Performance Evaluation of Jacobi Iterative Solution for Sparse Linear Equation System on Multicore and Manycore Architectures

Samiah Alzahrani[1], Mohammad Rafi Ikbal[2(✉)], Rashid Mehmood[3], Mahmoud Fayez[2], and Iyad Katib[1]

[1] Faulty of Information Technology, King Abdul Aziz University, Jeddah, Saudi Arabia
salzahrani0683@stu.kau.edu.sa, iakatib@kau.edu.sa
[2] Fujitsu Technology Solutions, Jeddah, Saudi Arabia
{mohammad.rafi, mahmoud.fayez}@ts.fujitsu.com
[3] High Performance Computing Center, King Abdulaziz University, Jeddah, Saudi Arabia
RMehmood@kau.edu.sa

Abstract. One of the common and pressing challenges in solving real-world problems in various domains, such as in smart cities, involves solving large sparse systems of linear equations. Jacobi iterative method is used to solve such systems in case if they are diagonally dominant. This research focuses on the parallel implementation of the Jacobi method to solve large systems of diagonally dominant linear equations on conventional CPUs and Intel Xeon Phi co-processor. The performance is reported on the two architectures with a comparison in terms of the execution times.

Keywords: Jacobi iterative method · Sparse linear equation systems
Intel MIC · Intel Xeon Phi

1 Introduction

Many applications in the field of Mathematics and Sciences involve solving linear equation systems. For instance, Markov modelling of smart city applications and systems give rise to very large linear equation systems, a problem referred to as the curse of dimensionality, see, for instance, [1–5]. There are many types of linear equation systems, which can be represented and solved using various methods. One category of such system of linear equations is Diagonally Dominant System of Linear Equations (DDSLE). DDSLEs are represented in the form Ax = B where A is a square matrix formed with coefficients in a system of linear equations. A system of equations is said to be diagonally dominant if n^{th} coefficient of nth equation is higher than sum of all other absolute coefficients irrespective of the sign it might have. For example, if we have n linear equations with n unknowns. Then first coefficient of first equation should be larger than the sum of other coefficients and the second coefficient of second equation should be larger than sum of other coefficients and so on. It should be noted

© ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2018
R. Mehmood et al. (Eds.): SCITA 2017, LNICST 224, pp. 296–305, 2018.
https://doi.org/10.1007/978-3-319-94180-6_28

that we only consider absolute value of the coefficient; the sign carried by the coefficient either positive or negative is ignored. Besides being diagonally dominant, these systems also tend to be sparse. Since they are sparse in nature, only very less proportionate of the values carry a value, rest of the values is zero.

In this paper, we have implemented Compressed Sparse Row (CSR) method for storing values of matrix during computation. This not only reduces the memory footprint of the application but improves performance of the application since iterations does not include zero-valued elements.

The objective of this research is to evaluate the performance of Intel many Integrated Core (MIC) Architecture, the Intel Xeon Phi co-processor, in solving large diagonally dominant system of linear equations. We are comparing the performance of the code on traditional Intel Xeon CPUs with Intel Xeon Phi co-processors.

2 Background

With the advent of technology in almost all domains, many applications depend on mathematical equations to solve variety of problems. These applications range from financial and trading software, navigation control systems, healthcare systems, astronomical systems, military applications, etc. For these to work efficiently and timely manner, solving mathematical equations in least possible time is crucial.

The Jacobi iterative method will be used to solve large diagonally dominant sparse matrices. Since these systems are sparse in nature, we have implemented compressed sparse row format to process the matrices. Since CSRs use comparatively smaller memory to store the sparse matrix as well as benefit performance of the application by eliminating iterations involved in processing zero-value based elements of the matrix. Jacobi's algorithm is implemented using OpenMP takes advantage of shared-memory systems to launch multiple threads, which run in parallel. Using OpenMP has another benefit; it runs on Intel MIC Architecture seamlessly. This enables us to execute the same code on Intel CPU and Intel Xeon Phi co-processor without any modification to the actual code except directives to run the code on Intel MIC architecture.

2.1 System of Linear Equations

System of linear equations is collection of equations with same solution or same set of values for its variables. We are using matrices to represent these equations and these matrices are diagonally dominant.

Following equation denotes a system of n linear equations of the form Ax = d

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad d = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix} \tag{1}$$

Matrix A is diagonally dominant if it satisfies following equation. The location of the element in the matrix is represented as values of i and j.

$$|a_{ii}| \geq \sum_{i \neq j} |a_{ij}| \tag{2}$$

2.2 Sparse Matrix

Sparse matrices have a higher proportion of zero-valued elements compared to non-zero elements. Since compute applications allocate same amount of memory for all the elements in an array irrespective of the value, this increases memory footprint of the application [6, 7]. There are many methods to overcome this problem and use the system memory efficiently, one such format is coordinate format (COO) [8]. But COO is not efficient in practice. We have implemented Compressed Sparse Row (CSR) format to store and retrieve matrices efficiently. This format uses one array to store values of the matrix whose value is non-zero and two arrays as look up table using, which we can retrieve, position of any element in the original sparse matrix. Several other formats to store sparse matrices exist, see [9, 10], and the references therein.

Since we are using CSR format to store the values of the matrix, we have changed the iteration method to process only non-zero values of the matrix. This reduces execution time of the application as the system does not process zero-valued elements, which resulted in reduction in complexity of the algorithm to Eq. (4).

Complexity of traditional algorithm is as follows, where n is dimension of square matrix

$$O = (n^2)$$

Complexity of algorithm using CSR format. M is number of non-zero elements in a matrix where $M \ll n^2$ since the matrices are sparse

$$O = (M)$$

2.3 Jacobi Method and JOR Iterative Methods

Jacobi method is one of the prominently used methods to solve diagonally dominant linear equation. This is one of stationary iterative method to solve set of linear equations. This method is of the form $x^{(k)} = Fx^{(k-1)} + c$, in this equation both F and c do not depend on k where $x^{(k)}$ is an approximate value tending towards the solution of the linear equations. These equations are in the form $Ax = b$ where A is a square matrix and $x, b \in R^n$. Computations performed by Jacobi's equation can be described in the following equation where M is the iteration count.

$$x_i^{(M)} = a_{ii}^{-1}(d_i - \sum_{i \neq j} a_{ij} x_j^{(m-1)}) \tag{3}$$

In the above equation, I and j represents row and column index of an element in the matrix and a denotes the element itself. $x_i^{(M)}$ and $x_j^{(M-1)}$ represents the ith coefficient of iteration numbered M and M − 1 respectively. Following is matrix notation of the above given equation.

$$x^{(M)} = \frac{1}{D}\left[(U + L)x^{(M-1)} + b\right] \tag{4}$$

Where D, L and U are partitions of square matrix A into its diagonal, upper triangular part and lower triangular part respectively.

There are many possible cases where Jacobi method does not converge, in such cases an under-relaxation parameter is introduced. This parameter can also be used as a catalyst to improve the convergence rate of Jacobi method. Using under-relaxation in Jacobi method is known as Jacobi over relaxation (JOR). Which can be represented in Eq. (5)

$$x_i^{(M)} = \alpha \widehat{x}_i^{(M)} + (1 - \alpha) x_i^{(M-1)} \tag{5}$$

In the above equation $\hat{x}$ denotes one Jacobi iteration as show in Eq. (3) and $0 < \alpha < 2$ is the relaxation parameter. i varies from 0 to n where n is number of rows in square matrix A. if $\alpha < 1$ then the method is known as under-relaxed method and if α is >1 then it is over-relaxed method. If $\alpha = 1$ then it would be called Jacobi method without relaxation. Both Jacobi and JOR equations exhibit very slow convergence rate as discussed in [10]. In Jacobi and JOR methods old approximation of the vector is a dependency in calculating new approximation, hence these iterative methods exhibit embarrassingly parallel behavior.

2.4 Intel Xeon Phi Coprocessor (Intel MIC)

Many Integrated Core (MIC) architecture is PCIe-based coprocessors which is added to traditional hardware to enhance the performance of the system. These coprocessors add an additional 60+ usable cores with each core supporting 4-way simultaneous multi-threading to existing system. The MIC coprocessor are based on x86 ISA with 64-bit and 512-bit wide SIMD vector instructions and registers. These cores have similar architecture as traditional Intel Xeon processor which enables standard programming language techniques such as OpenMP to be used on the accelerator card.

2.5 Programming Model

Though Intel Xeon Phi has similar architecture as traditional Intel Xeon processors, it's programming model is little different since it is connected to the host system through the PCIe bus. These devices are designed such that applications can leverage full

potential of vector processors, memory bandwidth and cache. Through existing parallel programming APIs like OpenMP and MPI are supported in Intel MIC architecture, application developer should restructure the code such that it run on Intel MIC efficiently. Intel MIC supports three programming modes.

Native Mode. To enable this mode, dedicated network must be configured on Intel Xeon Phi coprocessor with dedicated IP address and host name. Once network is configured, users can login or execute code directly on Intel MIC like any other Unix system since Intel Xeon Phi runs embedded Linux operating system. Since this environment is like any other Unix environment, users can execute their application directly by copying it to coprocessor or from a shared storage.

Offload Execution Mode. In this mode, the program initially starts executing on the host processor and then offloads the tasks to the MIC to execute a block of instructions. The process running on the host controls the exchange of data between host and device. While MIC is executing part of the program, host may or may not participate to work in parallel. Figure 1 show an overview of offload mode.

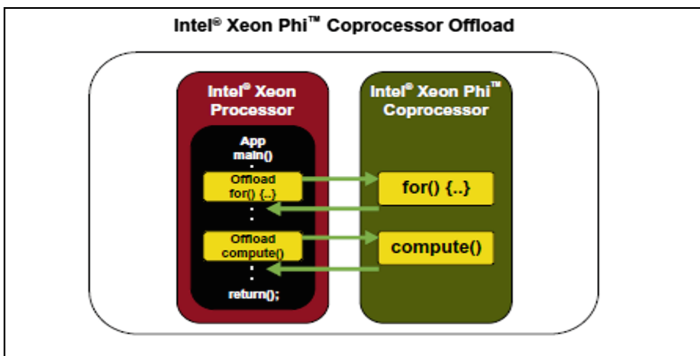

Fig. 1. Offload execution mode of Intel Xeon Phi

Symmetric Execution Mode. In this mode, the application processes execute on host and device. This may involve multiple hosts and device combinations. In modern application, the communication between processes takes place thorough Message Passing Interface (MPI). This mode is combination of traditional MPI programming model and Native mode execution of Intel MIC coprocessors since this model treats MIC as a node in a heterogeneous cluster.

3 Literature Review

Sparse matrix-vector multiplication (SpMV) is used in wide range of applications like signal processing, engineering, electronic circuit design, military, etc. Optimizing performance of applications solving sparse matrix-vector multiplication was topic of interest to various research communities and individuals.

In [11], they have implemented an extensive evaluation of SpMV kernel for both serial and parallel versions on diverse modern architectures. They have proposed number of optimization guidelines based on their study.

In [12], the authors have investigated optimization techniques to improve memory efficiency in SpMVs. Based on their study, they suggested to use register level optimization techniques if the size of the matrix is small. For larger matrices, which do not fit into registers available in the system, cache level optimization is recommended.

Vuduc [13] shows that the major challenge in developing highly optimized implementations of SpMV is selecting storage models and algorithm that take advantage of the properties of the matrix is unknown until execution. As per their observations, conventional implementations of SpMV can use only 10% or less CPU peak machine performance on cache-based superscalar architectures. While their implementations of SpMV, tuned using a methodology based on the empirical search, can, by contrast, reach up to 31% of peak machine speed and can be up to 4 times faster.

Design and implementation of SpMV on several important chip multiprocessor systems (CMP) is discussed in [14]. Since there are many available designs of CMP, selecting a design best suitable for solving SpMV along with selecting algorithms based on SpMV type was the challenge. Substantially, their results show that matrix and platform reliant on the tuning of SpMV for multicore.

A new strategy for improving the performance of SpMV is suggested in [15]. This solution involves using a loop transformation know as unroll-and-jam. Using this method improves performance by 11% which is a factor of 2.3. however; this approach is suitable only for sparse matrices that have dimensions of small number of predictable lengths.

One of the widely method used for SpMV application optimization is Blocking. There are two methods of Blocking, the first method exploits memory allocation at several levels like register [15, 16], L1 and L2 cache [12, 17] and storage buffers [17]. In second method indexing of elements is optimized to remove overhead which enhances memory bandwidth utilization [16, 18].

Intel Math Kernel Library (MKL) was used to optimize implementations of SpMV in paper [19]. As per their experiments, they could achieve 80% performance improvement with dense matrices. However; performance of sparse matrices was variable and the results out performed CPU in most cases.

In [20] they describe the factors that limit the SpMV performance Intel MIC architecture. The three factors are: low SIMD efficiency because of sparsity, the overhead caused by unstructured memory access and ununiform load balance caused due to uneven matrix dimensions. To overcome these problems, a new matrix format with name ELLPACK Sparse Block (ESB) was designed. This new matrix format was implemented on Knights Corner (KNC) which is latest generation of Intel Xeon Phi

coprocessor, which has wider SIMD and other advanced features. To solve load imbalance problem, the authors proposed three load balancers.

The [21] improve the ELLPACK format to produce sliced ELLPACK with the use of SIMD vectorization on General Purpose Graphics Processing Units (GPGPUs). The SELL-C-σ (sliced ELLPACK) show its appropriateness on different hardware platforms. The authors aimed to find a format to process unified sparse matrix data efficiently on traditional CPUs. There are lot of improvement areas to achieve higher performance, these methods need to be explored.

Implementation of Compressed Sparse Row 5 (CSR5) to solve sparse matrices is discussed in [6]. The results of the experiments conclude that the performance improvement is average.

Fourth-order Runge-kitta method was implemented to solve sparse matrices in [22]. They used CSR format to store matrices and Intel MKL routes to solve the equations. They were able to achieve good performance improvement on CPU and Intel MIC

The work in [22, 23] aims to shorten the computation time for the transition matrices that arising from Markovian models of complicated systems on Intel Xeon Phi. They used CSR format as used in work [24], and HYB format is similar as in work [20]. Their method takes advantage of the thread-level parallelism and vectorization for the SpMV implementation. The numerical experiments result of CTMCs executed on Intel Xeon Phi coprocessors using offload mode show that HYB format delivers higher performance rate than data stored in CSR format.

4 Research Design and Methodology

First step in the experiment was to parallelize Jacobi algorithm. We have implemented serial version of the algorithm and tested the functionality, then we have parallelized the row operations such that each row operation will be processed by a separate thread on a processor such that n rowed matrix will spawn n processes. Once we have made sure that the results of serial version and parallel version is consistent, we have implemented Compressed Row Format (CSR) to store and retrieve sparse matrices and eliminated iterations which involve zero-valued elements of the matrix. These modifications add two advantages to the program. First advantage is that there is reduction in memory foot print of the application and performance increase in memory access as mentioned in [20]. The number of thread spawned is now decreased to M where M is number of elements in a sparse matrix whose value is not zero and $M < N^2$ where N is dimension of the matrix since the matrix is sparse in nature.

We have conducted our experiment on E5-2695v2 (Ivy Bridge) dual socket server with 12 cores on each socket with 2.4 GHz frequency. Each server has 96 GB of physical memory. The coprocessor used is Intel Xeon Phi 5110P. This coprocessor has 60 usable CPU cores operating at 1.05 GHz frequency and 8 GB memory.

We have executed multiple iterations of the program using OpenMP on traditional CPUs and using Offload model on Intel MIC and the results are plotted for comparison.

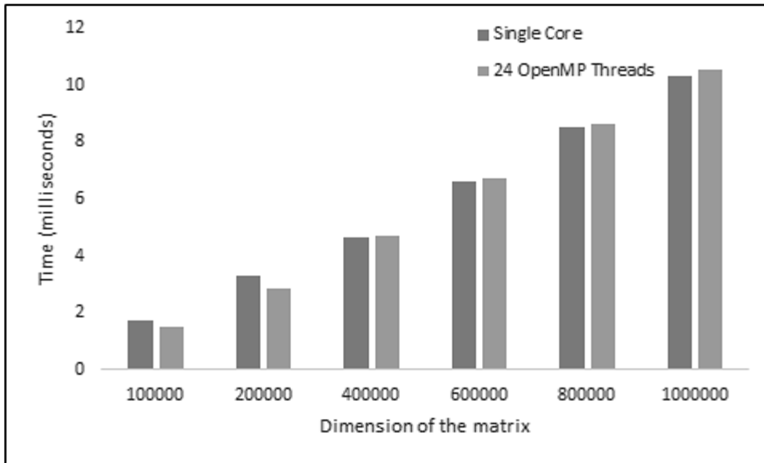

Fig. 2. Performance of single CPU *vs* 24 OpenMP threads

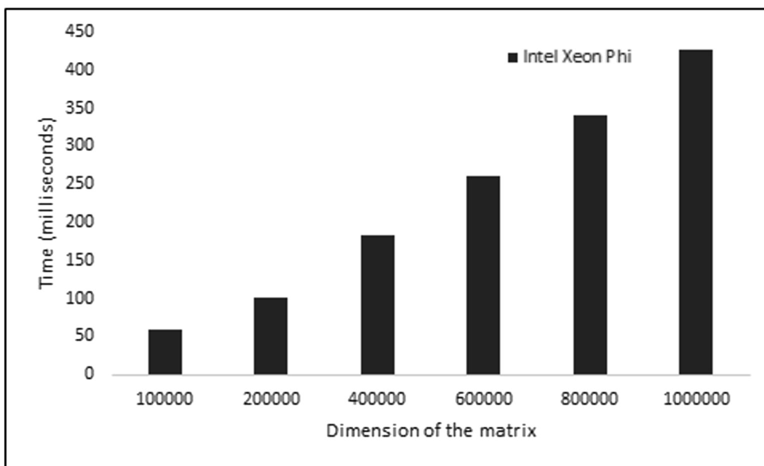

Fig. 3. Performance on Intel Xeon Phi coprocessor (Intel MIC)

5 Results

We have executed multiple iterations of the application with different matrix starting from 100000 to 1000000 and the results are plotted in Figs. 2 and 3. As seen in the Figs. 2 and 3, performance of the application on 24 CPUs using OpenMP does not show considerable performance over performance on single CPU. We have plotted a separate graph for the performance of Jacobi method on Intel Xeon Phi coprocessor since the performance of the application on Intel MIC architecture is very poor on

comparison with performance on CPU. We believe that this is because of initialization and communication overhead compared to small workload.

6 Conclusions and Future Work

Based on the results of our experiments, we believe that Intel MIC architecture is not suitable platform for problem like Jacobi method to solve diagonally dominant sparse matrices. This may be because of the sparseness of the system which reduces the computations significantly. Since the workload is very small, communication and initialization overhead overshadows the performance of the application delivering a poor performance overall.

Our future work is to implement Jacobi method on dense matrices and evaluate the performance on heterogeneous architectures. We will also investigate further our implementation of the Jacobi method to identify the performance bottlenecks and improve the speedup.

Acknowledgements. The experiments reported in this paper were performed on the Aziz supercomputer at King Abdul Aziz University, Jeddah, Saudi Arabia.

References

1. Altowaijri, S., Mehmood, R., Williams, J.: A quantitative model of grid systems performance in healthcare organisations. In: 2010 International Conference on Intelligent Systems, Modelling and Simulation, pp. 431–436. IEEE (2010)
2. Mehmood, R., Lu, J.A.: Computational Markovian analysis of large systems. J. Manuf. Technol. Manag. **22**, 804–817 (2011)
3. Mehmood, R., Alturki, R., Zeadally, S.: Multimedia applications over metropolitan area networks (MANs). J. Netw. Comput. Appl. **34**, 1518–1529 (2011)
4. Mehmood, R., Meriton, R., Graham, G., Hennelly, P., Kumar, M.: Exploring the influence of big data on city transport operations: a Markovian approach. Int. J. Oper. Prod. Manag. **37**, 75–104 (2017)
5. Mehmood, R., Graham, G.: Big data logistics: a health-care transport capacity sharing model. Procedia Comput. Sci. **64**, 1107–1114 (2015)
6. Liu, W., Vinter, B.: CSR5: An Efficient Storage Format for Cross-Platform Sparse Matrix-Vector Multiplication (2015)
7. Pissanetzky, S.: Sparse Matrix Technology electronic edition
8. Saad, Y.: Iterative Methods for Sparse Linear Systems. Society for Industrial and Applied Mathematics, Philadelphia (2003)
9. Mehmood, R.: Serial disk-based analysis of large stochastic models. In: Baier, C., Haverkort, B.R., Hermanns, H., Katoen, J.-P., Siegle, M. (eds.) Validation of Stochastic Systems. LNCS, vol. 2925, pp. 230–255. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24611-4_7
10. Mehmood, R., Crowcroft, J.: Parallel iterative solution method for large sparse linear equation systems. Technical report Number UCAM-CL-TR-650, Computer Laboratory, University of Cambridge, Cambridge, UK (2005)

11. Goumas, G., Kourtis, K., Anastopoulos, N., Karakasis, V., Koziris, N.: Performance evaluation of the sparse matrix-vector multiplication on modern architectures. J. Supercomput. **50**, 36–77 (2009)
12. Im, E.-J., Yelick, K., Vuduc, R.: Sparsity: optimization framework for sparse matrix kernels. Int. J. High Perform. Comput. Appl. **18**, 135–158 (2004)
13. Vuduc, R.W.: Automatic performance tuning of sparse matrix kernels (2003). https://dl.acm.org/citation.cfm?id=1023242
14. Williams, S., Oliker, L., Vuduc, R., Shalf, J., Yelick, K., Demmel, J.: Optimization of sparse matrix-vector multiplication on emerging multicore platforms. Parallel Comput. **35**, 178–194 (2009)
15. Mellor-Crummey, J., Garvin, J.: Optimizing sparse matrix-vector product computations using unroll and jam. Int. J. High Perform. Comput. Appl. **18**, 225–236 (2004)
16. Pinar, A., Heath, M.T.: Improving performance of sparse matrix-vector multiplication. In: Proceedings of the 1999 ACM/IEEE conference on Supercomputing (CDROM) - Supercomputing 1999, p. 30–es. ACM Press, New York (1999)
17. Nishtala, R., Vuduc, R., Demmel, J.W., Yelick, K.A.: When cache blocking of sparse matrix vector multiply works and why. Appl. Algebra Eng. Commun. Comput. **18**, 297–311 (2007)
18. Vuduc, R.W., Moon, H.-J.: Fast sparse matrix-vector multiplication by exploiting variable block structure. In: Yang, L.T., Rana, O.F., Di Martino, B., Dongarra, J. (eds.) HPCC 2005. LNCS, vol. 3726, pp. 807–816. Springer, Heidelberg (2005). https://doi.org/10.1007/11557654_91
19. Deshmukh, O., Negrut, D.: Characterization of Intel Xeon Phi for linear algebra workloads (2014)
20. Liu, X., Smelyanskiy, M., Chow, E., Dubey, P.: Efficient sparse matrix-vector multiplication on x86-based many-core processors. In: Proceedings of the 27th international ACM conference on International conference on supercomputing - ICS 2013, p. 273. ACM Press, New York (2013)
21. Kreutzer, M., Hager, G., Wellein, G., Fehske, H., Bishop, A.R.: A unified sparse matrix data format for efficient general sparse matrix-vector multiplication on modern processors with wide simd units. SIAM J. Sci. Comput. **36**, C401–C423 (2014)
22. Bylina, B., Potiopa, J.: Explicit fourth-order Runge-Kutta method on Intel Xeon Phi coprocessor. Int. J. Parallel Program. **45**, 1073–1090 (2017)
23. Bylina, B., Potiopa, J.: Data structures for Markov chain transition matrices on Intel Xeon Phi. In: 2016 Federated Conference on Computer Science and Information Systems (2016)
24. Saule, E., Kaya, K., Çatalyürek, Ü.V.: Performance evaluation of sparse matrix multiplication kernels on Intel Xeon Phi. Presented at the 2014 (2014)