Optimizing Speedup on Multicore Platform with OpenMP Schedule Clause and Chunk Size

Z I Abdul Khalib\textsuperscript{1,2}, H Q Ng\textsuperscript{1}, M Elshaikh\textsuperscript{1,2} and M N Othman\textsuperscript{1,2}

\textsuperscript{1} Embedded, Network and Advanced Computing Cluster, School of Computer and Communication, Kampus Pauh Putra, Universiti Malaysia Perlis 02600 Arau Perlis
\textsuperscript{2} Bioelectromagnetics Research Group, Kampus Pauh Putra, Universiti Malaysia Perlis 02600 Arau Perlis

zahereel@unimap.edu.my

Abstract. Despite the description of schedule type which can be easily found at any OpenMP reference material, little is known about the effect of different schedule type and chunk size on the parallel performance of shared memory multicore processor. Literature shows that performance analysis on different multicore platform overlooked the effect of different schedule type and chunk size, where often it was not explicitly specified. Hence, default assignment of the loop iterations among threads is assumed. By default, static schedule is used and size of chunk which is the ratio of total number of iteration to the number of threads is implemented. This research analyses the effect of different schedule type and chunk size on speedup achieved of different shared memory multicore platform under regular workload. Apart from that, the performance gain obtained after turning on/off certain multicore technologies and after turning on/off selected number of active cores per processor is also analysed. Results shows that different multicore technology exhibit different speedup value under different combination of schedule type and chunk size. Apart from that, it also observe that different multicore platform is better than the other in terms of speedup as the number of cores are increased.

1. Introduction

Current trend of increasing the number of cores on a single die increases the demand of parallel programming, so as to fully utilize all the cores at the same time. As opposed to traditional single-threaded application which only uses one of the cores during execution time, multithreaded application has the tendency to exploit multicore processor to its full capabilities. This can be accomplished by generating multiple threads, assigning each thread to one of the cores, and running all the threads in different cores concurrently.

Nevertheless, not all applications support multithreading by default. With regards to OpenMP, to make a serial application becomes parallel, a programmer needs to hunt for the optimum spots in their code and insert parallel construct in between lines of the code. Sometimes, a particular code is required to be rewritten so that the parallel construct can be fit into it. As parallel programming is becoming popular, OpenMP API has become the de-facto standard for high level shared memory parallel programming [1]. OpenMP API is a C/C++ and Fortran compiler extension which gives programmer the ability to add parallelism into a source code without the need to significantly rewrite the code. To date, OpenMP Architecture Review Board (ARB) had introduced an ample amount of...
parallel directives which offer different patterns of parallelism, in order to optimize CPU utilization [2].

In enhancing parallel performance, part of the compiler directives – schedule type and chunk size [3] are of paramount importance. With respect to multicore technology, some work has been done to evaluate the performance of different parallel programming language, construct, tools, and libraries [4].

However, with respect to literature review done, the performance of a parallel code with different OpenMP schedule types and varied chunk sizes on shared memory multicore processor is barely known. The lack of guideline on assigning appropriate schedule type and chunk size could cause an OpenMP user to leave the setting as default. Thus, the higher parallel performance gain which might be produced by other schedule option is indirectly being neglected. Since schedule type decides how the workload is divided among threads, whereas chunk size defines the granularity of work, there is a need to determine the suitable schedule type along with appropriate chunk size of a parallel code. Nonetheless, how much contribution does these directives gives in terms of performance enhancement of different multicore processors is yet to be identified.

2. Related work

Please To date, there are very little research that has been done on investigating the effect changing schedule clause and chunk size. Since there is no explicitly specified schedule by [5], it is assumed that the authors were using default setting. In the paper, the authors compared the parallel performance of Intel Core i7-2600 and AMD Opteron Processor 8347. They obtained speedup which is close to the number of physical cores when the matrix size is large enough at both platform. Also, they found that increasing the number of cores reduces the parallel efficiency.

Running dense matrix multiplication algorithm at a server which has two Intel Xeon 5650 processors, static schedule without specified chunk size gave fastest execution among than other schedule type with chunk sizes varied from 1 to 32 [4]. The study showed the effect of small size of chunk and compared the performance to a loop scheduling policy they created.

By using transpose then multiply approach, the highest speedup achieved by [6] is slightly higher than the total number of logical cores detected. And, the experimental platform was a quad-core processor with hyper-threads. In certain cases, hyper-threading could be beneficial to improving performance of parallelism.

With parallel extension of Microsoft .NET Framework version 4.0, 3.5× speedup could be achieved at Intel Core i5-2400, which is a multicore platform with 4 physical cores [7]. Though different programming language and parallel programming model were used, a speedup value which is close to the ideal case can be obtained when the cores have their own duplicated CPU resource.

[8] used Pthread with SIMD instruction, he obtained 8.35× speedup in running native matrix-matrix multiplication algorithm. However, the experimental platform is not clarified. Nonetheless, the author did mention that 8 threads were used and the AVX SIMD instruction implemented was of 256 bit.

By running a parallel matrix-matrix multiplication at a processor with MIC architecture, the parallel speedup of 1.83× was achieved [9]. The rather low speedup is because the problem size is not large enough, as the serial execution of the algorithm is just 312.83s. Higher speedup could be expected as the problem size grows. Running the same algorithm, 8.58× speedup was achieved by 12 cores (24 threads in total) at Intel Xeon processor [10].

[11] compared the SIMD performance of the matrix multiplication algorithm obtained through inline assembly language and intrinsic function. Besides, they also compared the speedup gained between Intel and MSVC++ compiler. They found that Intel compiler generates code which requires fewer number of cycles, hence it is a better choice in running a code with SIMD instructions. Nonetheless, the average speedup achieved was about 2.1× and 2.08× when the intrinsic function and inline assembly were implemented with Intel and MSVC++ compiler respectively.
3. Experimental Methods
The methodology is adopted to observe the performance improvement provided by multicore architecture of three processors, specifically Intel Core i5-2410M and AMD A12-9700P. The research is conducted by first preparing the Square matrix multiplication algorithm which involves regular workload per loop iteration.

Next the serial code is parallelize using OpenMP directives. Then, the parallel code is verified for correctness against known parallel problem like race condition. Next, the code is executed repetitively for sampling, where certain multicore technologies are enabled/disabled by issuing command at the Linux terminal.

![Workflow Methodology](image)

4. Result and discussion
This section focuses on evaluating the parallel performance of CPU + memory bound code with regular workload per iteration under different OpenMP schedule type and chunk size for the three different types of multicore processor. The assigned chunk size divides the total outermost loop iteration among threads available.

Referring to figure 2, If the directive combination of (static, 250) is used, and there is N×N×N iteration to be run in parallel, each thread would be assigned 250×N×N iteration in a round-robin fashion and execute it until all the N×N×N iteration is finished.
1. #pragma omp parallel shared (A, B, C) private (i, j, k)
2. #pragma omp for schedule (static, 250)
3. for i = 0 to N - 1 do
4.     for j = 0 to N - 1 do
5.         for k = 0 to N - 1 do
6.             begin
7.                 A[i, j] += B[i, k] × C[k, j];
8.             endfor
9.         endfor
10.     endfor

Figure 2. Matrix multiplication algorithm

Figure 3 compares the speedup achieved when different schedule type and chunk sizes are used in parallelizing matrix multiplication algorithm, on Intel Core i5-2410M platform, with different number of active cores. The maximum speedup achieved is approximately 2.0×, instead of 4×, which is the ideal case when there are four cores. This is due to the fact that Intel multicore processor comes with two terms, which are “physical core” and “logical core”. Physically, there are two physical cores on a processor. Because of HT, one physical core can run two threads simultaneously. Hence, there are 4 logical cores detected by the OS. Also, this logical core could not give equivalent performance as to physical core in this case, because the code bounded by CPU resources. Thus, the lower speedup than the ideal case. This fact is further proven by the graph of C0C1 and C0C2. Both C0C1 and C0C2 are viewed as two logical cores by OS. By right, C0C1 consists of two physical cores whereas C0C2 consists of one physical core and one logical core. The higher speedup between the two tells that physical cores are in demand in running this particular CPU- and memory-bound parallel code. By comparing the speedup between C0C1C2C3 (two physical cores with HT) and C0C1 (two physical cores), the former is of higher value. By comparing the parallel execution time of C0C1 and C0C1C2C3, it has been found that HT gives 1.2× ~ 1.3× faster execution in this case.

Turbo boost technology is a dynamic frequency scaling mechanism. The clock frequency of each core could be increased to a certain limit, if current workload is operating under power and temperature limit, and under limit of the Thermal Design Power (TDP) of that particular platform [12]. Since this technology is a dynamic feature, the clock frequency of processor will change dynamically, depending on the limiting factors. Thus, it is uncertain about the operating frequency would be. The faster parallel execution time of the code, and hence higher speedup, in any number of active cores might be due to the activation of this feature.

![Figure 3: Speedup comparison](image-url)
Figure 3. Graph of speedup against different schedule type and chunk size of parallel matrix multiplication algorithm at Intel Core i5-2410M (a) Turbo Boost Enabled (b) Turbo Boost Disabled

Figure 4 compares the speedup obtained when different OpenMP directives are used in parallelizing matrix multiplication algorithm, on AMD A12-9700P platform, with varied number of active cores. Similar to Figure 3, dual-core is capable of producing comparable speedup as to tri-core, just like two non-HT cores (in Intel Core i5-2410M) and one module with two cores (in AMD A12-9700P). It can be deduced that the two multicore platforms are meant to perform better in even-numbered cores with each core having duplicated CPU resources.

As shown in Figure 4(a), the speedup of dual- and tri-cores give higher speedup than quad-core. In terms of execution time, quad-core is still the winner among them. The serial execution time of quad-core is much lower than that of dual- and tri-cores. Although the parallel execution time of quad-core is lower than the other two, it differs not much (about ~60 seconds). Hence, when applied to the speedup formula, quad-core gives lower speedup than dual- and tri-cores, due to the small gap of time difference between serial and parallel execution time on quad-core and large gap of time difference between serial and parallel execution time on dual- and tri-cores.

When Turbo is disabled, both serial and parallel execution time decreases with the increment of number of active cores. Despite the shorter serial execution of tri-core as compared to that of dual-core, these two cases possess similar parallel execution time, hence the higher speedup produced by dual-core, as illustrated in Figure 4(b).
Figure 5. illustrates the speedup delivered when different OpenMP directives are used in parallelizing matrix multiplication algorithm, on ARM Cortex-A53 platform, with different number of active cores. ARM Cortex-A53 is a quad-core processor with each core having individual CPU resources. The two former multicore platforms which have shared and duplicated resources give the maximum speedup which is slightly more than 2, even when all cores are active, due to CPU resource restriction.

As is observed from figure 5, when number of active cores increases, the speedup increases. The graph of speedup of all the multicore platforms shows the range of performance improvement that could be obtained from parallelism. In running parallel matrix multiplication algorithm with regular workload per iteration, the workload is enough to make the scheduling overhead negligible, regardless of the combination of OpenMP directives used. Since all of the size of chunk specified is divisible by the problem size and number of threads, the workload is equally divided among threads in all kinds of directive combination. The size of chunk decides how many times the scheduling of task occur. The bigger the chunk, the lesser the scheduling overhead, as there is less request of chunk from thread. Yet, the overhead is negligible in all cases.

Figure 5. Graph of speedup against different schedule type and chunk size of parallel matrix multiplication algorithm at ARM Cortex-A53
5. Conclusion
In a nutshell, it is not always easy to identify the appropriate schedule and value for chunk size in running a parallel algorithm. It may depend (among other things) not only on the code in the loop, but also on the specific problem size and the number of threads used (Chapman, Jost, & Pas, 2008). The observations produced in this chapter may vary when the algorithm and/or problem size is different. The performance when a total of 4 logical cores detected is summarized in Table 1. The Turbo technology contribution for parallelism is the ratio of execution time of parallel algorithm when Turbo is disabled and when Turbo is enabled. The speedup of 4 logical cores as compared to 2 logical cores is the ratio of execution time of parallel algorithm when C0C1 is active and when C0C1C2C3 is active.

6. References
[1] Cownie J 30 Nov 2016 Retrieved December 1, 2016, from OpenMP: http://www.openmp.org
[2] ARB O November, 2015. OpenMP Application Programming Interface. Retrieved November 3, 2016, from OpenMP: http://www.openmp.org/specifications/
[3] OpenMP Loop Scheduling. 4, September 2012. Retrieved November 6, 2016, from Intel Developer Zone: https://software.intel.com/en-us/articles/openmp-loop-scheduling
[4] Ajkunic E, Fatkic H, Omerovic E, Talic K and Nosovic N 2012 A Comparison of Five Parallel Programming Models for C++ Proc. of the 35th International Convention MIPRO Opatija, Croatia IEEE p1780-84
[5] Anchev N, Gusev M, Ristov S and Atanasovski B 20-24 May 2013 Intel vs AMD: Matrix multiplication performance. 36th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), p 182-7
[6] Tomikj N and Gusev M 21-25 May 2018 Parallel Matrix Multiplication. 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO) p 204-9
[7] Delić M, Omerović E, Sokolovic A and Nosović N 2013 Performance Analysis of the Parallelized Matrix Multiplication Using C# Threading. 36th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO) p. 1285-88 Opatija IEEE
[8] Pradyumna S 2017 Performance Comparison of Matrix Multiplication. International Conference on Innovative Mechanisms for Industry Applications (ICIMIA 2017) p 461-6 Bangalore IEEE
[9] Wang E, Zhang Q, Shen B, Zhang G, Lu X, Wu Q and Wang Y 2014 High-Performance Computing on the Intel® Xeon Phi™: How to Fully Exploit MIC Architectures. In MIC Optimization Example: Matrix Multiplication p. 231-55 Haidian Beijing China Springer International Publishing Switzerland.
[10] Thouti K and Sathe S R 2012 Comparison of OpenMP & OpenCL Parallel Processing Technologies. International Journal of Advanced Computer Science and Applications (IJACSA) 3 56-61.
[11] Hassan S A, Hemeida A M and M M Mahmoud, M. 2016 Performance Evaluation of Matrix-Matrix Multiplications Using Intel’s Advanced Vector Extensions (AVX). Microprocessors and Microsystems 47 369-374.
[12] Intel. 1 November, 2016 Retrieved November 11, 2016, from Frequently Asked Questions about Intel® Turbo Boost Technology: http://www.intel.com/content/www/us/en/supportprocessors/00005641.html?wapkw=turbo+boost+2.0