King Saud University

Journal of King Saud University – Computer and Information Sciences

www.ksu.edu.sa
www.sciencedirect.com

ORIGINAL ARTICLE

Performance modeling and analysis of parallel Gaussian elimination on multi-core computers

Fadi N. Sibai [*]

P&CSD Dept. Center, Saudi Aramco, Dhahran 31311, Saudi Arabia

Received 16 October 2012; revised 2 February 2013; accepted 12 March 2013
Available online 20 March 2013

KEYWORDS

Gaussian elimination;
Multi-core computing;
Performance modeling and analysis

Abstract Gaussian elimination is used in many applications and in particular in the solution of systems of linear equations. This paper presents mathematical performance models and analysis of four parallel Gaussian Elimination methods (precisely the Original method and the new Meet in the Middle –MiM– algorithms and their variants with SIMD vectorization) on multi-core systems. Analytical performance models of the four methods are formulated and presented followed by evaluations of these models with modern multi-core systems' operation latencies. Our results reveal that the four methods generally exhibit good performance scaling with increasing matrix size and number of cores. SIMD vectorization only makes a large difference in performance for low number of cores. For a large matrix size ($n \geqslant 16$ K), the performance difference between the MiM and Original methods falls from 16× with four cores to 4× with 16 K cores. The efficiencies of all four methods are low with 1 K cores or more stressing a major problem of multi-core systems where the network-on-chip and memory latencies are too high in relation to basic arithmetic operations. Thus Gaussian Elimination can greatly benefit from the resources of multi-core systems, but higher performance gains can be achieved if multi-core systems can be designed with lower memory operation, synchronization, and interconnect communication latencies, requirements of utmost importance and challenge in the exascale computing age.

1. Introduction

The large majority of modern microprocessors integrate multiple processing cores on the same package leading to great performance/power ratio gains. Today both central processing units (CPUs) and graphic processing units (GPUs) integrate multiple processing cores. Multi-core processors implement various types of parallelism including instruction-level parallelism, thread-level parallelism, and data-level parallelism. While the trend toward increasing number of cores per processor is strong, it is an interesting and important problem to identify the performance weaknesses of multi-core processors in order to keep the increasing number of cores per processor trend going and avoid the multicore performance "wall." While real application profiling and scalability measurement are accurate performance indicators, they are time consuming in particular when exploring application scalability over a very large number of cores. A more flexible and faster method is analytical performance modeling

[*] Tel.: +966 3 8808523; fax: +966 3 8758302.
E-mail address: fadi.sibai@aramco.com.

Peer review under responsibility of King Saud University.

which captures the essence of the performance-impacting components of the software application and underlying hardware models, and provides sufficiently accurate answers in particular with regard to trends and key performance impactors. In this paper, we model the performance of parallel Gaussian Elimination (GE) on modern multi-core processors, and derive speedups versus increasing number of cores and derive learnings for future multicore processor designs.

Gaussian elimination (Grama, 2003; Quinn, 1994) is an efficient mathematical technique for solving a system of linear equations. Given the matrix set $A \times X = B$, where X is the variable matrix and A and B are constant matrices, Gaussian elimination (GE) solves for the elements of the X matrix. Gaussian Elimination has also other uses for it is also used in computing the inverse of a matrix. The algorithm is composed of two steps. The first step combines rows eliminating a variable in the process and reducing the linear equation by one variable at a time. This reduction step is repeated until the left side ($A \times X$) is a triangular matrix. The second step is a back substitution of the solved X variables into upper rows until further X variables can be solved.

The algorithm generally works and is stable and can be made more stable by performing partial pivoting. The pivot is the leftmost non-zero matrix element in a matrix row available for reduction. When the pivot is zero (non-zero), exchanging the pivot row with another row usually with the largest absolute value in the pivot position may be required. When two rows are combined to reduce the equation by one variable, a multiplication of all elements in the pivot row by a constant and then subtracting all elements in the row by the ones in the other row are performed resulting in a new reduced row. During back substitution, a number of divisions, multiplications and subtractions are performed to eliminate solved variables and solve for a new one.

Partial pivoting works as follows. First the diagonal element in the pivot column with the largest absolute value is located and is referred to as the Pivot. In the Pivot row which is the row containing the pivot, every element is divided by the pivot to get a new pivot row with a 1 in the pivot position. The next step consists of replacing each 1 below the pivot by a 0. This is achieved by subtracting a multiple of the pivot row from each of the rows below it.

Because partial pivoting requires more steps and consumes more time and because without partial pivoting Gaussian Elimination is acceptably stable, we ignore partial pivoting herein and accept to trade off performance for stability.

The numerical instability is proportional to the size of the L and U matrices with known worst-case bounds. For the case without pivoting, it is not possible to provide a priori bounds for Yeung and Chan (1997) provide a probabilistic analysis of the case without pivoting.

Sankar (2004) proved that it is unlikely that A has a large growth factor under Gaussian elimination without pivoting. His results improve upon the average-case analysis of Gaussian elimination without pivoting presented by Yeung and Chan (1997).

Xiaoye and Demmel (1998) proposed a number of techniques in place of partial pivoting to stabilize sparse Gaussian elimination. They propose to not pivot dynamically thereby enabling static data structure optimization, graph manipulation and load balancing while remaining numerically stable. Stability is kept by a variety of techniques: pre-pivoting large elements to the diagonal, iterative refinement, using extra precision when needed, and allowing low rank modifications with corrections at the end.

Geraci (2008) also avoids partial pivoting and recently achieved a performance of 3.2Gflops with a matrix size of 33500 on the Cell Broadband Engine.

Parallel solutions of numerical linear algebra problems are discussed in Demmel (1993), Duff and Van der Vorst (1999). While the Gaussian Elimination method yields an exact solution to the $A \times X = B$ problem, iterative techniques trade off execution time for solution accuracy. Parallel iterative techniques are discussed in Barrett (1994), Greenbaum (1997), Saad (2003), Van der Vorst and Chan (1997). These techniques start with initial conditions for the X vector and keep refining the X solution with each iteration until the X vector converges to a solution vector. The different selections of the initial values of the X vector result in a variety of preconditioning techniques with various performance yields.

Further performance gains can be achieved when the matrices are sparse with many zero elements. Sparse matrix computation techniques are discussed in Heath (1997), Gupta (1997), Gupta et al. (1998), Demmel et al. (1999).

In this paper, we analyze the performance of Parallel Gaussian Elimination and an improved version on multi-core computers with increasing number of cores, and with and without SIMD vectorization. Section 2 reviews relevant literature. Section 3 presents the multi-core processor model on which the parallel GE algorithms are executed. Section 4 reviews the parallel GE algorithm and presents a new improved version: *Meet in the Middle* algorithm. Section 5 presents the mathematical performance model of the parallel GE algorithm. Section 6 presents the mathematical performance model of the improved parallel GE algorithm version. Section 7 states the assumptions made in this performance modeling study. Section 8 presents the parallel GE algorithms' performance results on the multicore processor model with 4–16 K cores, and their analysis. The paper concludes in Section 9.

2. Related background

Various performance analysis tools and methods are available (Prinslow, 2011). Profiling and scalability measurements of real application performance lead to accurate results. Application profiling can be achieved by tools such as Intel VTune, AMD Code Analyst, or GNU gprof. However this requires implementation of the application, and running it on various machines with various configurations, such as processors with various number of cores, a process which consumes long time and extensive resources. More importantly, this approach cannot be adopted in our study as we wish to analyze the application scalability up to 16 K cores, and current hardware limits our exploration to only about 10 cores. Benchmarks are also very efficient in studying computer performance. However they cannot be used in our study for the same reason as real applications: lacking scalable hardware. Simulators can provide up to cycle accurate performance but they require a very long time to develop and are a suitable approach when the simulator is intended to be used extensively. Analytic performance modeling provides a quick and fairly accurate method for identifying trends and the key performance impactors. It is flexible and allows for very large of cores to be modeled, more than currently available in the market. For these reasons we employ analytic performance modeling in this study.