILU(0) in improving the convergence of Krylov solvers because it removes indirect but significant (in some sense) data dependencies. Therefore, this section evaluates two other coloring strategies that can preserve such significant data dependencies without losing the high concurrency of the original method as far as possible; one is an existing method called hierarchical block multi-color (HBMC) ordering, designed to keep the performance of ILU(0) in convergence rate while increasing the concurrency of ILU(0); the other is a novel method called two-step multi-color ordering, expected to have the same superiority to MC as HBMC and to enhance the concurrency of ILU(0) over HBMC because of its aggressive strategy in the coloring step. The numerical tests with a number of problems show that ILU(0) parallelized via graph coloring can perform well even on GPUs without losing its ability to improve the convergence of Krylov solvers.

The approach via graph coloring is quite attractive, but one problem remains. That is, for problems with a relatively large number of nonzero elements, the approach tends to lead to insufficient concurrency of ILU(0). This is because many nonzero elements mean that many adjacencies exist, making it challenging to gather many non-adjacent nodes in graph coloring. Admittedly, there is a possibility that the heuristic coloring strategy used cannot find a (near-)optimal coloring pattern, resulting in many colors and poor concurrency. However, this could be a limitation of the approach based on reordering the coefficient matrix. Therefore, In Section 4.2, I consider another approach to enhance the concurrency of the preconditioner. Specifically, this section demonstrates that directly approximating the inverse of the LU factors (the inverse of the coefficient matrix) by using the AINV algorithm can contribute to better concurrency and performance, particularly when solving problems with many nonzero elements. As mentioned, the AINV preconditioner can be a great candidate for GPU acceleration of Krylov solvers, but it includes a disadvantage. It is the long execution time for approximating the inverse of the coefficient matrix. In some test cases, preparing the preconditioner with the AINV algorithm takes about five times longer than the solution process with a Krylov solver. Thus, I have proposed a new variant of AINV called the statically and dynamically approximate inverse (SD-AINV) method to shorten the total execution time. Numerical experiments show that SD-AINV can approximate the inverse of the coefficient matrix more than twice as fast as AINV without degrading the quality of the approximation; in other words, two solvers preconditioned by AINV and SD-AINV, respectively, have almost identical convergence rates.

Chapter 5 considers using short bit-length integer arithmetic in Krylov solvers. Integer arithmetic mainly has the following three advantages both on existing high-performance computing devices such as GPUs and future, developing ones including FPGAs:

- It is easy to share a single instruction with several arithmetics, effectively using computing units, especially when combined with short bit-length arithmetic.

- It has many bits corresponding to the significand part of floating-point formats because it does not need an exponential part, which may lead to better precision and faster convergence of Krylov solvers than the same bit-width floating-point if managed appropriately.
- Its circuit structure is more straightforward than that of floating-point arithmetic, which is advantageous for creating more computing units with the same amount of circuit resources.

To exploit these advantages of integer arithmetic, specifically, this chapter develops an FGMRES method that mainly uses 32-bit integers with the iterative refinement framework. All procedures of the inner FGMRES part, including preconditioning, can be implemented using only integer (fixed-point) operations, except for the computation of the correction vector. Due to the iterative refinement framework and using fixed-point formats, the solver can achieve the same level of accuracy as a full-precision counterpart. In detail, numerical tests show that it can find an approximate solution whose corresponding relative residual norm is less than 10^{-10}. In execution time, the integer-based solver outperforms the full-precision solver for most test cases. Furthermore, in several test cases, it is superior to a mixed-precision solver that mainly uses half-precision floating-point operations.

As stated above, this research has shown a direction to take advantage of current and future high-performance computing devices in sparse linear solvers by examining the algorithms with structures suitable for vector processing and the implementation framework that uses shorter bit-length (32-bit) integers. Thus, the series of results of this research will contribute to the continued performance improvement of sparse linear solvers. However, there remain several areas for improvement. For example, it would be necessary to implement the integer-based solver on other devices, such as GPUs, while utilizing SIMD instructions with shorter bit-length integer (e.g., 16-bit integer) arithmetic. To achieve this, in addition to just implementing the integer-based solver on such devices, it is essential to investigate ways to further reduce the bit-length of the integer solver. Moreover, it is also crucial to consider applying the technique to control fill-ins based on the block structure to the preconditioners designed for utilizing the massive parallelism of GPUs, such as SD-AINV. Additionally, when considering software and hardware co-design using devices like FPGAs, examining ways to achieve optimal resource allocation for implementing numerous computing units is important.