DOI: 10.14529/jsfi180311

Developing Efficient Implementations of Bellman–Ford and Forward-Backward Graph Algorithms for NEC SX-ACE

Ilya V. Afanasyev[1], *Alexander S. Antonov*[1], *Dmitry A. Nikitenko*[1], *Vadim V. Voevodin*[1], *Vladimir V. Voevodin*[1], *Kazuhiko Komatsu*[2], *Osamu Watanabe*[2], *Akihiro Musa*[2], *Hiroaki Kobayashi*[2]

The main goal of this work is to demonstrate that the development of data-intensive applications for vector systems is not only important and interesting, but is also very possible. In this paper we describe possible implementations of two fundamental graph-processing algorithms for an NEC SX-ACE vector computer: the Bellman–Ford algorithm for single source shortest paths computation and the Forward-Backward algorithm for strongly connected components detection. The proposed implementations have been developed and optimised in accordance with features and properties of the target architecture, which allowed them to achieve performance comparable to other traditional platforms, such as Intel Skylake, Intel Knight Landing or IBM Power processors.

Keywords: graph algorithms, NEC SX-ACE, vector computing, data-intensive applications.

Introduction

With a modern variety of hardware and software solutions in supercomputing, it is very important to study fundamental properties of algorithms in order to understand which architectures are more suitable for various groups of algorithms. One very important and challenging example is a group of data-intensive algorithms: those algorithms usually stress target platform memory subsystem, as a result demonstrating low performance, high latency and poor data-cache usage due to low data locality. Graph algorithms represent data-intensive applications extremely well since they tend to have an enormous amount of random memory accesses, paired with low computational complexity. Moreover, graph processing is extremely relevant nowadays since it is used in various important real-world applications, such as web-graphs and social-networks processing. On the hardware side, vectorisation is an important feature of modern processors. NEC company has a long history and a lot of experience in developing vector systems with its computers having a large amount of unique features, such as extremely long vector length. Moreover, NEC computers are equipped with high-performance nodes and high-bandwidth memory, what makes them a good candidate for development of data-intensive applications. Since the area of data-intensive application development (and particularly graph algorithms) for vector systems is currently not studied enough, this paper describes implementation details of two fundamental graph processing problems for a NEC SX-ACE computer: single source shortest paths computation and strongly connected components search.

1. Target Platform and State of the Art

In this work we investigate implementations for a NEC SX-ACE computer. A single SX-ACE node includes 4 vector cores, each one with 1 GHz clock frequency. A single vector processing unit (VPU) of each core is capable of processing vectors of 256 length. The peak vector performance

[1]Lomonosov Moscow State University, Moscow, Russian Federation
[2]Tohoku University, Tohoku, Japan

of a single NEC SX-ACE socket is 256 Gflop/s, while peak scalar performance is only 4 Gflop/s per socket. Each vector core also supports out of order execution for vector load and store operations, with advanced data forwarding in vector pipes chaining. Each node is equipped with 256 GB/s high-bandwidth memory. Therefore, the bytes per flop ratio (the ratio of the peak performance to the memory bandwidth) is equal to 1, which is somewhat higher compared to other scalar and accelerator computers.

The approaches described in this paper are suitable only for shared-memory architectures (e.g. servers or individual supercomputer nodes). Graph algorithms implementations for shared-memory architectures are generally much more efficient compared to implementations for distributed-memory systems, since graph-processing problems tend to have an enormous amount of inter-node communications, what can easily become a bottleneck for overall performance. Moreover, modern studies demonstrate that platforms with shared-memory architecture are widely used to process various real-world graphs (social networks, road graphs and many others). There are not many currently existing studies and researches, which cover the development of graph algorithms or other data-intensive applications on NEC systems. However, there are several existing approaches describing the development of graph algorithms for other systems focused on vectorisation, such as Intel Knight Landing processors [1, 3].

2. Algorithms

In this paper two fundamental graph processing problems are investigated: single source shortest paths (SSSP) and strongly connected components (SCC). The SSSP problem implies finding the shortest paths in an undirected weighted graph from the selected source vertex to other vertices. The SCC problem implies partitioning a directed unweighted graph into disjoint sets of vertices, each one representing a strongly connected component where each vertex is reachable from another. In order to solve the shortest paths problem, the Bellman–Ford algorithm is used, while the Forward-Backward algorithm with a trim step is used for solving the strongly connected components problem. Those algorithms, together with possible approaches for their parallel implementation are described in detail in [2, 4]. From the computational point of view the Bellman–Ford algorithm requires both floating point and integer arithmetics together with frequent indirect memory accesses. In the meantime, the Forward-backward algorithm requires only integer arithmetics with frequent indirected memory accesses, and has a more complex nested parallelism potential. Furthermore, the Bellman–Ford algorithm has a slightly higher computational complexity than the Forward-Backward algorithm, since it operates with undirected graphs. Another important property of both algorithms is that they can be implemented without atomic operations usage, which tend to strongly impede vectorisation.

3. Implementation Details

An important point in any graph algorithm development is the selection of a data structure. This point is even more important for vector systems, since the selected data structures have to support efficient (bank-conflict free) memory accesses. In order to achieve efficient vectorisation, input graphs are represented in the format of a list of edges. This format allows a very simple and very efficient vectorisation of the loops, where the whole graph is traversed. Moreover, if the list of edges format is used, the edges can be stored in any order; some particular order

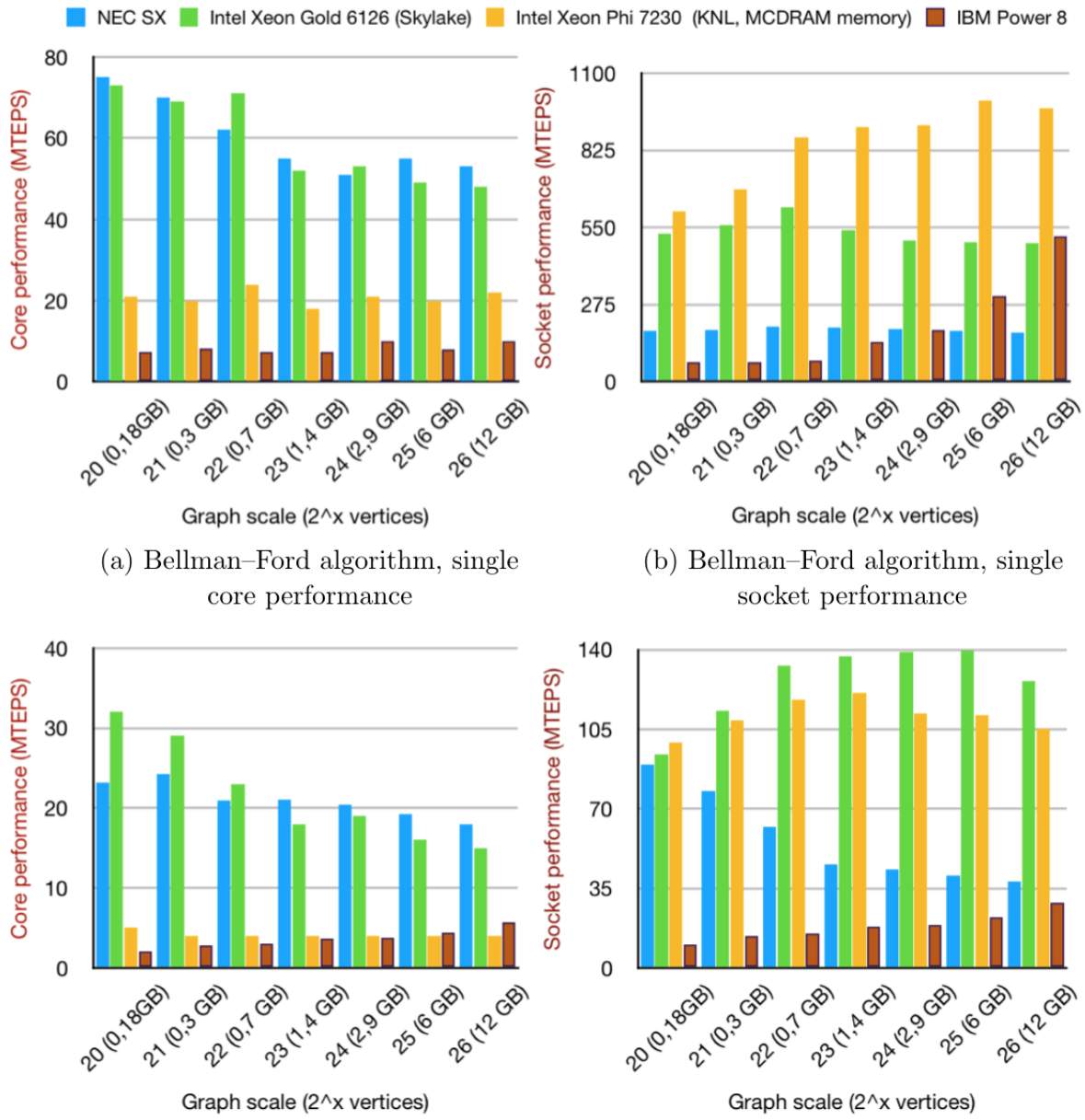
can greatly help to improve data locality and avoid bank conflicts, which results in significantly better performance.

In order to improve data locality, the following sorting strategy is used: input graph edges are sorted in a way that edges, stored in adjacent memory cells, point to adjacent cells in reachability (or distances) arrays. This reordering results into gather and scatter vector operations being much more efficient for adjacent edges (since information is gathered from adjacent cells of memory in distances or reachability array). It is also possible to remove loops and multiple arcs during this pre-processing stage. The proposed optimisation of storage formats provides almost up to a 20x performance improvement compared to the implementation with randomly sorted edges of the input graph.

To achieve efficient parallelisation and vectorisation inside a single NEC SX-ACE socket, several OpenMP and compiler-specific directives are inserted. The *#pragma parallel for* OpenMP directive is used with the *schedule(guided,1024)* OpenMP clause, in order to reduce overheads of synchronisations between threads; *#pragma cdir nodep* directive is used to provide the information to the compiler that there is no dependency (caused by indirect memory accesses) among loop iterations, which can prevent vectorisation. Moreover, *#pragma cdir vprefetch* directive, *#pragma cdir vovertake* and *#pragma cdir vob* directives are used to effectively issue vector gather and scatter instructions, which can easily become bottlenecks, as early as possible by overtaking other vector instructions. To effectively apply the overtakes of the vector gather and scatter instructions, loop unrolling is applied manually. Both implemented algorithms demonstrate high (above 99%) vector operation ratio (percentage of vector operations with maximum length inside the generated code) with an average vector length of 256, which means that the whole code for both algorithms was successfully vectorised.

4. Performance Evaluation

In this paper we compare the developed programs for the NEC SX-ACE computer with implementations for other architectures - Intel Knight Landing, Intel Skylake and IBM Power8. All those architectures support vector data-processing: modern Intel architectures include AVX-512 support, while IBM architecture has VMX instructions support. All implementations for other architectures are also highly optimised, with data locality improvements and vectorisation support. In order to compare various graph algorithms on different platforms, Traversed Edges Per Second (TEPS) metrics is used, which is equal to the number of edges in the input graph, divided on the algorithm execution time. As a result, the performance defined in TEPS is independent from the size and internal structure of an input graph. Figure 1 represent the sustained performance (in TEPS) of both implementations for a single socket and a single core of NEC SX-ACE system, compared to other platforms with vectorisation support: Intel Xeon Gold 6126 (Skylake family) processor, Intel Xeon Phi 7230 (Knight Landing family) and IBM Power 8 processors (RMAT graphs are used). The provided comparison demonstrates that NEC SX-ACE architecture allows to achieve the highest per-core performance among other architectures, but unfortunately lacks per-socket performance, possibly only due to insufficient number of cores. Moreover, the developed implementations demonstrate high parallel efficiency and vector operation ratio, which proves that at least these two graph algorithms can be efficiently vectorised.



(a) Bellman–Ford algorithm, single core performance

(b) Bellman–Ford algorithm, single socket performance

(c) Forward–Backward algorithm, single core performance

(d) Forward–Backward algorithm, single socket performance

Figure 1. Performance evaluation: Bellman–Ford algorithm, single core (a) and single socket (b); Forward-Backward algorithm, single core (c) and single socket (d)

Conclusion

In this paper efficient implementations of two important graph-processing algorithms for NEC SX-ACE platform have been described. The performance values have been compared to the performance of other implementations, obtained on various Intel and IBM platforms capable of vector data processing. The provided comparison demonstrates that NEC SX-ACE architecture allows to achieve the highest per-core performance among other architectures, but unfortunately lacks per-socket performance.

Acknowledgments

The research is carried out using the equipment of the shared research facilities of HPC computing resources at Lomonosov Moscow State University supported by the project RFMEFI62117X0011 and the computational resources of Cyberscience Center at Tohoku University supported by Joint Usage/Research Center for Interdisciplinary Large-scale Information Infrastructures in Japan (Project ID: jh170049-ISJ).

References

1. Besta, M., Marending, F., Solomonik, E., Hoefler, T.: SlimSell: A vectorizable graph representation for breadth-first search. In: 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS). IEEE (may 2017), DOI: 10.1109/ipdps.2017.93

2. Fleischer, L.K., Hendrickson, B., Pınar, A.: On identifying strongly connected components in parallel. In: Lecture Notes in Computer Science, pp. 505–511. Springer Berlin Heidelberg (2000), DOI: 10.1007/3-540-45591-4_68

3. Jiang, L., Chen, L., Qiu, J.: Performance characterization of multi-threaded graph processing applications on many-integrated-core architecture. In: 2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). IEEE (apr 2018), DOI: 10.1109/ispass.2018.00033

4. Nepomniaschaya, A.S.: An associative version of the bellman-ford algorithm for finding the shortest paths in directed graphs. In: Lecture Notes in Computer Science, pp. 285–292. Springer Berlin Heidelberg (2001), DOI: 10.1007/3-540-44743-1_28