Multi-Core Program Optimization: Parallel Quick Sort in Intel Cilk Plus

Sabahat Saleem¹, M. IkramUllah Lali¹, M Saqib Nawaz¹, and Abou Bakar Nauman²

¹Department of Computer Science & IT, University of Sargodha, Sargodha, Pakistan,
²Department of CS & IT, Sarhad University of Science & IT, Peshawar, Pakistan.

Abstract. New performance leaps has been achieved with multi-programming and multi-core systems. Present parallel programming techniques and environment needs significant changes in programs to accomplish parallelism and also constitute complex, confusing and error-prone constructs and rules. Intel Cilk Plus is a C-based computing system that offers straightforward and well-structured model for the development, verification and analysis of multi-core and parallel programming. In this article, a parallel program is developed using the framework of Intel Cilk Plus. A sequential quick sort program written in C++ language is converted to parallel quick sort program to achieve better performance. Converted program in Cilk is then checked for various conditions using tools of Cilk and after that, performance and speedup achieved with parallel quick sort is discussed and reported.

Keywords: Multi-Core · Single-Core · Cilk · Parallelism · Sorting Algorithms · Quick Sort · Speedup.

1 Introduction

With current technology, improving the number of transistors does not increase the computing capability of a system. One of the new techniques used to increase the performance of a system is the development of multi-core processors. In multi-core processors, two or more processors are used in order to increase efficiency and enhance performance [7]. Multi-core processing enhances user experience in many ways: improving the performance of activities that are bandwidth-intensive and compute, boosting the number of PC tasks that can be performed simultaneously and increasing users (workers) total number that can utilize the same server or computer and the architecture’s flexibility will scale out to meet new usage models that are bound to arise in the future as digitized data continues to proliferate [2].

Nowadays, multi-core computing is a flourishing area as computers having single-core processor are reaching the physical limits of possible speed and complexity. Various companies have already produced and are working on multi-core products include Intel, ARM, AMD, VIA and Broadcom [7]. Multi-core architecture is used both in personal systems and embedded systems. Flynn’s taxonomy classifies computer architecture into four classes that are Single Instruction stream, Single Data stream (SISD), Single Instruction stream, Multiple Data stream (SIMD), Multiple Instruction stream, Single Data stream (MISD) and Multiple Instruction stream, Multiple Data stream (MIMD). Out of these four classes, only MIMD machine can execute multiple instructions simultaneously in combination of working on a separate and independent data stream [23]. That’s why MIMD model is the most suitable for parallel computing.

Intel Cilk Plus language is designed for the writing of multi-core programs. Cilk offers a model that is well-structured and simple for program development, analysis and
verification of written program [10]. Any program that works on an Intel single-core processor will work with an Intel multi-core processor and Intel Cilk Plus is considered the fastest and easiest way to tackle both multi-core and vector processing power [8]. Objective of the Intel Cilk is to achieve parallelism and enhance performance and speedup. A program performance running on a single-core can be enhanced over multi-core by splitting the whole program into different threads which can concurrently run on multiple processors. Due to the increasing popularity of multi-core technologies, many attempts of parallelism have been reported in literature. Apart from existing techniques, there is still enough room for developing efficient technique for better parallelism. In this article, multi-core program optimization is done where a single-core sorting algorithm (quick sort) is converted to a parallel programming code using Intel Cilk Plus technology to achieve efficient parallelism.

2 Intel Cilk Plus Technology

The Intel Cilk Plus language, developed at Cilk Arts is built on the Cilk technology at M.I.T. over the past two decades. With Cilk Plus, C language is extended for writing of parallel applications that efficiently make use of multiple processors. The Cilk Plus language is well suited for divide and conquer algorithms. Divide and conquer algorithms are tackled mostly with recursive technique and Cilk support these techniques [4]. Intel Cilk Plus is a part of the C compiler, therefore no extra steps are required while working with any code of C or C++ in Cilk Plus [14]. Intel Cilk Plus maps an arbitrary number of tasks to a limited set of workers and parallelism is achieved without supervising call chains and also without oversubscribing resources. In Intel Cilk Plus, parallelism is achieved by exploiting the number of workers available in the pool instead of usage of unbound resource during scheduled tasking [14]. For parallel programming in Cilk Plus, three keywords offer a powerful model.

2.1 Cilk Plus Keywords

In order to achieve task parallelism, Cilk plus offers three powerful expressions that are: `cilk_for`, `cilk_spawn` and `cilk_sync`. In Cilk, `cilk_for` is the replacement for C for loop, where iterations in loop are allowed to run in parallel [4]. In `cilk_for`, body of loops executes in parallel, so in order to avoid race conditions, it must not modify control variable as well as any non-local variable. `Cilk_spawn` keyword is used to start parallel computations. It informs the runtime system that spawned function can run in parallel. `Cilk_sync` keyword is used to wait for the spawned procedures to complete. `Cilk_sync` function cannot run in parallel with spawned functions. Cilk offers race detector tool `cilkscreen` that monitors Cilk program execution.

The `cilkscreen` tool detects all data races that are encountered during execution. It reports only those races that occur due to Cilk keywords. For efficient parallel program, all race condition detected by `cilkscreen` should be resolved first. Cilkview scalability and performance analyzer tool reports statistics about a parallel Cilk program. It predicts how the performance of program in Cilk scales on multi-cores. It also presents a graphical view of performance and scalability. Analyzer measure and evaluate the expected speedup on two, four, eight, sixteen and thirty two multi-processors [4].

3 Sorting Algorithms

In most frequent operations that computer performs, sorting operation is the one in it. Sorting algorithms are kind of algorithms that put elements in certain order. Methods for sorting are generally based on the comparison between elements and record moves.
Generally utilized sorting algorithms are: bubble sort, quick sort, simple sort, merge sort, heap sort, and so on. All sorting algorithms are different from one another on the basis of their performance and efficiency. Sorting algorithms allow us to put information in some meaningful order and are used in many applications. Database systems make huge use of sorting operations [26]. They are also used in sparse matrix multiplication, image processing, computer graphics, statistical methodology and map reduce [22, 23]. Due to the importance of sorting algorithms, efficient sorting algorithms are needed for various parallel architectures.

Along with advantages, sorting methods have disadvantages too. Main disadvantage is that the best sorting algorithms have performance of $O(n \log n)$. It means that large data will take long time to sort. Best, average and worst time complexity of above mentioned sorting algorithms are given in table 1.

<table>
<thead>
<tr>
<th>Sorting Algorithms</th>
<th>Best Case</th>
<th>Average Case</th>
<th>Worst Case</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bubble Sort</td>
<td>$O(n)$</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
<td>1</td>
</tr>
<tr>
<td>Insertion Sort</td>
<td>$O(n)$</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
<td>1</td>
</tr>
<tr>
<td>Bucket Sort</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>1</td>
</tr>
<tr>
<td>Quick Sort</td>
<td>$O(n \log n)$</td>
<td>$O(n \log n)$</td>
<td>$O(n^2)$</td>
<td>$\log(n)$</td>
</tr>
<tr>
<td>Merge Sort</td>
<td>$O(n \log n)$</td>
<td>$O(n \log n)$</td>
<td>$O(n \log n)$</td>
<td>$n$</td>
</tr>
<tr>
<td>Heap Sort</td>
<td>$O(n \log n)$</td>
<td>$O(n \log n)$</td>
<td>$O(n \log n)$</td>
<td>1</td>
</tr>
</tbody>
</table>

Insertion sort, bubble sort and bucket sort are the simple sorting algorithms with average time running time of $O(n^2)$ and $O(n)$ respectively. In three of them, insertion sort is the most simple one and insertion sort works well when the records sequence is in order or when the value of $n$ is small [24]. In all of the sorting methods given in the table, quick sort is usually the faster one and has the best average performance of $O(n \log n)$. But in worst case when the record is in order, the quick sort has the worst case running time of $O(n^2)$. Both in average case and in worst case, heap sort running time is $O(n \log n)$. In average case, quick sort is little faster than heap sort. In worst case, heap sort has the advantage over quick sort algorithm and it is the biggest edge of heap sort over the quick sort, but heap sort is not a stable sort as heap operations may change the order of equal items. In merge sort, two or more than two lists or tables are merged in order to make two or more new ordered lists or tables. On comparison of merge sort with quick sort and heap sort, merge sort is a stable sort. Main drawback of the merge sort is that is require more memory as compared to quick sort. Most implementation of merge sort has to be done in $O(2n)$ space, whereas quick sort implementation can be done in place.

### 3.1 Related Work

Rashidy et al. [6] developed a parallel bubble sort algorithm using streaming model approach. In stream program, the input data to the program comes in stream form. In stream processing with a given set of data (stream), various kinds of operations can be applied to every element in the stream. In [6], they have written the code for parallel bubble sort in Java language and used Java library Jstream. Stream programs can be mapped easily to the architecture of multi-core or distributed systems [1]. All stream programs exhibit various features like these programs can work with large data streams and has stable pattern of computations. Bader et al. [13] in their article introduced SWARM, an open-source library for basic primitives that fully make use of multi-core
processors. In [29], they have presented a parallelized algorithm called Map Sort to accelerate Electronic Design Automation (EDA) softwares performance on multi-core architecture.

In order to speed up sorting process, multiprocessors are used for parallel sorting. Different parallel sorting algorithm such as bi-tonic sort [15, 16], column sort [20], parallel merge sort [12], parallel radix sort [21] and randomized parallel sorting [18] has been developed. In [9], a parallel sorting algorithm using Graphics Processing Units (GPU’s) has been devised. Qian and Xu [24] analyze the parallelism of sequential algorithms of sorting that are based on multi-core systems. Nadathur et al. [28] designed high-performance parallel routines of merge and radix sort for many-core GPU’s in CUDA. Man et al. [11] developed an efficient parallel algorithm psort in C language that is compatible with standard qsort. Any program that uses qsort from standard C library can be accelerated by simply changing qsort call with psort. For local sequential sorting, psort make use of standard qsort as a sub-routine. So if the qsort performance is improved, psort performance will automatically be improved.

4 Parallel Quick Sort in Cilk Plus

Among all sorting algorithms, quick sort is considered the most popular and vastly used sorting algorithms. Quick sort algorithm developed by Tony Hoary, is a divide and conquer algorithm. In quick sort algorithm, array is partitioned into two by a pivot element [17]. Those elements that are smaller compared to pivot is moved before it and all elements that are greater than pivot are moved after it. Moving of elements before or after pivot can be done in linear time [17].

In this section, sequential quick sort algorithm is converted to parallel quick sort in Cilk Plus. We have used Visual C++ express edition 2008 with Cilk technology. In parallel quick sort, recursive calls are spawned with cilk spawn keyword so they can run in parallel. The cilk sync keyword indicates that the recursive function cannot be allowed to continue its operation before all the requests of cilk spawn in the same recursive function have completed. Our parallel quick sort program written in Intel Cilk Plus is given below.

```cilk
// Cilk parallel program for Quick Sort Algorithm....
#include <cilk.h>
#include <cilkview.h>
#include <algorithm>

void quick_sort(int * beg, int * end)
void quick_sort(int * beg, int * end)
    {if (beg < end)
        {end = end--;
        int * pivot = partition(beg,end, bind2nd(less<int>(), *end));
        swap(*end, *pivot);
        cilk_spawn quick_sort(beg, pivot);
        pivot++;
        end++;
        quick_sort(pivot, end);
        cilk_sync;}}
int cilk_main()
    { int n;
        cout<<"Enter total number of elements that are to be sorted: ";
        cin>>n;
        int* a = new int[n];
```
cilk_for (int x = 0; x < n; x++)
    {a[x] = x; }
random_shuffle(a, a + n);
cout << "Sorting " << n << " elements" << endl;
cilk::cilkview cilkview;
cilkview.start();
quick_sort(a, a + n);
cilkview.stop();
cilkview.dump("Quick_Sort_Results", false);
cout << "Quicksort on " << n << " items succeeded." << endl;
cout << "Total Time taken by Quick Sort: "
    << cilkview.accumulated_milliseconds()/1000.f <<"seconds"<< endl;
return 0;}

Every program starts as a single-threaded program. To begin parallel activities, the context of Cilk runtime must be created and initialized and begin execution at a precise entry point in the program. This can be achieved in two ways. One approach is to replace the main with cilk\texttt{main}, while the second approach is to first create an explicit context and use cilk\texttt{::run} to enter in this context. In above code, we have replaced the main with cilk\texttt{::main}. In the code, cilk\texttt{::cilkview} object is created. Methods of start(), stop() and dump() are declared in the code that will produce the performance measurements. Above program will be run $N$ times. Cilk\texttt{view} will run the program one extra time with the option of parallel performance analyzer to predict the scaling of the performance. Above program is run with visual studio 2008 command prompt. In order to build and compile a Cilk program on a Windows system, cilk\texttt{pp} command is used. After successful compilation, cilk\texttt{pp} command makes an executable file of the program with the same name. Figure 1 shows how to run the program.

![Fig. 1: Running the parallel quick sort program](image)

As discussed in section 2.1, the cilkscreen tool checks the program operation while program is running with some test input. Cilkscreen detects all data races that are detected during execution of the program. For a reliable parallel program, all races that cilkscreen reports should be review and resolved [4]. Cilkscreen run the program on single worker and all the reads and writes to memory are monitored by cilkscreen. After program finishes, cilkscreen shows all the conflicts information that are found during write/read and any write/write. If any possible program schedule output result
that is different from the serial program execution then a race condition has occurred. Figure 2 shows that there are no data races in our parallel quick sort program.

![Fig. 2: Cilkscreen race detection of parallel quick sort](image)

By default, the worker threads number is equal to the number of cores on the system on which Cilk program is running. We can increase or decrease the number of workers from command line using `cilk_set_worker_count` command. Figure 3 shows the execution and time taken by the quick sort with different counts values of the worker to sort 1000000 elements.

![Fig. 3: Parallel quick sort program execution with different workers count](image)

*Cilkview* analyzer helps in understanding the parallel performance of a program. Result of *cilkview* appears on Visual Studio command prompt window after program finish execution and *cilkview* also display a graph that shows the predicted speedup and parallelism. Report of *cilkview* is partitioned in two sections. One is called the Parallelism Profile and the other is called the Speedup Estimate [4]. Parallelism profile is shown in figure 4 while speedup estimate (graph) is shown in figure 5. The Parallelism profile is shown on command prompt demonstrate all the statistics of program execution in Cilk. To get whole picture of the program parallel operation, *cilkview* scal-
Multi-Core Program Optimization: Parallel Quick Sort in Intel Cilk Plus

ability and performance analyzer run the program on a single worker and tracks all the spawns. Statistics about work, span, burdened spawn, parallelism, burdened span etc. are shown in Parallelism profile.

![Parallelism profile of parallel quick sort](image1)

Cilkview also measures the anticipated speedup on number of processors such as on 2, 4, 8, 16 and 32 processors. In the graph, estimates of speedup are displayed with ranges of lower bounds and upper bounds. Upper bound is the smaller number of workers and the program parallelism and the lower bound in the graph stands for estimated overhead. Total overhead depends on various factors, which include the program parallel structure and the total number of workers. Lower bound that is less than 1 shows that instead of speed up, the program may slow down when it is run on more than one processor [4]. Figure 5 shows the estimated speed up of program in the form of a graph.

![Speedup estimate of parallel quick sort](image2)
Cilkview performance and scalability analyzer help in understanding the parallel performance of the program written in Cilk. It shows the parallel statistics of the program, prediction of the program performance over multiple processors, benchmark the program on one or more processors and it also presents a graphical view of speedup and scalability.

5 Conclusion

Technology of multi-core has and is changing the basic concepts in the field of software research specifically those applications that runs on personal computers and on servers. Programs and applications that are developed with ideas of parallel programming not only improved the performance dramatically, it is also considered as key factor for the evolution and popularity of multi-core. Intel Cilk Plus technology present straight forward and well-structured model for the development, verification and analysis of multi-core and parallel programming. In Intel Cilk Plus, programmers can develop their applications either by writing program in the semantics of call/return and then specifying which call can be executed in parallel or by writing the program in multi-threaded environment.

In this work, popular sequential quick sort algorithm is converted to parallel quick sort algorithm in the Intel Cilk Plus model. Performance and speedup is achieved over sequential quick sort by converting sequential program into parallel Cilk program. Speedup and performance achieved by conversion of sequential quick sort to Intel Cilk program is discussed. Parallel quick sort developed in Cilk has been evaluated on Intel core i3 system with Microsoft Windows 7 Operating System and results are shown in figures and graph. Along with quick sort, merge sort is the second divide and conquer sorting algorithm. In future, it will be interesting to write parallel program for merge sort in Cilk.

References