Abstract— Now days various programming languages are used mostly sequential programming. But today, the multi-core processors are available in the market shares, and the programming personnel also must face the collision brought by the revolution of multi-core processor. The most challengeable is limit performance growth for single core microprocessors. So programmer or compiler explicitly parallelize the software is the key for enhance the performance on multi-core chip. In this paper, what we want to know is there any effective way that can reduce our time on rewrite or can automatically parallel the program for multi-processing purpose and do speedup the processing. In this paper how to compile code using OpenMP libraries. OpenMP introduces an additional level of the parallelism and it works with the thread in new programming paradigms and reduce the time of the program.

Keywords— Multicore, Multiprocessor, OpenMP, Parallel, Thread.

I. INTRODUCTION

Today, low-cost multicore processors are widely available for both desktop computers and laptops. As a result, applications will increasingly need to be parallelized to fully exploit the multicore processor throughput gains that are becoming available [1]. Why multi-core? We all think that multi-core can give us higher performance than only one core, but it should be in the situation of parallel processing. If a task is finished by each core, then the efficiency should proportional with number of core [1].

OpenMP
OpenMP is an API (application program interface) used to explicitly direct multithreaded, shared memory parallelism. OpenMP was introduced in 1997 to standardize programming extensions for shared memory machines. It has been widely used since then in parallelizing scientific programs. In OpenMP the user specifies the regions in the code that are parallel. The user also specifies necessary synchronization like locks, barriers, etc to ensure correct execution of the parallel region. At runtime threads are forked for the parallel region and are typically executed in different processors sharing the same memory and address space. Recently many chip manufacturing companies have announced multi-core processors. The OpenMP API is used for writing portable multithreaded applications written in FORTRAN, C and C++ languages. The OpenMP model provides a platform independent set of compiler pragmas, directives, function calls, and environment variables that explicitly instruct the compiler how and where to use the parallelism in the application.[2]

Fig. 1. Block Diagram of a Shared Memory Parallel Computer

In multi-core processors, each processor die has many processing elements and they typically have a common L2 cache. The whole die acts as a traditional multiprocessor system except that they now share the L2 cache. The advantage now is that we could execute different threads in these processing elements and the communication cost between the threads is very less since they share the L2 cache. Another advantage of having multiple cores is that we could use these cores to extract thread level parallelism in a program and hence increase the performance of the single program. A lot of research has been done on this area.[1][2] Many techniques rely on hardware based mechanisms and some depend on compiler to extract the threads So we see that we have a system which can be used as a traditional multiprocessor system with minimum communication cost and also it can be used to improve single thread programs. OpenMP [9].

So far have been used only in traditional multiprocessor
environments and we saw that the multi-core processors are very similar to a traditional multiprocessor. So it is natural to think of OpenMP in multi-core processors. Also we saw that multi-core processors can be used as a traditional multiprocessors or it can be used to implement TLS.[4] Consequently OpenMP can also be used in both the ways:

![Fig. 2. Fork-Join model for OpenMP Program](image)

**II. RELATED WORK**

The paper of OpenMP Parallel Optimal Path Algorithm and Its Performance Analysis used for solving the optimal path problem in large scale transportation networks in real time. In order to get the effective optimal path algorithms for the actual transportation networks, three optimal path algorithms are chosen to be parallelized.[3][4]. The parallel optimal path algorithms are then implemented based on dual-core. Processor and the speed-up ratio and efficiency of parallel algorithms are tested and analyzed in actual Xi'an road networks with 4525 nodes and 6616 paths. The experiment results show that the parallel algorithms devised in this paper are efficient and the speed-up ratio of the three parallel algorithms are satisfied in solving optimal path problems in large scale networks. However, the parallel algorithm experiments based on multi-core computer are still lacks, so we implemented 3 optimal path algorithms based on dual-core processor with OpenMP, multi-thread program. Meanwhile we design a visualized interface to show the computation result of optimal path, and the time cost. OpenMP is a set of interfaces designed for the applications on the memory sharing multiple processors. [6]

It is simple, portable and extensible. OpenMP is based upon the existence of multiple threads in the shared memory programming paradigm. A shared memory process consists of multiple threads. OpenMP uses the fork-join model of parallel execution. All OpenMP programs begin as a single process: the master thread. The master thread executes sequentially until the first parallel region construct is encountered, the master thread then creates a team of parallel threads. The statements in the program that are enclosed by the parallel region construct are then executed in parallel among the various team threads.[7] When the team threads complete the statements in the parallel region construct, they synchronize and terminate, leaving only the master thread.

**Parallel Dijkstra Algorithm**

Dijkstra’s algorithm is often used to solve the shortest path problem. The basic idea of it is to find the shortest path from the assigned vertex to other vertexes in the weighted graph based on the increase order of the path length. Classifying the vertexes in graph into 2 sets: s1 is the set that the shortest path has been determined and v-s1 is the set that the shortest path has not been determined. Initially s1 only contains the source vertex p0, v-s1 contains the other vertexes other than the source vertex. Selecting the vertex in v-s1 with the shortest distance to p0 and add it to s1. Keeping this process until s1=V and before the condition is reached, assign the serial program to the multi-thread parallel program to execute.[8]

**Heuristic-hierarchical Path Finding Algorithm**

The heuristic-hierarchical path finding algorithm is based on the idea to apply the nonhierarchical algorithm such as Dijkstra’s algorithm to different levels and the entrance points and exit points (node E) between high-level and low-level are obtained by the heuristic directing search approach. The algorithm procedure to find the satisfactory path, in terms of the minimum travel time based on the Manhattan distance and travel speed associated with the edges in the network, is described as follows:

**Step 1.** Obtain randomly both nodes O and D in the road network.

**Step 2.** Determine the level of road network which O and D are located.

**Step 3.** If O and D are located at the high-level then go to step.

**Step 4.** Else use heuristic directing search approach to find the node E1 and E2 from O and D to the high-level road network respectively, and then obtain a partial path P1 between O and E1 as well as a partial path P2 between D and E2.

**Step 5.** Use nonhierarchical algorithm to find a shortest path P3 between E1 and E2 in high-level road network.

**Step 6.** Connect P1, P2, and P3 to form a path from O to D and output it.

**Advantages**

1. One of the advantages of OpenMP is that its parallel programming model is much more structured than with hand-threading so that Assure’s explanation mechanism is stronger. We were confident that if Assure for OpenMP did not record any diagnostics, the parallel version would produce the same results as the serial version. Assure for Threads is weaker. It is only capable of detecting unsynchronized, shared accesses by multiple threads.

2. Parallelize small parts of application, one at a time
Can express simple or complex algorithms.

3. Code size grows only modestly. Expression of parallelism flows clearly, so code is easy to read

A. Disadvantages
1. Limited scalability, not much speed up.
2. Threads are executed in a non-deterministic order.
3. OpenMP requires explicit synchronization.

B. Limitations
1. OpenMP works only for shared memory.
2. Limited scalability, not much speed up.
3. Threads are executed in a non-deterministic order.
4. OpenMP requires explicit synchronization.
5. Limited scalability.

III. PROPOSED WORK

Objective
We are dividing the sequential algorithm into parallel algorithm, for this threading concept is used. Program divided into number of threads and each thread is executed independent of other threads.[1][4] As number of threads executed simultaneously, time required to execute that program reduces. Main objective of this project is to save the time required to execute the programs.

Input Stream (Data sets)
Input to the program: Input to the program is size of input data set and number of threads going to execute that program. This is done by using command line argument. For this following syntax is used int main(int argc, char *argv[]); argc will count the number of arguments given by user and argv[n] is the actual arguments. or we can set number of thread using following command: int omps_num_threads(); A f intrandom number = rand();

Expected Outcome
We tested the algorithms by writing the program using OpenMP on multi-core system and measure their performances with their execution times.

We describe the techniques and algorithm involved in achieving good performance by reducing execution time through OpenMP Parallelism on multi-core.

In output stream we are getting the time require to execute the program for this following timing functions are used:

Clock library included in include < time.h > library

clock start, end;

Accept input and initialize variables

start = clock();

program code

end = clock();

time exe = ((double)(end − start));

time exe will give the timer required to execute the program.

Working Module

The main goal of a parallel program is to utilize the multi-core resource variable in common for improving the performance of algorithm. There are some numerical problems which are large and complex. The paper solutions of which takes more time using sequential algorithm on a single processor machine or on multiprocessor machine.[3]

The fast solution of these problems can be obtained using parallel algorithms and multicore system. In this, we select three numerical problems schematic presentation of it shown in figure 4.

A. Matrix multiplication:

In matrix multiplication algorithm, there is no task dependency hence thread and kernel instances parallel running reduces execution time. Matrix multiplication problem also solved using OpenMP.

![Fig 3. Execution of the Model](image)

![Fig 4. Flow chart Of OpenMP Module](image)
It will give best result by executing it sequentially rather than parallel when number of rows and columns are less, but as number of rows and columns (i.e., matrix A[1000][1000]) increases sequential algorithm’s performance decreases where role of parallel programming comes into play. In this method threading

1. int main(int argc, char *argv[])
2. Declare variables to store allocated memory.
3. Declare variables to input matrix size and variables to be used by OpenMP functions ar, ac, br, bc, nthreads, tid, chunk.
4. Declare variable to calculate the time difference between the parallelization.
5. Accept row and col.
6. allocate memory for matrix one.
7. allocate memory for matrix two.
8. allocate memory for sum matrix

\[
\begin{align*}
c & = \text{malloc}(\text{ar}) \\
\text{for } (i = 0; i < \text{bc}; i++) \\
c[i] & = \text{malloc}(\text{ar})
\end{align*}
\]
9. start the timer
   Double start = omp getw time()
10. #pragma omp parallel shared(a, b, c, nthreads, chunk) private(tid, i, j, k)
    tid = omp get thread num() if (tid == 0)nthreads = omp get num threads()
    printf Starting matrix multiplication

Distance Vector Routing

There are two classes of representative hierarchical path finding algorithms at present, i.e., the nearest E-nodes method and the optimal E-nodes method. Both methods have the pros and cons respectively. The nearest E-nodes method is simplicity but the search accuracy will be reduced and the search efficiency of optimal E-nodes method is relatively poor though the optimal solution can be obtained. The paths found by non hierarchical path finding algorithms namely flat algorithms are called flat paths and the paths found by hierarchical path finding algorithms are called hierarchical paths. A hierarchical algorithm consists of the non hierarchical algorithm and a set of rules for reasoning on a hierarchical structure. The non hierarchical algorithm provides a solution for a single level of the hierarchy and the rules state how to use such a structure, i.e., which part of the data set to consider, how the reasoning process progresses over the levels, and how to combine partial results obtained at each level into a final result.

The nearest E-nodes algorithm is based on the idea to apply the non hierarchical algorithm such as Dijkstra’s algorithm to different levels and the entrance points and exit points (nodes E) between high-level and low-level. The algorithm procedure is described as follows:

Step 1. Obtain randomly both nodes O and D in the road network.
Step 2. Determine the level of road network which O and D are located.
Step 3. If O and D are located at the high-level then go to step 4, else find the node E1 and E2 from O and D to the high-level road then obtain a partial path P1 between O and E1 as well as a partial path P2 between D and E2.
Step 4. Use nonhierarchical algorithm to find a shortest path P3 between E1 and E2 in high-level road network.
Step 5. Connect P1, P2, and P3 to form a path from O to D.
IV. CONCLUSION

The algorithms with small data set give good performance when executed by a sequentially programming. But as data set increases performance of sequential execution falls down where parallel execution is used for large data set then it gives best results than sequential execution. System is also scalable for handling large volumes of transactions.

REFERENCES


[8] Barbara Chapman, Gabriele Jost, and Ruud van der Pas “Portable Shared Memory Parallel Programming” Published by the IEEE Computer Society January 2008 (Vol. 9, no. 1), art. no. 0801-rds2008010003 1541-4922 © 2008 IEEE.