ISSN (Print) : 0974-6846
Indian Journal of Science and Technology, *Vol 9(43), DOI: 10.17485/ijst/2016/v9i43/91712, November 2016* ISSN (Online) : 0974-5645

A Survey of Performance Analysis Tools for OpenMP and MPI

J. Sairabanu[1*], M. Rajasekhara Babu[1], Arunava Kar[1] and Aritra Basu[2]

[1]School of Computer Science and Engineering (SCOPE), VIT University, Vellore - 632014, Tamil Nadu, India;jsairabanu@vit.ac.in, mrajasekharababu@vit.ac.in, arunava.kar2014@vit.ac.in
[2]School of Electronics Engineering (SENSE), VIT University, Vellore - 632014, Tamil Nadu, India; aritra.basu2014@vit.ac.in

Abstract

Objective: Due to the recent development of multiple parallel programming tools with varying features, it is difficult to choose the best tool according to the needs of the user. **Methods:** This problem is addressed by making a comparative analysis study of different features like license type, source code availability, targeted platforms and languages supportedby these diverse tools. There are different parallel programming languages that support the present multi-core architecture like Message Passing Interface (MPI) and Open Multi-Processing (OpenMP). These are widely used to provide different performance characteristics of parallelism in different test cases. The new architecture and the complexity strengthens the need to monitor and analyze the performance of the various OpenMP kernels and constructs on multi-core processors. **Findings:** There are many papers that have been published in the past but non of them focuses on a comparative study among the performance analysis tools (PATs) that we mostly opt for. This paper intends to analyze the parallel computing ability of OpenMP and MPI, besides helping the user to understand which tool suites his task the best. **Improvement:** This study shows that MPI offers the best performance characteristics in the field of shared memory programming whereas OpenMP is a better choice because of the global style of the resulting program. It also provides a roadmap to select the best tool when designing a parallel programming system.

Keywords: MPI, Multi-Core System, Multi-Threaded System,OpenMP, PAT, Parallelization,

1. Introduction

Modern applications require a lot of computational power which reduces the performance of the system. The solution lies in finding alternative processors or algorithms with low cost. An ineffective solution is a supercomputer which is very expensive and hence, not affordable by many institutions. Thus parallel programming has come to the fore as a very successful way of increasing the computation speed. Performance analysis is the process of gathering information about the execution characteristics of a program[1,2]. It lays the foundation for PATs, which are responsible for analysing the performance data, thereby enabling us to improve the efficiency of our programs.

In this paper, we have analysed six tools in detail. We have opted for these tools because they are targeted for Windows, Linux and Unix Operating System, which are the most commonly used platforms. In addition to that, their recent versions and source codes are available for download. In this paper, we have analysed these tools in a comparative manner by taking into account their current version, license type, home site, source code availability, binary packages, targeted platforms, languages supported and important features as the parameters as shown in Table 1.

The paper is organized as follows – Section 2 elaborates on the importance of PATs, Section 3 focusses on the parallel programming languages – OpenMP and MPI and their important features, Section 4 gives a detailed explanation of all the six PATs that has been analyzed along with a comparative analysis table, and Section 5 concludes the paper.

2. Performance Analysis Tools

** Author for correspondence*

Table 1. Comparative analysis of PATs for parallel programs in OpenMP and MPI platforms

TOOLS	VAMPIR	TAU	GLOW CODE	PABLO	PIN	INTEL VTUNE AMPLIFIER
CURRENT VERSION	VAMPIR 8.5	TAU 2.5	9.2	PCF 4.1	2.12	2013
LICENSE TYPE	Commercial (Evaluation copy only)	Open source	Open source	Free for education, research, and non-profit purposes.	Proprietary, free, but cannot be redistributed	Proprietary
HOME SITE	www.vampir.eu	http://www.cs.uoregon.edu/research/tau/home.php	https://www.glowcode.com	http://renci.org/research/pablo/	https://software.intel.com/en-us/articles/pintool/	https://software.intel.com/en-us/intel-vtune-amplifier-xe
SOURCE CODE AVAILA BILITY	No	Yes	No	Yes	Yes	No
BINARY PACKAGES	RED HAT RPM-No DEBIAN-No	RED HAT RPM-No DEBIAN-No	-	RED HAT RPM-No DEBIAN-No	-	-
TARGETED PLATFORMS	Linux, Teraflops, Fujit	Unix	All versions of Windows	Unix, Linux, Sun Solaris	Linux, OSX, Windows, Android	Linux, Windows
FEATURES	•Powerful zooming and scrolling in all displays. •Adaptive statistics for user selected time ranges. •Filtering of processes, functions, messages and collective operations. •Hierarchical grouping of threads, processes, and nodes. •Support of source code locations.	•Provides graphical analysis of all the performance analysis results, both in aggregate and single node, context or thread forms. •The user can quickly identify and point out the plausible sources of performance bottlenecks. •Gathers performance information through the concept of instrumentation of functions, methods, basic blocks, and statements.	•Helps programmers to find performance bottlenecks and detect memory leaks. •Ensures code coverage, isolate boxing errors, identify excessive memory usage, and find hyperactive and loitering objects. •Unrivaled speed allowsprogrammers to isolate and correct errors early and often, continuously building a solid foundation of clean code.	•It includes Autopilot, an infrastructure for real-time adaptive control. •It includes Virtue, a collaborative virtual environment for direct software manipulation •It includes Pablo, a scalable performance analysis toolkit. •Any data that can be translated to the self-describing data format can be analysed.	•PIN provides an extensive API for instrumentation at different abstraction levels. •It also supports callbacks for many events such as library loads, system calls, signals/exceptions and thread creation events. •A large array of optimization techniques are used to obtain the lowest possible running time and overhead memory use.	•Profiles dynamically generated code. •Shows thread relationships to identify synchronization issues. •Finds long synchronization waits that occur when cores are underutilized. •Removes the clutter of data gathered during uninteresting times. •Finds specific tuning opportunities like cache misses and branch mispredictions. •Data Access Analysis identifies memory hotspots and relates them to code hotspots.
LANGUAGES SUPPORTED	C, C++, FORTRAN, Java	Python	C, C++, C#, any .NET framework compliant language	C, FORTRAN	Platform independent	C, C++, FORTRAN, NET, Java

(PATs)

Performance is defined as the process of collection of information regarding the way a program gets executed in the processor of any system. This task is carried out by the PATs. A PAT consists of three interfacing software layers. These are known as the instrumentation layer, measurement layer, and the analysis layer[3,4]. The instrumentation layer tells us about the different events that will be executed by the processor. The measurement layer deals with the performance of the event that is being monitored. It helps the user to visualize how the tool performs this measurement. The analysis layer works with the data that has been calculated by the previous layer and displays it in a form that be visualized by the user in a comprehensible form using the performance tools.

2.1 Features of a Good PAT

- The tool should not crash frequently.
- It should not crash when the user performs a wrong action. Instead, diagnostic messages must be displayed to enable the user to rectify his error.
- Efficient error handling features and a debugger must be provided.
- There must be a documentation support for ease of use and a user-friendly interface.
- Adequate features should be provided to execute the desired task more effectively rather than making the user perform low level tasks.
- The tool should be able to handle large numbers of processes and long-running programs.
- Most parallel programmers work on different platforms simultaneously. So, the tool should be able to analyse and display the data regarding the performance charactertics using different ways based on the platform the user is working on.
- The tool should support hybrid environment, which is defined as a combination of shared and distributed memory.
- The tool should support one of the latest parallel programming trends of passing messages in the shared memory within a node as well as between nodes in a distributed memory.

The debugging, analysis and tuning of parallel programs is more challenging than serial programs because the performance of a parallel program is determined by the complex interactions between the hardware and software components of the system. Thus, efforts must be taken to reduce the inefficiencies derived from dependencies, resource contentions, uneven work distributions and loss of synchronization among processors. This is where PATs play a very important role.

3. Parallel Programming Languages

A parallel programming language consists of constructs which permit multiple instructions in different blocks of codes to be executed during the same clock cycle. This feature makes such a language unique from the sequential counterparts, which have its constraints allowing only a single instruction to be executed in a single clock cycle. Although parallel languages are essentially sequential at their base, their constructs have much looser constraints, which allow multiple tasks to be performed at a particular instant of time. OpenMP is the most widely used shared memory Application Programming Interface (API), while MPI is the most commonly used message-passing system API.

3.1 OpenMP

OpenMP serves as a standard for shared-memory parallel programming. It is an API with a flexible and user-friendly interface for developers of parallel applications in FORTRAN, C, and C++[5].The latest version is OpenMP 4.5 launched on 15 November 2015. The OpenMP standard has been jointly set up and developed by a group with members from major companies like Hewlett Packard, IBM, Sun Microsystems, Intel, Silicon Graphics, etc. OpenMP was originally developed based on parallel loops and was primarily aimed at handling dense numerical applications[6]. OpenMP has been attracting worldwide interest because of the use of a shared memory model, simplicity of its interface, and a simple and portable parallel programming model. The OpenMP API comprises of library routines, compiler directives and environment variables[7]. Library routines supervise threads, processors and environment variables. It also manages thread synchronization. Compiler directives controls the operations of the compiler, which processes the sections of the code that are designed for parallel execution. Environment variables executes of the OpenMP program.

3.2 Features of OpenMP

- OpenMP comes with mechanisms for the mapping of unstructured data.
- OpenMP can perform asynchronous execution of data and manage runtime routines for device memory management which deals with the processes of allocation and freeing of memory[8].
- It also has an in-built algorithm to parallelize loops with dependences.
- It provides support to divide loops into tasks. This leads to better thread management as all threads are no longer required to execute a single loop.
- Hint mechanisms are provided which ensures a user-friendly interface when it comes to assigning the relative priority of tasks as well as the preferred synchronization implementations.
- Single Instruction Multiple Data (SIMD) extensions are provided to specify the additional data-sharing attributes.

3.3 Message Passing Interface (MPI)

MPI serves as a benchmark for the developers of message passing libraries. The primary objective of MPI is providing a base for writing message passing programs that are user-friendly and can be easily implemented[9]. Its major focus is on the message-passing parallel programming model whereby the data is passed from the memory space of one process to that of another process. This can involve passing of data messages within the shared memory of a node as well as between nodes in a distributed memory. The MPI interface is not only practical, but also efficient and flexible.

3.4 Features of MPI

- MPI can be invoked and called by C, C++ and FORTRAN programs, which are the most commonly used high level programming languages.
- MPI is more like a specification rather than a particular implementation[10]. An error free MPI program should run on all MPI implementations.
- Sending and receiving messages are the two fundamental jobs of MPI. A tag integer is introduced alongside every message when it is sent. A communicator is also present to deal with the technical aspects of context and group. This communicator serves an important role in most point-to-point and collective operations. The rank of a process in the group is identified by the communicator and always referred to by the destination or source as specified in any send or receive operation. Each process is ranked in its group according to a linear numbering. MPI programs use the concept of contexts to separate messages in various portions of the code.

4. PATs for Parallel Programs in OpenMP and MPI Platforms

4.1 Vampir

Vampir is a commercial OpenMP and MPI analysis tool which was developed by Pallas GmbH[11]. Vampirtrace, also developed by Pallas, is a MPI profiling library that produces trace files which are analyzed using Vampir[12]. The Vampirtrace library comprises of an API which allows for the insertion of user-defined events into the trace files. A filtering mechanism is used during runtime to focus only on the higher priority events thereby limiting the amount of trace data to be processed[13]. Instrumentation is performed by adding the Vampirtrace calls to the source code and linking the application with the Vampirtrace library[14]. Vampirtrace automatically corrects clock offset for systems that do not have a global time reference. Vampir can graphically display the program state changes through a user defined interface. It also provides a statistical analysis of all the parallel operations and hardware performance counters. It is designed to be an user-friendly tool, using which developers can display the behaviour of the program at any level of abstraction. Powerful zooming and scrolling facilities are also provided to identify the exact location and reason behind the performance bottlenecks. The interface also provides a number of customization options through context-sensitive menus. The filtering capabilities help to limit the unwanted information by focusing only on the matters of interest. Several graphical displays are provided by Vampir for real time analysis and visualization. The timeline and parallelism display is very useful if the user wants to know about the application related activities that are being performed at any instant of time. Many other displays are also included for statistical analysis of the performance characteristics during execution of the program and a dynamic calling tree display. The current version of Vampir can support up to 512 processes. However, it is very impractical to display so many processes simultaneously, which does not make the interface very user-friendly[15]. So, a new version of Vampir is in the process of development to resolve these issues by providing a

hierarchical scalability display.

4.2 Tuning and Analysis Utilities (TAU)

TAU is a profiling toolkit designed for performance analysis of parallel programs. It supports the most popular high level languages like Java, C and Python[16]. TAU is capable of accumulating performance related information through the instrumentation of functions and statements in addition to the sampling based on events[17]. TAU supports all the latest C++ language features including namespaces and templates. This makes it a formidable competitor to other parallel processing tools in the market. The API also provides the user with the discretion of selection and analysis of profiling groups for organization and control of instrumentation. The instrumentation can be added into the source code by making use of an automatic instrumental tool which is quite similar to the Program Database Toolkit (PDT). This can be done manually by the use instrumentation APIor dynamically with the help of the Dyninst API during runtime in the Java Virtual Machine (JVM)[18].

"paraprof", the profile visualization tool of TAU, provides graphical analysis of all the performance analysis results, both in single node and aggregated node context as well as the thread forms, thereby enabling the user to identify and point out the possible sources of performance bottlenecks in the application with the help of the graphical interface. TAU v2.22has the advanced feature of a topological view to map the routine to the underlying topology. In addition to that, TAU can also help in the generation of traces that can be displayed using the trace visualization tools ofVampir and Paraver[19].

4.3 Pablo

The research group based on Pablo has designed a variety of software tools for the performance analysis of distributed systems. The software distributions are primarily intended for academic and government research sites and other non-profit organizations[20]. Pablo is primarily aimed at Unix, Linux and Sun Solaris platforms. The software available includes: Autopilot, an infrastructure supported tool for real-time adaptive control; Virtue, a virtual environment intended for direct software manipulation and alteration; and Pablo, a scalable performance analysis toolkit which acts as a GUI for source code performance correlation[21]. Its latest version PCF 4.1 is available for free download. Pablo can be used in a data analysis environment for the construction, configuration and execution of a data analysis graph. Because the environment is capable of processing any data specified in the self-describing data format, the Pablo software instrumentation need not be the source of the data. This is a major advantage as data analysis using Pablo is no longer limited to application program behaviour. Any data that can be translated to the self-describing data format can be analysed. Thus, one could use the Pablo performance analysis environment to explore the trace data drawn from a simulated computer architecture or to study the performance of operating system resource management policies on massively parallel systems.

4.4 GlowCode

GlowCode is a real-time performance analyzer and memory profiler primarily aimed at Windows application program developers using C, C++, or any other .NET framework-supported language. It helps the programmers in optimizing the performance of their application by providing tools to identify performance bottlenecks, isolate boxing errors, detect memory leaks and resource flaws, segregate portions of code with excessive memory usage due to the presence of hyperactive objects, trace the real-time execution of a program and tune and profile the code. All these features are very important when it comes to improving the efficiency of a program because they tend to reduce the performance of a processor. GlowCode's unrivalled speed enables programmers to rectify errors early, thereby proving an easy and quick solution to the construction of an error-free code. With a proactive development environment, GlowCode helps to keep any algorithm clear of bottlenecks. GlowCode 9 can profile code built using C, C++, or C# in Visual Studio 2013, as well as all 32-bit or 64-bit code written in any .NET Framework-supported language. GlowCode is the only tool that has been experimentally found to be able to handle large unmanaged processes directly at some test systems. It is a very user-friendly product since it can remove all of the endless hours of guesswork by identifying where the performance bottlenecks are. Most of the world renowned companies like Siemens, IBM, Hewlett Packard, Google, etc. test all of their code using GlowCode.

4.5 PIN

PIN is an instrumentation framework used for creation

of dynamic performance analysis tools[22]. There are three types of routines that form the base of this framework – instrumentation routines, analysis routines and call back routines. Instrumentation routines allow for the insertion of analysis routines whenever a code that is yet to be debugged is about to be executed[23]. Call back routines are only called when a certain event has occurred or specific conditions have been encountered[24]. PIN makes use of an API at different abstraction levels ranging from a single instruction to a complete binary module. It also supports call backs for events such as system calls, library loads and thread creation events. The working mechanism of PIN is based on taking control of the program as soon as it is loaded into the memory[25]. This is performed by the Just-In-Time (JIT) compiler, which recompiles small sections of the binary code. Advanced instructions to carry out real time analysis are then added to the recompiled code by the PIN tool. Finally, different optimization techniques are used to obtain the minimum possible overhead memory use, thereby leading to the most efficient running time.

4.6 Intel VTune Amplifier

Intel VTune Amplifier is a commercial performance analysis tool for Linux and Microsoft Windows operating systems. It comes with both GUI and command line interfaces and can be availed either as a standalone software or as a component of the Intel Parallel Studio package. It requires an CPU manufactured by Intel for its proper functioning. and can be used in various kinds of code profiling such as stack sampling, hardware event sampling and thread profiling[26]. The profiler provides the user with essential information such as the time spent in each sub routine, which enables one to identify the performance bottlenecks. The tool can be also used for the analysis of the performance of the respective threads[27]. It includes several features like software sampling, JIT profiling support, locks and waits analysis, threading timeline, source view, Hardware Event Sampling (HES) and Performance Tuning Utility (PTU). Software sampling gives the locations of where stack is called. Locks and waits analysis searches for synchronization delays arising due to underutilization of cores and removes the cluster of data formed during application startup leading to performance issues. JIT profiling support analyzes dynamically generated code. Threading timeline identifies the relationships between various threads in order to eliminate load balancing issues. Source view helps in viewing the sampling results line by line on the source code. Using the on-chip performance monitoring unit of an Intel processor, HES searches for cache misses and branch mispredictions which can lead to stalls. PTU helps VTune Amplifier XE users to identify memory hotspots by giving them access to experimental tuning technology. Whether one is amateur who is just into performance tuning or a professional dealing with advanced optimization issues, Intel VTune Amplifier provides all it users with a rich set of performance insight into the parallel computing performance of GPU along with scalability, bandwidth, caching and much more. Analysis is much quicker and easier using VTune Amplifier because it understands common threading models and presents information at a higher level which is easier to interpret.

5. Conclusion

Parallel programming has presented itself as a challenge for programmers. Although a number of high-level programming tools are available to facilitate this process, it is at no point a simple task. The more efficient parallel algorithm, the more complex is the task. The goal of this study is to help the user to choose the best available PAT in the market to suit his needs. It can be inferred from the analysis that programming in MPI offers the best performance in the field of shared memory programming. On the other hand, programming with OpenMP provides a more comfortable solution because of the global style of the resulting program as it supports sequential control among the parallel loops with which all programmers are usually comfortable. However, to ensure greater efficiency, the programmers must also be experts in the field of performance analysis in order to analyze the potential performance problems and their severity. In addition, they also need to have a good knowledge about complex user interfaces. Through the comparison and analysis of the PATs, we aim to show how one can choose the best tool when designing a parallel programming system.

6. References

1. Chanthini P, Shyamala K. A survey on parallelization of neural network using MPI and Open MP. Indian Journal of Science and Technology. 2016 May; 9(19). DOI: 10.17485/ijst/2016/v9i19/93835.

2. Kalyani R. Application of multi-core parallel programming to a combination of ant colony optimization and genetic algorithm. Indian Journal of Science and Technology. 2015 Jan; 8(S2). DOI: 10.17485/ijst/2015/v8iS2/59091.
3. Browne S, Dongarra J, London K. Review of performance analysis tools for MPI parallel programs. NHSE Review. 1998 Jan; 3(1):241–8.
4. Moore S, Cronk D, London K, Dongarra J. Review of performance analysis tools for MPI parallel programs. Recent Advances in Parallel Virtual Machine and Message Passing Interface Springer Berlin Heidelberg; 2001 Sep 23. p. 241–8.
5. Mohsen MS, Abdullah R, Teo YM. A survey on performance tools for OpenMP. Proceedings of the World Academy of Science, Engineering and Technology. 2009 Jan 20; 3(1):102–13.
6. Dagum L, Enon R. OpenMP: an industry standard API for shared-memory programming. Computational Science and Engineering. 1998 Jan; 5(1):46–55.
7. Rabenseifner R, Hager G, Jost G. Hybrid MPI/OpenMP parallel programming on clusters of multi-core SMP nodes. 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing, Weimar; 2009 Feb 18. p. 427–36.
8. Sato M. OpenMP: Parallel programming API for shared memory multiprocessors and on-chip multiprocessors. Proceedings of the 15th International Symposium on System Synthesis; 2002 Oct 2. p. 109–111.
9. Gropp W, Lusk E, Doss N, Skjellum A. A high-performance, portable implementation of the MPI message passing interface standard. Parallel Computing. 1996 Sep 30; 22(6):789–828.
10. Gabriel E, Fagg GE, Bosilca G, Angskun T, Dongarra JJ, Squyres JM, Sahay V, Kambadur P, Barrett B, Lumsdaine A, Castain RH. Open MPI: Goals, concept, and design of a next generation MPI implementation. Recent Advances in Parallel Virtual Machine and Message Passing Interface, Springer Berlin Heide; 2004 Sep 19. p. 97–104.
11. Knüpfer A, Brunst H, Doleschal J, Jurenz M, Lieber M, Mickler H, Müller MS, Nagel WE. The vampir performance analysis tool-set. Tools for High Performance Computing, Springer Berlin Heidelberg; 2008. p. 139–55.
12. Brunst H, Winkler M, Nagel WE, Hoppe HC. Performance optimization for large scale computing: The scalable VAMPIR approach. Computational Science-ICCS, Springer Berlin Heidelberg; 2001 May 28. p. 751–60.
13. Müller MS, Knüpfer A, Jurenz M, Lieber M, Brunst H, Mix H, Nagel WE. Developing scalable applications with vampir, VampirServer and VampirTrace. PARCO; 2007 Sep 15. p. 637–44.
14. Brunst H, Kranzlmüller D, Nagel WE. Tools for scalable parallel program analysis-vampir NG and Dewiz. Distributed and Parallel Systems Springer US; 2005. p. 93–102.
15. Brunst H, Hackenberg D, Juckeland G, Rohling H. Comprehensive performance tracking with vampir 7. Tools for High Performance Computing Springer Berlin Heidelberg; 2010. p. 17–29.
16. Mohr B, Brown D, Malony A. TAU: A portable parallel program analysis environment for pC++. Parallel Processing: CONPAR 94—VAPP VI, Springer Berlin Heidelberg; 1994. p. 29–40.
17. Mohr B, Brown D, Malony A. TAU: A portable parallel program analysis environment for pC++. Parallel Processing: CONPAR 94—VAPP VI, SpringerBerlin Heidelberg. 1994. p. 29–40.
18. Mohr B, Malony AD, Shende S, Wolf F. Zentralinstitutfür Angewandte Mathematik; The Journal of Supercomputing. 2002; 23(1):105–28.
19. Shende SS, Malony AD. The TAU parallel performance system. International Journal of High Performance Computing Applications. 2006 May 1; 20(2):287–311.
20. Reed DA, Aydt RA, Madhyastha TM, Noe RJ, Shields KA, Schwartz BW. An overview of the Pablo performance analysis environment, Technical Report, Department of Computer Science, University of Illinois, Urbana-Champaign; 1992 Nov 7. p. 1–45.
21. De Rose LA, Reed DA. Svpablo: A multi-language architecture-independent performance analysis system. 1999 International Conference on Parallel Processing, Aizu Wakamatsu City; 1999. p. 311–18.
22. Bach M, Charney M, Cohn R, Demikhovsky E, Devor T, Hazelwood K, Jaleel A, Luk CK, Lyons G, Patil H, Tal A. Analyzing parallel programs with pin. Computer. 2010 Mar; 43(3):34–41.
23. Reddi VJ, Settle A, Connors DA, Cohn RS. PIN: A binary instrumentation tool for computer architecture research and education. Proceedings of the 2004 Workshop on Computer Architecture Education: held in conjunction with the 31st International Symposium on Computer Architecture USA; 2004 Jun 19. p. 1–22.
24. Luk CK, Cohn R, Muth R, Patil H, Klauser A, Lowney G, Wallace S, Reddi VJ, Hazelwood K. Pin: Building customized program analysis tools with dynamic instrumentation. Acmsigplan Notices. 2005 Jun 12; 40(6):190–200.
25. Patil H, Pereira C, Stallcup M, Lueck G, Cownie J. PinPlay: A framework for deterministic replay and reproducible analysis of parallel programs. Proceedings of the 8th annual IEEE/ACM International Symposium on Code Generation and Optimization, NY; 2010 Apr 24. p. 2–11.
26. Tousimojarad A, Vanderbauwhede W. Comparison of three popular parallel programming models on the Intel Xeon Phi. Euro-Par 2014: Parallel Processing Workshops Springer International Publishing; 2014 Aug 25. p. 314–25.
27. Marowka A. On performance analysis of a multithreaded application parallelized by different programming models using intel vtune. Parallel Computing Technologies, Springer Berlin Heidelberg; 2011 Sep 19. p. 317–31.