Performance Analysis of Parallel Cellular Automata Model using OpenMp and PThread Programming Constructs

Padmaja Annapureddy
School of Computing and Information Technology
REVA University, Banglore, India.
Padmaja.annapureddy@gmail.com,

Manjushree Sahana V
School of Computing and Information Technology
REVA University, Banglore, India.
manjushreesahanav@gmail.com

Shilpa Chaudhari
School of Computing and Information Technology
REVA University, Banglore, India.
shilpa.chaudhari@reva.edu.in

***Abstract* – Conway's Game-of-life cellular automaton is most well-known used approach for analyzing the performance of parallel programming constructs. The universe of the Game-of-life is a 2-dimensional array of cells wherein each cell takes two possible states, alive or dead. The state of every cell is repeatedly updated according to those of eight neighbors. A cell will be alive if exactly three neighbors are alive, or if it is alive and two neighbors are alive. This paper simulates this Game-of-life using OpenMP and PThread parallel programming construct and analyzes their performance on Linux platform. The main contribution of this paper includes parallel implementations of Game-of-life to improve overall speedup. It is observed that the performance of PThread based implementation is better than the OpenMP model.**

Keywords— OpenMP, PThreads, parallel programming, cellular automata

I. INTRODUCTION

Game-of-life is a cellular automaton devised by the British mathematician John Horton Conway in 1970. Its realization encapsulates universal computation and construction [1]. Evolution of Game of Life starts with its initial state/configuration that does not require further input. It consists of grid of cells. We represent the number of states as s=2 in our solution where the grid is n-dimensional (n=2). The evolution of the cell at discrete time, namely at time step t, is just observed over the infinite 2-dimensional orthogonal grid of square cells with t=0 as initial step. Hence it is also called as zero-player game. Each cell can be either in alive or dead state and interact with set of cells called its neighborhood (usually includes cell itself with horizontal, vertical, or diagonal adjacent eight neighbors). Four rules for transitions at each step are as follows. (1) Any live cell with under population (fewer than two live neighbors) dies. (2) Any live cell with two or three live neighbor's lives on to the next generation. (3) Any live cell with overcrowding (more than three live neighbors) dies. (4) Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction[2].

An efficient solution to this problem for performance improvement is proposed I this paper using parallel programming constructs. The proposed solution is implemented using Open Multiprocessing (OpenMP) and POSIX threads (PThreads) the parallel programming constructs in such a way that the program is partitioned into serial and parallel regions. OpenMP and PThreads are an APIs that supports multi-platform shared memory multiprocessing programming in C, C++, and Fortran, on most processor architectures and operating systems, including Solaris, AIX, HP-UX, GNU/Linux, Mac OS X, and Windows platforms[3]. OpenMP consists of a set of compiler directives, library routines, and environment variables that influence run-time behavior. Pthreads is a lower level API for generating threads and synchronization explicitly, fine-grained control over thread management (create/join/etc), mutexes[4].

The developed simulation programs for Game of Life use simple text based characters to represent the cell death and survival. The proposed parallel solutions for Game of Life are based on the OpenMP and PThreads, which are analyzed to determine an optimal solution.

II. RELATED WORKS

Open-MP is used to specify shared memory parallelism for C/C++ and Fortran programs. It can also operate on distributed memory systems with additional layer of software. The additional layer is needed to manage the memory coherency[6]. It uses the fork-join model of parallel execution. Every Open-MP program begins execution with a single thread referred as master thread. When the master thread encounters a parallel region, it forks additional worker threads. The PThreads are same as the OpenMP but only difference is OpenMP using the high level abstraction and PThread is the low level abstraction.

The fascination of Conway's Game is that, the simple rules leads to an unbelievable variety of mathematical problems, puzzles and patterns, yet at the same time it appears to exemplify emergent and self-organized behavior[3]. For example if the initial population consists of only one or two live cells, then it loses validity only one step next.

The authors of [5] recommend a comparison of serial program and parallel program. Many programs still based only on single-threaded

development, cannot make good use of CPU resources. They show that the parallel programs not only have a higher time performance relative to the serial program, but also have better CPU resources utilization.

The complex combinatorial and biological problems of the real world are solved using parallel technique in [6]. An efficient parallel computational method for the Game of life problem is solved using Open-MP constructs. The inherent flexibility of the Open-MP such as higher thread level abstraction and breakdown of serial parallel regions make it comfortable for the proposed solution to fit in the field of bio-informatics, especially when dynamic allocation and de- allocation techniques are used.

The ForeC mechanism can achieve better parallel performance than Esterel[7]. The authors uses synchronous language for concurrent safety critical systems and OpenMP-a popular desktop solution for parallel programming. They demonstrate that the worst-case execution time of ForeC programs can be estimated accurately.

The Microbial Fuel Cell (MFC) is a bio-electrochemical transducer converting waste products into electricity using microbial communities. Cellular Automaton (CA) is a uniform array of finite-state machines that update their states in discrete time depending on states of their closest neighbors by the same rule. A theoretical design of such a parallel processor by implementing CA in MFCs is proposed in [8].

The ten unison principles of living organisms in game of life given in [9] are as follows. (1) Organized body, (2) Distinctive and symmetrical structure; (3) Autonomous and energetic metabolism; (4) Excitability, sensibility, and response; (5) Modeled reproduction; (6) Homeostatic equilibrium; (7) Growth and transformation; (8) Synchronized rhythms; (9) Autonomous preservation, and (10) Patterned behavior.

III. PARALLEL PROGRAMMING MODEL FOR GAME-OF-LIFE

Each cell have its own dedicated thread to compute the cell's value in the next generation. This thread is embedded in a class that is instantiated M*N times by the driver, once for each cell in the grid. Thread synchronization is done with semaphores. The primary problem solved is coordinating all the cell threads during each generation. A cell thread cannot start computing the new cell value for the next generation until all other cells have completed their computation for the current generation. To develop this model, we used the shared memory model of ANSI-C - OpenMP and PThreads APIs for parallelization. The OpenMP and PThreads based model for the game of life solution is conducted on two different architecture systems (Dual-core and Quad-core). The parallel algorithm is developed using OpenMP and PThreads constructs considering the matrix into minor equal pieces.

The cellular automaton is represented with M*N matrix, where each cell shows the living status (alive or dead). The equal size second matrix to store the calculated next position is used. A swap between two pointers of the matrixes at the end of each round is done, in order to have next round current state. The dimensions of the table and the number of thread in the proposed solution are dynamically changed, with the default values for dimension as 90*30 and one thread. The dynamic memory is allocated using malloc() function for each cell and thread. In order to handle current situation, two grids are used - one for original status and other to store the new situation.

A. Distribution of Parallel regions

Every cell value is dependent on the current cell values in Game of Life that takes M*N grid size. This shows that the cells are dependent both on data as well as processing that are to be done on them. Assuming the system to be multithread shared memory model we have to divide the threads equally among processors. A simple way of achieving this can be done just by dividing the cells into the grids.

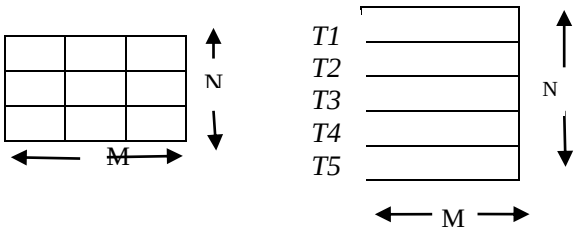


Figure 1 Game of Life that takes M*N grid size

Considering M as rows and N as columns, the grid is divided into parts from the output obtained by M*N as shown in figure 1. This gives the solution to divide the grid into cells whereas a closer look tells us what problems are faced in doing so. Though the hardware is invented to dual core or quad core, we are still using sequential methods to solve such problems. More parallelism to the required problem increases efficiency and performance of the process. The parallelism is added based on columns and sections divided by lines which are accessed when ever required by columns. This method gives better performance to the shared memory model. Problem can be solved using PThread and OpenMP for separating parallel portions. The limits are defined as follows: Each thread is identified using unique ID starting from 0. So the mathematical relationship is given as shown below.

$$ST=ID*h/NT$$

F=ST+h/NT

Where ST-Start Thread, ID-Unique Id, h-Height, NT- the number of tendril of height and height (number of rows present) of the grid.

B. Synchronization

Multithreading based solution is used in this model wherein the threads are dependents on each other. This dependency requires proper synchronization for accurate solution. In PThread based model, <pthread.h> file is used for using the thread synchronization primitives. Once the thread encounters pthread_barrier_wait(), it suspends and wait until remaining threads reaches the barrier or saturation level. The limit is decided based on the values set during initialization of the barrier. No threads move out of barrier area until all threads reaches it. These provide consistency in our data and have required control during processing.

IV. IMPLEMENTATION

The implementation divides the operation into two modes game play mode and game bench mode. The whole operation should undergo these two modes. The game play mode appears on the terminal. It consists of current state of the game and few other information. To feel the output, display of the output should be fit to the screen size. So some default values are given such as 100*50. The output size should not exceed this value. An additional feature such as time measurement is added in the game bench mode. It can be calculated using the function gettime(). This method gives no information about table status rather it can be made to statistically represent size, number of threads running and running time. User can go with default or non-default values. The default values has dimension of 90*30 where as the user can defined their own values not exceeding the limit in the initial phase.

The table 1 discusses some basic common function used as OpenMP and Pthread parallel programming constructs.

Table 1: OpenMP and Pthread parallel programming constructs

Function	Arguments	Return	Operation
Initialize_board()	Int **current, int dflag	void	If the default flag is 1, the authority table 0, otherwise randomly 0,1.
Read_file()	Int**current, chat*filename	void	Reading the original state of the file.
Copy_region()	Int**current, int**next	void	Copy of the region and in the second table.
Adjacent_to()	Int**current, int i,int j	int	Returns the number of live neighbors of the cell of the current with position(i,j).
Play()	Int**current, int**next, int start,finish	void	Applying the rules of the game, and renewal of values in the table next state(point next).
Print()	Int**current	void	Print the status of the current panel
Arg_check()	Int argc, chat*argv[]	int	Check the parameters with which the executable was called. Returns 0 on successful testing.
Print_help()	void	void	Display possible parameters such outside and interpretation.

A. Implementation using PThread

The barrier value is initialized in the main function to make all threads undergo synchronization for effective utilization of the resources. Thread is created using the input function entry_fun(). This function defines the limit of the screen as mentioned earlier, then we have 102 iterations which is for execution of 102 laps of Game of Life. The program

starts executing using game play function which consists of (curr & nxt) and limits the grids on which rules shall be applied. The function game play()n applies rules and values in the grid are refreshed which give next index. Here the control makes sure no data is altered. Following the game play comes the barrier wait() function which helps tendrils to change the values in the grid to the next state which is available. The thread with ID ==0 swaps current pointer to the pointer in the next round which becomes current round in the next generation. Suppose at this stage if call game play function it takes a print to give status of the grid in the current present round. This swapping continuous and barrier makes sure the thread would not enter the next round if it has not undergone the current round.

B. Implementation using OPENMP

The Open MP implementation is not much different from PThread. Also the option num_thread() is used to define how many threads has to be executed in the parallel region. The functionality of working is same as that of PThread. All Threads undergo barrier_wait() function to have synchronization.

V. RESULTS

A. System requirements

Two different systems (2-core and 4-core) are used for execution of the developed Pthread and OpenMP model. The specifications of the systems are shown in the Table 2.

Table 2: System Specifications

characteristics	DualCore system	Quad core system
Cpu model	Intel (R) Core(TM) 2 Duo CPU	Intel(R) Core(TM) i5-4570 CPU
Frequency	2.10 Ghz	3.20Ghz
#Cores	2	4
L2 Cache /Core	2048KB	1024KB
#Threads	2	4
RAM	3.00GB	4.00GB
System Type	32 bit OS	64bit OS

B. Analysis od Pthread Model Results

The Figure 2 to 5 shows the graphs obtained for implementation with Pthreads with different dimension. It is observed that the time taken for execution of the game of life application is reduced with respect to increase in number of threads in the model. After some specific number of threads the increase in threads does not affect the time taken for execution of the model. The number for threads can be four irrespective of dimension of the filed in game of life to achieve optimal execution time.

C. Analysis od OpenMP Model Results

The Figure 6 to 9 shows the graphs obtained for implementation with OpenMP with different dimension.

Figure 2 Number of Threads Vs time in PThreads Model with 90x30

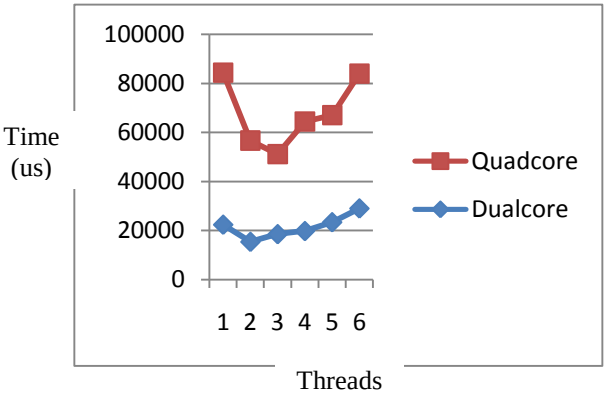


Figure 3 Number of Threads Vs time in PThreads Model with 200x50

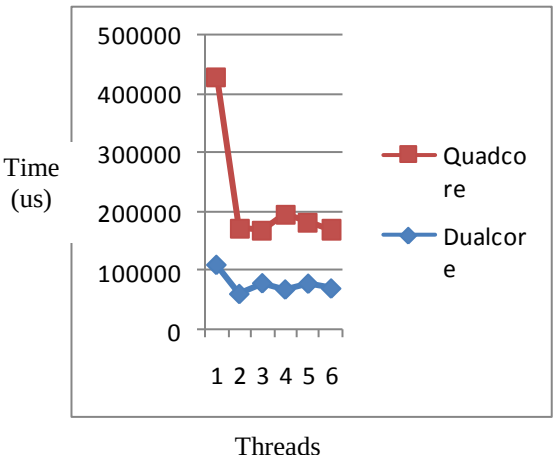


Figure 4 Number of Threads Vs time in PThreads Model with 200x500

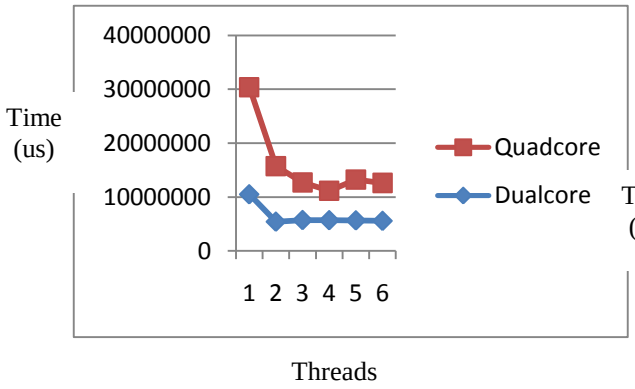


Figure 5 Number of Threads Vs time in PThreads Model with 200x200

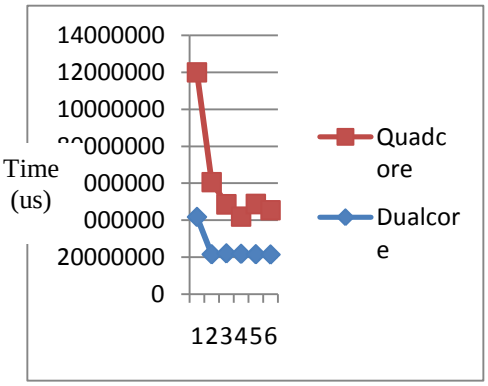


Figure 6 Number of Threads Vs time in OpenMP Model with 90x30

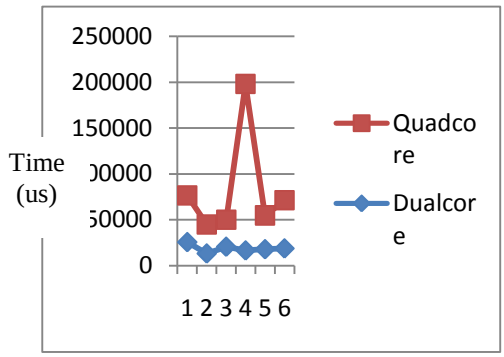


Figure 7 Number of Threads Vs time in OpenMP Model with 200x50

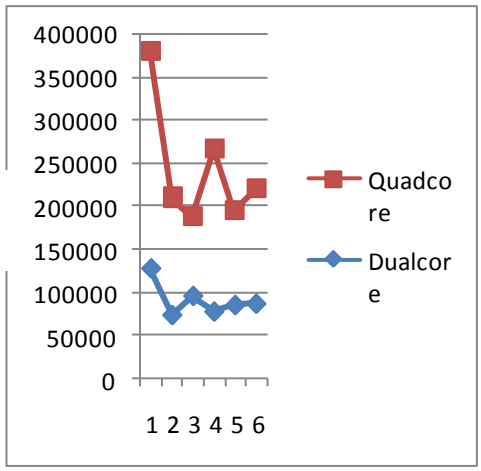


Figure 8 Number of Threads Vs time in OpenMP Model with 200x500

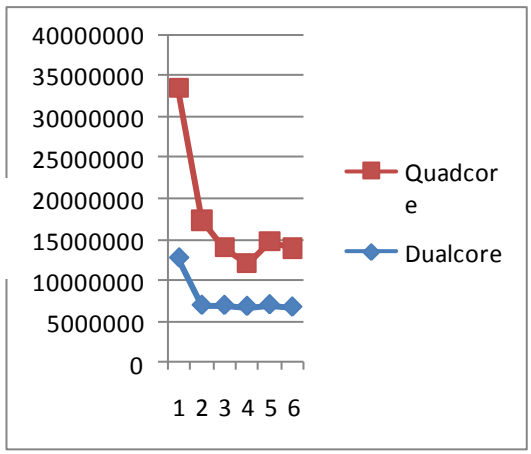
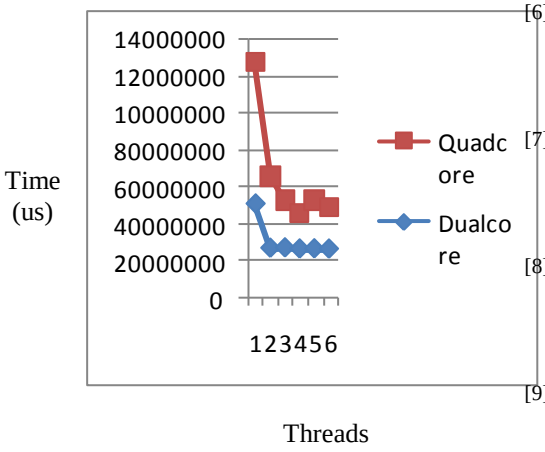


Figure 9 Number of Threads Vs time in OpenMP Model with 200x200



Similar to Pthread model, it is observed that the time taken for execution of the game of life application is reduced with respect to increase in number of threads in the model. After some specific number of threads the increase in threads does not affect the time taken for execution of the model. The number for threads can be four irrespective of dimension of the filed in game of life to achieve optimal execution time.

It is obeserved that Pthread implementation is better than that of OpenMP, as OpenMP has higher level of abstraction API's than compared to Pthread which has low level approach.

VI. Conclusion

The performance of the Game of Life problem is improved by assigning more number of threads in the parallel programs, The parallel implementation of sequential program is more powerful to utilize the software and hardware in optimal way. The proposed solution is used to run infinite number of threads within a grid. Pthread being a low level abstraction has higher and better performance than compared to OpenMP which is high level abstraction. This model can be used to teach the parallel programming concepts.

References

[1] Adamatzky A. Game of Life cellular automata. vol. 1. Springer; 2010.

[2] Hossain, Fahmida, Md Shamsujjoha, and Md Nawab Yousuf Ali. "Biological and combinatorial problems exploration using parallel and evolutionary computing." ICT and Knowledge Engineering (ICT&KE), 2016 14th International Conference on. IEEE, 2016.

[3] Fujita, Toru, Koji Nakano, and Yasuaki Ito. "Fast Simulation of Conway's Game of Life Using Bitwise Parallel Bulk Computation on a GPU." International Journal of Foundations of Computer Science 27.08 (2016): 981-1003.

[4] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

[5] Longfei Ma, Xue Chen and Zhouxiang Meng."A performance Analysis of the Game of Life based on parallel algorithm" *International Journal of Foundations of Computer Science 20.09(2012):1209.4408.*

[6] [6] Fahmida Hossain, Md. Shamsujjoha ,and Md. Nawab Yousuf Ali " Biological and combinatorial problems exploration using parallel and evolutionary computing" Fourteenth International Conference on ICT and Knowledge engineering,2016:07804095.

[7] [7] Eugene Yip, Alain Girault, Partha S. Roop and Morteza Biglari-Abhari."The ForeC synchronous deterministic parallel programming language for multicores " Institute of Electrical and Electronics Engineers(IEEE),2016:hal-01412102.

[8] [8] Michail-Antisthenis Tsompanas, Andrew Adamatzky, Georgios Ch. Sirakoulis, John Greenman, and Ioannis Ieropoulos." Towards implementation of cellular automata in Microbial fuel cells" *International Journal of Foundations of Computer Science 05.03(2017):1703.01580.*

[9] [9] LUDUS VITALLS."The game of life ten precepts and a patterned process" *International Journal of Foundations of history and philceophy of medicine, vol.XV,*2007:450-922-1-SM.