"Spotting the techniques on OPENMP Compilers and its Optimization"

Hafsah Mahmood
Doing Software Engineering,
Semester 5, Fatima Jinnah Women University,
The Mall Rawalpindi,

Abstract: OPENMP is a parallel programming technique which is employed in order to improve the optimization. The research paper proposes a number of techniques which can be used to enhance the performance and execution of parallel programs by applying OPENMP and executing them in the OPENMP special compilers using twenty papers. The results yield that the compiler should be provided with the dynamic memory allocation of threads in order to rise the performance and runtime efficiency.

Key-words:OPENMP, OPENUH, Parallel Dynamic, Hyper-threading Technique, Compilers, Optimization

1. Introduction

OPENMP is mainly used for parallel programming which is implemented on shared and distributed memory architecture. The parallel execution of the threads in programs leads to the better utilization of the multi-processors and carve a way to better memory latency. The optimization can be achieved by execution of different threads on different logical processors. This leads to better synchronization as well as coherency.

For parallel programming, the criticality lies in the junction of shared and distributed memory architectures modelling resulting in single parallel programming model. The parallel programming can augment the performance of compilation and execution of the programs. They are playing significant role in the performance directed applications. The parallel programming is particularly true for those application in which large data collection is required. The OPENMP thread is used for the execution of OPENMP task scheduled for asynchronous execution.

The next section will explain the optimization techniques in OPENMP compilers and the frameworks which are proposed in this regard. This will be followed by the analysis parameters used for the evaluation of the techniques. At the end, conclusion and recommendations are being made along with the tables showing the results of the parametric analysis done on all papers.

2. Optimization techniques for OPENMP compilers

Parallel programs are usually linked with two types of the systems that include multiprocessor systems and shared memory multiprocessor systems. The formers are using processors in which communication is done through messages and memory is taken by the processors. On the other hand the later uses the OPENMP techniques and convert the serial code to parallel multi program. OPENMP has two major issues which include the programmer dependency on checking the correctness and accuracy of the parallel program and the scheduling of the tasks leads to another problem as it is also programmer dependent. Along with that optimization is a significant criteria for finding out the best approach which can be used to address the above mention problem. So for this purpose, following number of optimization techniques are briefly explained.

2.1 OPENMP in Parallel molecular dynamics for shared memory machine [1]

The execution of the program in the OPENMP is done sequentially as far as the parallel construct is not reached. The machines which are using shared memory can easily parallelize the loops. So a substitute solution to the DD loops is given in which DD loops are combined with communication libraries that are used for parallel MD codes. Private (scope is local to the iteration), reduction (local copy maintained by each thread) and shared (shared among all threads) variables are explained in the independent loops. The do loops are divided over each thread and an optimized way of executing the MD simulation is achieved for the programs which are having less sophistication.

2.2 Distributed memory architectures using OPENMP [2]

In order to optimize OPENMP programs so that the compilation process can be further improved and performance can be enhanced, a new technique called KLCOMP which is working on partial shared array model is proposed. The mentioned technique can be used as an alternative to MPI. It provides a portable programming environment. Optimization is achieved by using partial shared array model with the amalgamation of MPI in demand to diminish the amount of data redundancy and data volume, reduction of redundant communication due to its dependence on the producer/consumer relationship, and lastly engendering a nonlinear reference communication which is provided at compile time and as a outcome it lessens the run time overhead.

2.3 Compiler optimization techniques for OPENMP programs [3]

The factors which effect the optimization in the OPENMP API can be listed as control synchronization, data synchronization and data locality. So the optimization of the compiler is studied in which the code uses the internal representation and reaching definitions along with memory synchronizations. For every single synchronization node, the reaching definition is examine which leads to set of definitions that are used to synchronize a node. The optimization is done by reducing the overhead of coherence by flushing the shared data which is present on synchronization point. Redundant barrier remover can be improve by obtaining single parallel regions by amalgamation of all multiple parallel regions that in return results in enhanced performance of the complier.

2.4 OPENUH, an optimizing, portable compiler using OPENMP [4]

For enhancing performance in OPENMP programs, OPENUH, an OPENMP compiler version is proposed which customs the already current analysis and implementing a source to source translator procedure that is not delivered deprived of intrinsic backend. The OPENUH contains a set of layers which comprises a front set (C or C++&FORTAN77or90),IPA(Interproceduralanalysis), LNO (Loop nestoptimizer),LOWER_MP(transformation of OPENMP), WOPT (global scalar optimizer), WHIRL2C & WHIRL2F (IR to source for none-Itanium) and CG (code generation for Itanium). This compiler is predominantly design for Itanium platforms. It lifts the performance by modularizing the overall infrastructure. It is permitting diverse sub-teams of the code to execute concurrently.

2.5 Discovering the Usage of Hyper-Threading Technology for Multimedia Applications with Intel_ OPENMP [5]

The optimization can be achieved by execution of different threads on different logical processors. So the paper suggests Open MP compilers using hyper threading architecture and executing OPENMP pragmas. The goal is to optimize the media applications including SVM (support vector machine) and AVSR (audio video speech recognition). The compiler yields countless demands in memory systems by invoking thread level parallelism which causes inter-thread interference.The outcomes display that the 1.28x speedup routine is reached by the planned compiler on HT-enable single CPU system. The performance is even more prevailing in HT-enable dual CPU systems by the factor of 2.23x speed.

2.6 Executing OpenMP Programs on Pentium- and Itanium-Architectures [6]

The major focus is on the generation of effective multi-threaded codes which are directed by OPENMP directives and pragmas using C++ and FORTAN compilers in Intel architectures. The results obtain leads to the conclusion that the performance is made better when the programs are compiled with C++/FORTAN compiler with modification which is well suited with the Intel hyper-threaded technology provided in the Pentium 4 processor single and multi-processors and Intel Itanium processor family.

2.7 Using Array Section Descriptor for PC Clusters in OPENMP Compiler - [7]

The optimization technique which is adopted in this paper is new array description which is termed as ‘quad’. The quad consist of 4 integers which correspond to the first subsection which is referring to the head of the array, accessed section’ total length, non-accessed subsections’ length and the repetition number respectively. The two quads are checked first which are analyzed by two different processors and then the intersection operation among them takes place and is further analyzed for synchronization. After this push function is used to transfer the data.The compiler will use the division of the array into sub arrays. The modification will lead to 7.82 fold speedy performance.

2.8 Impression of Compiler-based Data-Prefetching Techniques Application Performance [8]

There is a need to improve performance in the multi-threaded applications. The paper is covering the extensive study of data prefetching and its impact on the multi-threaded applications. The compiler proposed framework is optimized by implementation of multi-level parallelism (MLP), inter procedural optimization (IPO), high level optimization (HLO), scalar optimization and task queuing model. The results of the serial execution are analyze with single thread execution and it is seen that the performance is enhanced many fold which is 11.88% to 99.85% gain for 6 out of 11 applications with the exception made by only one application which has less than 1% gain in its performance.

2.9 A Compile-time Cost Model for OPENMP [9]

Cost effectiveness is a significant factor, whenever we talk about the optimization in compiler. So there is a need to do the cost analysis by means of a cost model. The paper proposes a cost model which is inculcating major features of OPEN64 and considering more details of OPENMP programming. The model focuses on the activated thread-context and thread processing mapping and binding. The model captures the static scheduling and also measures the system noise during execution when cache misses takes place.The results reveal that 1.25% time as mandatory to the kernel for compilation and the outcomes which are gained in the early modelling can be recycle on evaluation of different threads.

2.10 BSP2OMP: A Compiler for decoding BSP programs to OPENMP [10]

There is a major difference in the performance issues which are persisted in the two shared and distributed memory models termed as OPENMP and BSP respectively. The paper converges these two models in single model called BSP2OMP for optimization and acting as a bridge between gaps. The proposed compiler is design for easy transformation of code from BSP to OPENMP. The header file of BSP2OPENMP is to be mentioned at the top of the program and interpreting the SPMD part which plays a key role in the BSP program. The course grain computation is used in the designing of the BSP programs which leads to enhance performance.

2.11 Incremental Check pointing technique for OPENMP Applications [11]

In order to overcome Peta-Flop failures, check pointing is used in which it enables the users to save the state and capability to restart of calculation after failure. The user invokes the checkpoint function. The portion of code that is not having any writes in the array is given space in the array. This will lead the data to compile in the separate threads and the checkpoints will be reduce in response. The analysis leads to the optimization by dropping the checkpoints size by 80% and permitting the asynchronous checkpoints in both sequential and OPENMP applications. The results are compared with NAS parallel benchmarks.

2.12 Performance information for automated analysis [12]

Parallel programming can augment the performance of compilation and execution. The performance experimentation of automation of source code is given by OPENUH and a scripting interface is provided by the Perl-Explorer. There is a sole address for all processor nodes. The nodes are associated through a memory hub to form a computational brick and memory is allocated to each node. The bricks are connected through memory routers and hence the hierarchy is formed. Outcomes show that the anticipated framework is positive in OPENMP and MPI code tuning, parametric characterization and power modelling.

2.13 Well-organized Use of Distributed Scratchpad Memory in MPSoCs [13]

OENMP compiler is used for partitioning of data in arrays. The compiler is using a light weight software address translation technique and enables the profile base allocation, which targets the mostly referenced subset of shared arrays. It allows dynamic locality at different regions. Whereas the compiler is not providing mechanism for sharing of the data without the provision of arrays. 8x speedup is achieved along with cache based data management solutions.

2.14 Design and Execution of OMPi Compiler [14]

The OMPi compiler is a compiler which compiles and executes the OPENMP program and converts the C serial code in to the parallel code containing threads. The OMPI uses the BFS (breath first scheduling) where as it occasionally uses the DFS (depth first scheduling). The DFS is used when the queue is full. The queue size when reaches to 30% empty then the throttling mode is disabled. The use of the PS-Threads leads to better performance in OMPI. In order to make the threads tied, id number is used which has to be remember by the task so that it can reuse it. Result yields better performance and optimized execution.

2.15 The Design and Implementation of OMPit Compiler [15]

The OMPIT is proposed in order to share data dynamically. This can help in debugging and tutoring of a compiler. Three different sharing behaviors are discussed in this regard which is assessed on the basis of work sharing logging including work assignment and timestamps. The output including text- files or visuals figures can be considered as the logging information. The compiler will be able to find load unbalance and unreasonable chunk sizes or sections number. The results are better than before but there is a drawback of the technique that for processing, some logged in information is required.

2.16 OPENMP compiler for Hybrid Computing Architecture [16]

For hybrid computing, OPENMP compilers can be used. Each parallel region can be executed by choosing the options of CPU or GPU by the user and the directives of the OPENMP compiler. The results show that the programs which are used for testing are as much optimized as they were expected while writing them by CUDA API in which the program synchronization is not required in abundant. The performance achieved by using the proposed framework however it shows that the rise in the performance of GPU modules takes place only by the factor of 2-7 times related with CPU modules. Moreover, the compiler is not using texture memory or shared memory along with less latency rate in GPU modules.

2.17 Static scheduler for Multi-core compiler in OPENMP [17]

The compiler task scheduling leads to the execution of the n tasks on p processers which use the two level scheduling (compilers and operating systems handling the scheduling). The compilers of OPENMP usually use the dynamic scheduling. The compiler goes through the following steps during compilation.

1. The extended parallel directives present in the library.
2. Parallel library support.
3. The source code conversion is done on the parallel directive

The serial and parallel program execution leads to parallel program speedup calculation and even distribution of load leads to load balancing.

2.18 Compiler Infrastructure [18]

The compiler infrastructure should be understandable by the programmer as to implement the program as well as to understand the ways to optimize the performance. The Rose and Cetus infrastructure can be used in order to create such systems which are capable of auto tuning. It also described ways to optimize the compilers using GPU modules which are dependent on characteristics of the program input. Showing the infrastructure leads to the results that the compiler should be containing fault tolerance for message passing applications.

2.19 An OPENMP compiler, MPtostream for heterogeneous parallel systems [19]

A compiler MPtoStream is designed in order to process AMDs stream in OPENMP compilers. The tasks are being divided over the CPU and GPU modules with the help of the set of compiler directives. Eager load and lazy save strategies are also used. The experimental results reveals that modification of only 11% achieved by extended programming using OPENMP directives can lead to boost in performance by 3.1 to 17.3 in heterogeneous systems.

2.20 OPENMP to GPGPU, a technique for translation and optimization [20]

A framework is proposed by converting standard OPENMP applications into CUDU based GPGPU applications. The purpose of the conversion is to achieve better programmability and creating OPENMP applications which are agreeable to GPGPU. The transformation of loop level parallelism to data level parallelism is acquired. Along with that matrix transpose technique is also employed. The results demonstrate that the optimization is attained by the deliberated framework for regular as well as irregular applications containing regular and irregular expression respectively up to 50x. The drawback is that the technique that is establish for the shared memory multi-processors cannot be used for GPU architectures directly and fine tuning of hardware resources is required.

3. Analysis

The task parallelism is supported by OPENMP in order to achieve high productivity and enhancement in programming. However, runtime support as well as efficient code is the key factors for proficient execution of a parallel program or a serial program converted to parallel one along with designing decisions. So there are some parameters on the basis of which the compiler needs evaluation. Those parameters along with analysis are given below.

3.1 Cost effectiveness

Cost effectiveness is a key factor while estimating the performance and optimization of the compiler. The cost models are used for this purpose which is nothing but mathematical algorithms. Conversely not using cost model cannot fully expressed the overheads and speedups which are caused by compilation techniques. The only papers during survey which are analyzing the compiler on the basis of cost model are parallel modular dynamics [1], compiler infrastructure [18] and cost model for a compiler [9].

3.2 Efficiency of code generated

The OPENMP compiler is usually converting the serial code into parallel code for compilation which must be efficient as it is one of the significant parameters in compilers. Analysis reveals that only some of the papers during survey are producing code that is not meeting standard to increase the productivity of compiler.

3.3 Case study

Case study is a precise illustration of something employed or investigated permitted to explain the proposed technique. Concepts get more understandable while explaining them through case studies. Maximum papers under examination are using case studies in order to explain their proposed frameworks.

3.4 Testability

Testing is an important criteria while evaluating the correctness of a technique under specified scenarios. The drawback of not employing this technique can leads to fatal errors which are unsolvable and undetectable. All the techniques in papers are testable.

3.5 Runtime Efficiency

Most of the compilers are failed when the compiler are not efficient at runtime. The compiler must be self-trained to handle the errors occurring at runtime. During the estimation, the results disclose that almost half of the new frameworks are generating runtime productivity and remaining is still working on it.

3.6 Reusability

It means that compiler can be re-usable in future with modification or not. Those that are not having room for amendment are of no use as the technology is increasing day by day. Only three of the proposed compilers are not mentioning any reusability including framework using check pointing [11], OMPit compilers [15]and a framework for automatic translation and optimization [20].

3.7 Portability

Portability means the aptitude of a program to implement in another operating system excluding the specified one for which it is built. Those frameworks are always given due consideration which are portable as they can be modified. The study during the survey reveals that approximately half of the proposed compilers were portable and half were designed for the particular operating system and hardware.

3.8 Benchmarks

Benchmarks are standards that are particularly used for comparison. There are many standards and benchmarks which are described for compiler construction and testing of compiler against those benchmarks can increase the reliability of the technique. Knowing its importance, only half of the papers used benchmarks for the evaluation and testing of their technique and hence making them more worthy.

3.9 Productivity

It implies that the proposed technique is increasing the productivity and performance than before or not. Mostly the proposed frameworks are productive than their previous release but some of them are although presenting a new way to compile a program but still they are not productive that the one prior to them.

3.10 Flexibility

A successful technique is that which is flexible enough to change. The change must be provided as with speed of increasing technology. The analysis show that half of the techniques are flexible and half are not.

3.11 Robustness

Robustness means that compiler can accept errors not earlier specified during execution and at runtime. Only some of the compilers are self-sufficient to be robust. The analysis show that parallel modular dynamics framework [1], OPENMP for distributed memory architecture [2] and [12] [13] [14] [19] are robust otherwise the remaining all are not meeting this characteristic.

3.12 Reliability

Reliability means that in the given timeline, compiler is working or not. It is an important feature that the compiler must possess. Those compilers that are not reliable are not worthy enough to be used by the users. The analysis disclose that all the compilers are reliable except the one compiler which is translating BSP programs to OPENMP programs as due to increase in overhead at runtime leads to poor results [10]. So it is not reliable.

3.13 Compiler and operating system options

Some of the compilers are supporting specific operating system and they can be run on such operating systems leading to less portable compilers. The analysis reveals that around half of the compilers have mentioned the operating system along with them and the rest are not.

3.14 Data representation

Data representation is to support the data in different formats. All the compilers have a mechanism to represent the data like arrays, Boolean etc. the analysis is made on the basis that which techniques have mentioned the data representation. So on this criteria only half papers are specifying the data representation and the rest are silent on this matter.

3.15 Design Tools

It explains the design model which is used for the implementation of the compiler. It helps the compiler to explain the specification as well. The study shows that only [6] [9] [10] [12] [17] and [19] are specifying the design model that they are using.

4. Conclusion

OPENMP are the compilers which are particularly designed for the parallel programming that is gaining a significant importance nowadays. Optimization and efficiency is an important feature in these compilers. The results show that most of the proposed techniques and frameworks which are discussed in the papers are improving and enhancing the optimization that is round about the frameworks leads to almost 90% of the efficiency gain.

The future work will be extended bidirectional. One for the memory banks, the performance can be estimated by regulating them automatically. Second the impression of the stated procedure in the parallel programs for task queuing models.

Acknowledgement

This research was supported by IEEE conference publications, ACM and springer's publications. I acknowledge my contemporaries from Fatima Jinnah Women University, the mall, Rawalpindi who delivered sympathetic and dexterity that expressively supported the research, though they may not resolve with all of the descriptions of this paper

I am enormously appreciative to Miss Mehreen Sirshar for her comments on an earlier version of the paper, even though any slipups are on my own and should not discolor the reputations of the mentioned well-regarded person.

References

[1] C. Raphaël Couturier, "Parallel molecular dynamics using OPENMP on a shared memory machine," Science Direct, 2005.
[2] H. C. Z. J. &. L. J. WANG Jue, "OpenMP compiler for distributed memory architectures," Science Direct, 2010.
[3] K. K. a. M. S. Shigehisa Satoh, "Compiler Optimization Techniques for OpenMP Programs," ACM, 2008.
[4] O. H. B. C. W. C. W. Z. Chunhua Liao, "OpenUH: An Optimizing, Portable OpenMP Compiler," Science Direct, 2006.
[5] M. G. S. G. R. L. S. S. Xinmin Tian Yen-Kuang Chen, "Exploring the Use of Hyper-Threading Technology for Multimedia Applications with Intel OpenMP Compiler," IEEE, 2003.
[6] M. G. S. S. D. A. E. S. P. P. Xinmin Tian, "Compiler and Runtime Support for Running OpenMP Programs on Pentium- and Itanium-Architectures," IEEE, 2003.
[7] K. W. Naoki Yonezawa, "An Implementation of OpenMP Compiler for PC Clusters based on Array Section Descriptor," IEEE, 2003.
[8] R. K. H. S. M. G. W. L. Xinmin Tian, "Impact of Compiler-based Data-Prefetching Techniques on SPEC OMP Application Performance," IEEE, 2009.
[9] C. L. a. B. Chapman, "Invited Paper: A Compile-time Cost Model for OpenMP," IEEE, 2007.
[10] A. Marowka, "BSP2OMP: A Compiler for translating BSP programs to OpenMP," IEEE, 2008.
[11] D. M. K. P. Greg Bronevetsky, "Compiler-Enhanced Incremental Checkpointing for OpenMP Applications," IEEE, 2007.
[12] O. H. V. B. S. C. B. C. A. D. M. L. C. M. a. B. N. Kevin A. Huck, "Capturing Performance Knowledge for Automated Analysis," IEEE, 2008.
[13] A. M. a. L. Benini, "An OpenMP Compiler for Efficient Use of Distributed Scratchpad Memory in MPSoCs," IEEE, 2012.
[14] P. E. H. V. V. D. Spiros N. Agathos, "Design and Implementation of OpenMP Tasks in the OMPi Compiler," IEEE, 2011.
[15] Y. C. C. L. C. K. Qiuming Luo, "The Design and Implementation of OMPit□An OpenMP Compiler Characterized by Logs for Parallel and Work-sharing," IEEE, 2011.
[16] T.-Y. L. J.-L. J. Hung-Fu Li, "An OpenMP Compiler for Hybrid CPU/GPU Computing Architecture," IEEE, 2011.
[17] a. D. G. Benbin Chen, "A Static Scheduling Scheme of Multicore Compiler for Loop Load Imbalance in OpenMP," Springer, 2010.
[18] S. M. Rudi Eigenmann, "Compiler Infrastructure," Springer, New York, 2013.
[19] T. T. W. G. J. J. &. X. X. YANG XueJun, "MPtostream: an OpenMP compiler for CPU-GPU heterogeneous parallel systems," Science Direct, 2012.
[20] S.-J. M. a. R. E. Seyong Lee, "OpenMP to GPGPU: A Compiler Framework for Automatic Translation and Optimization," ACM, 2009.

Table 1. Evaluation of Optimization techniques in OPENMP Compiler through specific Parameters:

Evaluation Parameters	Meaning	Possible Values
Cost effectiveness	The compiler has an effective cost model.	Yes/No
Efficiency of code generated	The code generated by the compiler for compilation is efficient.	Yes/No
Case study	The proposed methodology is working on different examples.	Yes/No
Testability	Testing is done on compiler or not.	Yes/No
Reusability	Compiler can be re-usable in future with modification or not.	Yes/No
Runtime efficiency	The time taken to execute the codes.	Yes/No
Portability	The compiler can run on different platforms.	Yes/No
Benchmarks	The comparison of features against a standard.	Yes/No
Productivity	The framework enhances the productivity.	Yes/No
Flexibility	The compiler is flexible enough to change.	Yes/No
Robustness	Compiler can accept errors not earlier specified	Yes/No
Reliability	In the given timeline, compiler is working or not.	Yes/No
Compiler and operating system options	Which operating system, the compiler is supporting.	Yes/No
Data representation	Data representation is to support the data in different formats	Yes/No
Design tools	Which design model is used for implementation of the proposed framework?	Yes/No

Table 2. Evaluation of Papers based on parameters:

Paper Ref No:	Cost effectiveness	Efficiency of code generated	Case study	Testability	Runtime efficiency	Reusability	Portability
[1]	Yes	No	Yes	Yes	Yes	Yes	No
[2]	No	Yes	No	Yes	Yes	Yes	No
[3]	No	Yes	Yes	Yes	Yes	Yes	No
[4]	No	Yes	No	Yes	No	Yes	No
[5]	No	Yes	Yes	Yes	No	Yes	Yes
[6]	No	Yes	Yes	Yes	No	Yes	No
[7]	No	Yes	Yes	Yes	No	Yes	No
[8]	No	Yes	No	Yes	Yes	Yes	No
[9]	Yes	Yes	Yes	Yes	Yes	Yes	Yes
[10]	Yes	No	Yes	Yes	No	Yes	No
[11]	No	Yes	Yes	Yes	No	No	Yes
[12]	No	Yes	Yes	Yes	Yes	Yes	No
[13]	No	No	Yes	Yes	Yes	Yes	No
[14]	No	Yes	Yes	Yes	No	Yes	Yes
[15]	No	No	Yes	Yes	Yes	No	No
[16]	No	Yes	No	Yes	No	Yes	No
[17]	No	Yes	No	Yes	Yes	Yes	Yes
[18]	Yes	No	No	No	No	Yes	Yes
[19]	No	No	Yes	Yes	Yes	Yes	Yes
[20]	No	Yes	Yes	Yes	Yes	No	No

Table continued…

Paper Ref No:	Benchmarks	Productivity	Flexibility	Robustness	Reliability	Compiler and operating system options	Data representation	Design tools
[1]	Yes	Yes	No	Yes	Yes	No	Yes	No
[2]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
[3]	No	Yes	No	No	Yes	Yes	Yes	No
[4]	Yes	Yes	Yes	No	Yes	Yes	No	No
[5]	No	Yes	Yes	No	Yes	Yes	No	No
[6]	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
[7]	No	Yes	No	No	Yes	Yes	Yes	No
[8]	No	Yes	Yes	No	Yes	Yes	No	No
[9]	Yes	No	Yes	No	Yes	No	Yes	Yes
[10]	Yes	Yes	Yes	No	No	Yes	No	Yes
[11]	Yes	Yes	Yes	No	Yes	No	No	No
[12]	No	Yes	No	Yes	Yes	No	No	Yes
[13]	Yes	Yes	No	Yes	Yes	No	Yes	No
[14]	Yes	Yes	Yes	Yes	Yes	Yes	No	No
[15]	No	Yes	Yes	No	Yes	No	No	No
[16]	No	Yes	No	No	Yes	No	Yes	No
[17]	No	Yes	No	No	Yes	No	Yes	Yes
[18]	No	No	Yes	No	Yes	No	No	No
[19]	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
[20]	Yes	Yes	No	No	Yes	No	Yes	No