IJCSE International Journal of Computer Sciences and Engineering Open Access
Survey Paper Volume-4, Issue-6 E-ISSN: 2347-2693

Vectorization on Intel Xeon Phi : A Survey

Akhilesh Thool[1*] and Hemlata Channe[2]

[1]*Post Graduate Department of Computer Science and Engineering (PICT), Pune University, India*
[2]*Assistant Professor, Pune Institute of Cosmputer Technology (PICT), Pune University, India*

www.ijcseonline.org

Received: May/16/2016 Revised: May/30/2016 Accepted: Jun/18/2016 Published: Jun/30/ 2016

***Abstract*—** Intel Xeon Phi coprocessors are the new family members of processors and platforms to the Intel family. Intel Xeon Phi is the first in the family of Intel MIC (Many Integrated Core) architecture. Software running on the coprocessor should leverage innumerable cores as well as make use of wide SIMD operation. Vectorization is the process of converting an algorithm from scalar implementation to vector. It is the form of parallel programming where the processors perform same operation simultaneously on N data elements of vector i.e. one dimensional array of scalar data objects such as floating point object , integers or double integers floating point. When the hardware is coupled with C/C++ compiler that supports it, developers have easier time delivering more efficient and better performing software.

***Keywords*—**Vectorization, Parallelism, High Performance Computing.

I. INTRODUCTION

Intel Xeon Phi is designed to extend the reach of application that has figured the ability to fully utilize the scaling capabilities of this co-processor based system. The applications also make use of available processor vector capabilities or memory bandwidth. For such application the Intel Xeon co-processor offers additional power efficient scaling, vector support and local bandwidth. Therefore the programmability is maintained by it and support associated with Intel Xeon Processors [7]. The first Intel Xeon Phi coprocessor has innumerable score of cores with wide vector or SIMD registers. Software running on this processor should enable to utilize these cores as well as take advantage of the wide SIMD operations i.e. Vector operations. Intel Xeon Phi provides high computational numerical performance. Thus to achieve this performance software must be properly tuned. The software must be vectorized, scalable and make efficient use of memory. These are the three factors i.e. scalability, vectorization and memory utilization that most influence performance on Intel Xeon Phi co-processor.

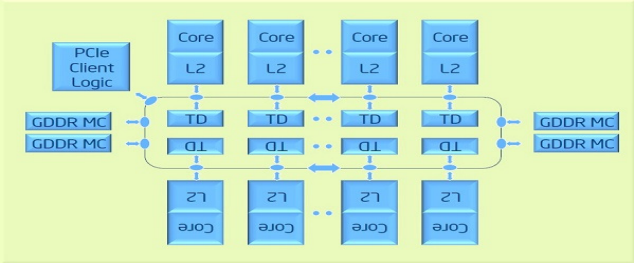

Fig.1. Intel Xeon Phi microarchitecture (Knights Corner)

The following list illustrates the features of first Intel Xeon Phi family product. It is named as “Knight’s Corner”.

- 50 above cores which run the Intel instructions set architecture (called as x86 Intel Architecture Instruction set).
- 4 threads per physical core.
- 512 bit register for SIMD operation (vector operation.
- 512K L2 cache per core.
- High speed bidirectional ring connecting the 50+ cores.

These cores are simpler than Intel Xeon since they have dual in-order execution pipeline as contradistinction to the out-of-order execution model on Intel Xeon processors. The computational power comes from 512 bit registers. The threads help to mask the effect of latencies on in order instruction execution. The optimum performance will only be achieved when the number of cores, threads or SIMD operations is used efficiently.

II. VECTORIZATION

Vectorization is the form of parallel programming where the processors perform same operation simultaneously on N data elements of vector. Vectorization can give as much as double precision (8x) or single precision float (16x) speedup on the Intel Xeon Phi coprocessors. The application may not reach this potential speedups, if the code is not vectorized.

A. *ESSENTIALS*

The very first step to work with vectorization is to understand the vectorization i.e. to understand what is vectorization, how it is used, weather compiler is able to vectorize the code or not.

It is also important which section of the code the compiler cannot vectorize and why it cannot. The developers should know the compiler pragmas and direction to assist the compiler to achieve vectorization.

III. VECTORIZATION USING SIMD REGISTERS

A. *ARRAY NOTATION*

One of the keys to the performance value of Intel Xeon Phi coprocessors is the 512-bit registers and associated SIMD operations. One of the first things to do to tune existing code is to understand where the time is spent. The sections of code which consume the most compute cycles are the hotspots and are the places to focus tuning effort first. Thus to make sure the optimization is working well in the hotspot region following steps should be followed.

Write using an array notation style such as available in Intel Cilka Plus or Fortran 90. When array notation is used, the compiler will vectorize or utilize the SIMD instruction set. The array notational syntax is the preferred method for ensuring the compiler will effectively utilize the SIMD operation.

B. *DIRECTIVES AND PRAGMAS*

Most software does not need to be rewritten in array notation to benefit from vectorization. The Intel compiler automatically vectorizes many for loops and constructs. However, it is typically insufficient or inadvisable to rely solely on auto-vectorization. Developers should be prepared to assist the compiler generate efficient vector code. In these cases the addition of pragmas or directives can provide the compiler with sufficient information to vectorize the code.

Efficient vectorization is important, too. Giving the compiler more information can help it generate far better vectorized code. So after verifying that the compiler reports a section is vectorized, check that it is doing so efficiently. Check that the code is not using split loads or stores and that the code avoids gather / scatter operations.

C. *BANDWIDTH*

Applications which are bandwidth-bound can run faster on Intel Xeon Phi coprocessor systems. In such cases the speedup is not a ratio of the number of additional cores. It is more closely related to the total available aggregate bandwidth on Intel Xeon Phi coprocessors, which exceeds the bandwidth of the current class of Intel Xeon. Good memory access patterns (unit stride 1) and prefetching help to maximize the achieved memory bandwidth.

IV. SCALABLE OPTIMIZATION

A. *GRANULARITY*

If a problem size remains constant as the number of cores increases, the amount of work for each core decreases, becoming smaller and smaller (it is referred as granularity). If overhead remains constant for each unit of work, the overhead will make up a larger fraction of the runtime for smaller units of work. Thus, as the grain size decreases, the efficiency decreases. One way to check for this on a sequential system is to think about the problem size and amount of work that will be done in parallel.

B. *WORKLOAD BALANCE*

On Intel Xeon Architecture, processes are likely to have 200+ active threads. If all of the threads are issuing and executing SIMD instructions, things are very efficient. But if a small group of threads have more computation to complete and the remaining threads must idly wait for these few threads to catch up, the overhead and inefficiency increases. In other words the application fails to scale to use the available resources. One imbalanced task assignment can cause many threads or processes to sit idle and thus decrease system performance. If a developer sees poor workload balance across an application, the developer may want to explore techniques for improving it.

C. *BARRIERS*

System calls are barriers that many developers frequently overlook. The two most common unintended calls that impact scalability are malloc and get time of day. Calls to malloc encounter locks inside it that serializes its callers, serializes execution. If a thread allocates a large block of memory and then operates for a long time, this overhead is not as critical. Applications that make many calls to malloc will be well served by using a more efficient memory allocator. Intel Threading Building Blocks includes memory allocation calls that scale extremely well for this purpose. There are other third party memory allocation libraries available which also do better than standard malloc.

Any fork, join, mutex, lock or barrier potentially reduces efficiency. It is best to choose only the locks/barriers or controls necessary to ensure there are no data races present in the code.

V. RELATED WORK

Interactive computer aided code vectorization [1] presents user friendly, interactive web application system that detects *for* loop and vectorize it automatically, which allows programmer to simply select the *for* loop which needs to be converted into a vector equivalent operation, debug the result and compare performance gain. The goal of this system is to ease the job of learning and applying code vectorization technique since it requires skilled programming to convert a loop oriented code to a vector based operation. In most cases, vectorization modifies the program control flow. It modifies loop bounds of existing loops. It also adds new loops and new *if* statements.

Taking complete advantage of SIMD instruction in C program is a hectic task and requires non portable programming using intrinsic. Thus to allow C programmers to gain better performance of their application Whole Function Vectorization (WFV) [2][6] is introduced. WFV improves SIMD utilization without moving out of C programming. WFV is a technique for extending the use of SIMD across entire function.

To achieve high performance of the application code running on Intel Xeon coprocessors exploiting SIMD vector units is one of the important feature. In [3] several SIMD vectorization techniques are introduced. Techniques such as less-than-full-vector loop vectorization, Intel MIC specific alignment optimization, small matrix 2-D vectorization is implemented on Intel C/C++ and Fortran compilers for Intel Xeon Phi coprocessors. The performance results show that up to 12.5x performance gain on Intel Xeon Phi coprocessors is achieved.

In [4], evaluation of energy efficiency of different processors of Intel family using selected benchmark from the PARSEC suite with variable core count and vectorization technique to quantify energy under the Thermal Design Power (TDP). According to this evaluation vectorization should be given higher priority than parallelization as it is better in energy efficiency.

In [5] a method to trace and maintain flow information form source code to machine code when vectorization optimization is applied is introduced. Through this traceability Worst Case Execution Time (WCET) is benefitted. In this paper it is proven that vectorization not only improves average case performance but also WCET's.

VI. CONCLUSION

The compiler vectorizer can help user to get good performance out of Intel Xeon architecture through effective use of SIMD hardware, in addition to the benefits of threading over the many cores.

Users should look for the inner loops referred as hot spots to see whether they are vectorized, and if necessary help the compiler to vectorize them. For application s dominated by vectorizable kernels, the speedups may be large.

REFERENCES

[1] S. Sawadsitang, K. Suankaewmanee, S. Kuo, B. Bhumiratne, "Interactive Computer Aided Code Vectorization", 2012 Ninth International Joint conference on Computer Science and Software Engineering(JCSSE).

[2] Gil Raport, Ayal Zaks, Yosi Ben-Asher, "Streamlining Whole Function Vectorization in C using Higher Order Vector Semantics", 2015 IEEE International Parallel and Distributed Processing Symposium Workshops.

[3] Xinmin Tian, Hideki Saito, Serguei V. Preis, Eric N. Garcia, Sergey S. Kozhukhov Matt Masten, Aleksei G. Cherkasov and Nikolay Panchenko "Practical SIMD Vectorization Techniques for Intel Xeon Phi Coprocessors" 2013 IEEE 27th International Symposium on Parallel and Distributed Processing

[4] Juan M., Lasse Natvig, Jan C. Meyer,"Improving Energy Efficiency through Parallelization and Vectorization on Intel Core i5 and i7 processors",SC Companion: High Performance Computing, Networking Storage and Analysis,2012.

[5] Hanbing Li,Isabelle Puaut, Erven Rohou "Tracing Flow Information for Tighter WCET Estimation: Application to Vectorization" 2015 IEEE 21st International Conference on Embedded and Real-Time Computing Systems and Applications

[6] Ralf Karrenberg, Sebastian Hack "Whole-Function Vectorization" 978-1-61284-357-5/11/2011 IEEE

[7] O. Krzikalla1, K. Feldhoff1, R. Mller-Pfefferkorn1 andWolfgang E. Nagel1 Auto-Vectorization Techniques for modern SIMD architecture

[8] Ilan Baron,A Practical Parallel Algorithm for Solving Band Symmetric Positive Definite Systems of Linear Equations, ACM Transactions on Mathematical Software (TOMS), 323-332, Dec 1987.

[9] J. Jeffers and J. Reinders. Intel Xeon Phi Co-Processor High Performance Programming.

[10] P. Gavali, M. Shah, Gauri Kadam, Earthquake simulationof large scale structures using OpenSEES software on-grid and high performance computing in India,Earthquake simulation of large scale structures using OpenSEES software on Grid and high performance computing in India, Beijing, China,Oct 2008

[11] P. Gavali, M. Shah, Gauri Kadam, Kranti Meher, "Seismic response and simulations of reinforced concrete bridge using OpenSEES on high performance computing, CSI Transaction on ICT, Sep 2013.

[12] Chenggang Lal,Zhijun Hao,, Miaoquing Huang, Comparison of parallel programming Models on Intel MIC computer cluster, Parallel and Distributed Processing Symposium Workshop (IPDPSW),2014 IEEE International.

[13] https://software.intel.com/enus/articles/vectorizationessential

[14] https://software.intel.com/enus/articles/optimization-and- performance-tuning-for-intel-xeon-phi-coprocessors-part-1-optimization
[15] https://software.intel.com/en-us/articles/intel-xeonphi-coprocessors-performance-snapshot-on-cdac-cluster
[16] https://software.intel.com/enus/articles/theimportance-of-vectorization-for-intel-many-integratedcore-architecture-intel-mic

