Journal of Theoretical and Applied Information Technology
31st October 2014. Vol. 68 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

IMPLEMENTATION WITH MULTITHREADING PROCESS USING EMBEDDED SYSTEM

[1]K R JAYACHITRA, [2]Dr. L C SIDDANNA GOWD

[1]Research Scholar, St.Peter's University, Avadi, Chennai, India.
[2]Professor, ECE, GRT Institute of Engineering and Technology, Tiruttani, India
Email: [1]jayamura1912000@gmail.com, [2]gouda.lcs@gmail.com

ABSTRACT

In Embedded system based applications, the evolution of multicore architectures offers many performance enhancements like speed, concurrency, real-time implementation support etc. However, design issues like critical section handling, selecting optimal number of threads, racing condition avoidance, concurrent tasks handling etc. needs to be addressed. In this work, these issues are implemented for multicore architecture using openMP tool. Barrier region limitations are removed to exploit concurrency and demonstrated applications include are, (i) array filling multitasking (ii) sorting of number, and (iii) sorting of strings. In all the above examples, the performance of multicore is enhanced compared to single core.

Keyword: *Multicore, Openmp, Racing, Multithreading, Embedded System.*

1. INTRODUCTION

Parallel programming models are required to exploit multicore architecture and eliminate performance handicapped languages [2,7]. Typical examples of multicore processor are listed in Table 1.

Table 1 Typical Processors And Reported Cores

Processor	Reported cores
Intel SCC	48
AMF ATI RV7000	10
NVIDIA Tesla C1060	30
Intel XEON	4
ARM MPCore	4

1.1 Power reduction in multiple cores

The power dissipation in single is related as $P_{single} = CV^2 f$ where F→ number of times / second the circuit is oscillated [14,15]. In a multicore, the power dissipation is reduced [10] and is given by $P_{multi} = 0.396CV^2 f$. A processor with two cores has the frequency of oscillation influenced as in Figure 1.

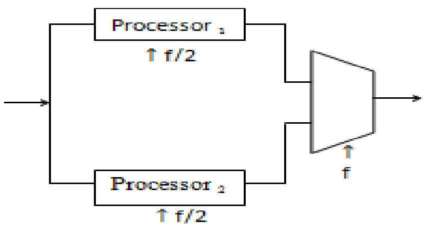

Fig. 1 Example Of Two Cores

For the dual core processor, the frequency of oscillation is only half of the total oscillation and hence, power loss is reduced [9]. The two important performance enhancement factor and parallelism (multiple tasks actually active at one time) and parallelism (multiple tasks actually active at one time) are handled effectively to avoid limiting issues like racing, redundant threads, barrier region etc. In this work, the OpenMP (an API for writing multithreaded application with a set of compiler directives and library routines) suited for parallel processing is used [3,4,5,6]. The architecture of OpenMP is given in Figure 2.

1.2 Multithread Implementation

Threads are light weight processes and share process state among multiple threads and reduce the cost of switch context [8]. Threads interact through reads/writes to a shared address space. The synchronization among threads requires a scheduler and shall determine when to run which threads [16]. The use of multithreads offer performance enhancement [11], but certain issues such as racing avoidance, critical section handling, barrier region, etc. needs to ensure yield correct results.

2. RACE CONDITION IN MULTITHREADING

In a shared address model, unintended sharing of data causes race condition and results in altering the program's outcome (i.e. data loss) since, the threads are scheduled differently. Solving race conditions involves use of synchronization to protect data

Journal of Theoretical and Applied Information Technology
31st October 2014. Vol. 68 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

conflicts (or) change how data is accessed to minimize the excessive use of synchronization [1]. In this paper, fork-join parallelism model is used to transform a sequential algorithm into a parallel one. The code to create multithreads 'N'is listed below:

```
omp_set_num_threads [n]
#progma OMP parallel
{
        int ID=omp_ge, thread _num);
}
```

The omp_set_num_threads and _get_thread_num are runtime function and used to request a certain number of threads [11,13] and returning a thread ID respectively. The first tread is int exe from 'o', and the last thread to N-1. Also, only N-1 threads are created since the Nth Parallel section can be invoked from the parent thread and a thread Pool exists to minimize cost of threads and a thread creation and destruction is eliminates for each parallel region. Barrier region refer to the waiting time due to some threads waiting for all the other threads to finish before proceeding.

3. IMPLEMENTATION OF THE WORK

3.1 Array Filling

In this case, an array is to be filled with elements both using single core single thread and multicore multithread [12,17].
The implementation code is listed in Figure 2.

```
#pragma omp parallel shared © private (I,tid)
Num_threads (NR_THREADS)
{
        tid=omp_get_thread_num();
        if (tid==0
        {
        nthreads=omp_get_num_threads();
        printf("Starting array filling with %d
hreads\n",nthreads);
        }
}
```

Fig. 2 Progmaomp master (Fork-Join parallelism model)

The threads are implemented as shown in Figure 3.

```
#progmaomp for nowait
for(i=0; i<value; i++)
{
        c[i]=i;
}

#progmaomp for nowait
For (i=value+1;i<value*2+1; i++)
{
        c[i]=1;
}
if (tid==0)
{
        St=omp_get_wtime();
}
```

Fig. 3 Threads Implementation

The meaning of different shared variable in this application is given in Table 2.

Table 2 Different Shared Variable Application

NR_THREADS	No. of multithreads
Value	size of the array i.e. the No. of Multithreads array to be handled by one thread
tolarrsize	represents the total array size
C	array of size "tolarrsize"
Tid	Thread identifier

The results are given in Figure 4 and Figure 5.

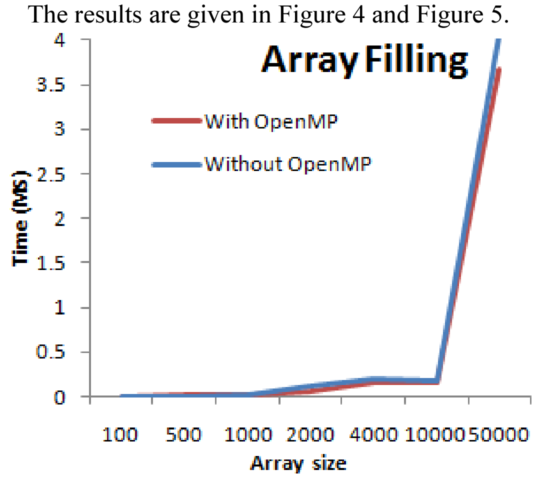

Fig. 4 Array Filling

Journal of Theoretical and Applied Information Technology
31st October 2014. Vol. 68 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

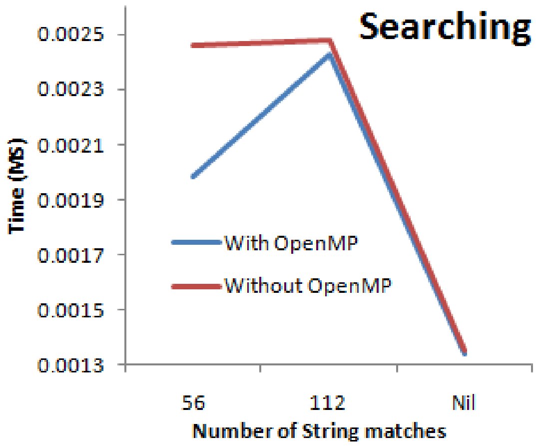

Fig. 5 Searching of Numbers and String

3.2 Sorting of Numbers

In this application, numbers are to be sorted in either ascending or descending order using multithread implementation. For comparison, a non multithread implementation is also considered. The obtained results are shown in Figure xx.

3.2.1 Multithread Implementation of Sorting Numbers in Ascending Order

```
if (choice==1)
 {
  ifp = fopen (readfile, "r");
  st=omp_get_wtime();
  #pragma omp shared(a,tsortno) private(i,count)
   {
    if ((ifp) == NULL)
    {
      printf("Error in logfile1.txt file \n");
     }
         fscanf (ifp,"%d", &i);

    while (!feof (ifp))
    {
     a[count]=i;
     count=count+1;

   //printf("count value is %d\n",count) ;
     //printf ("%d\n ", i);
     fscanf (ifp,"%d", &i);
     }
  fclose (ifp);
  #pragma omp nowait
   {
  ascend(a,tsortno);
    }
}
ed=omp_get_wtime();
printf("\nTotal time taken for to ascend by Core:
%f\n",ed-st);
fprintf(logfile_1,"\nTotal time taken for to ascend
by Core is :");
fprintf(logfile_1,"%f",ed-st);
fprintf(logfile_1," for ");
fprintf(logfile_1,"%ld",tsortno);
fprintf(logfile_1," input ");
}
```

3.2.2 Non-multithread Implementation of Sorting Numbers in Ascending Order

```
if(choice==1)
 {
  ifp = fopen (readfile, "r");
  st=omp_get_wtime();
     {
     if ((ifp) == NULL)
     {
      printf("Error in logfile1.txt file \n");
     }

     fscanf (ifp, "%d", &i);

     while (!feof (ifp))
     {
      a[count]=i;
      count=count+1;


      fscanf (ifp, "%d", &i);
     }
  fclose (ifp);
      {
  ascend(a,tsortno);
     }
}
 ed=omp_get_wtime();
printf("\nTotal time taken for to ascend by CPU:
%f\n",ed-st);
fprintf(logfile_1,"\nTotal time taken for to ascend
by Cpu is :");
fprintf(logfile_1,"%f",ed-st);
fprintf(logfile_1," for ");
fprintf(logfile_1,"%ld",tsortno);
fprintf(logfile_1," input ");
}
```

4. RESULTS AND DISCUSSION

In this work, results described the consumed times to sort numbers using OpenMP multithreading in ascending and descending order respectively as shown in Figure 6 and Figure 7.

Journal of Theoretical and Applied Information Technology
31st October 2014. Vol. 68 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

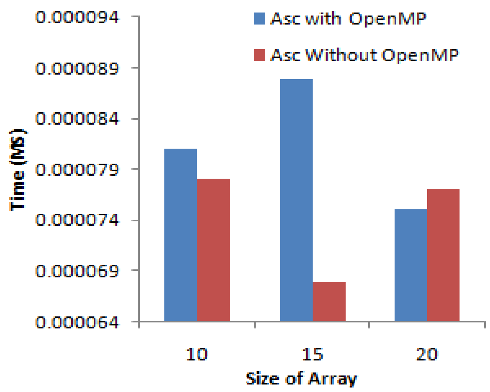

Fig. 6 Time Taken To Sort Number Using Openmp Multithread (Ascending)

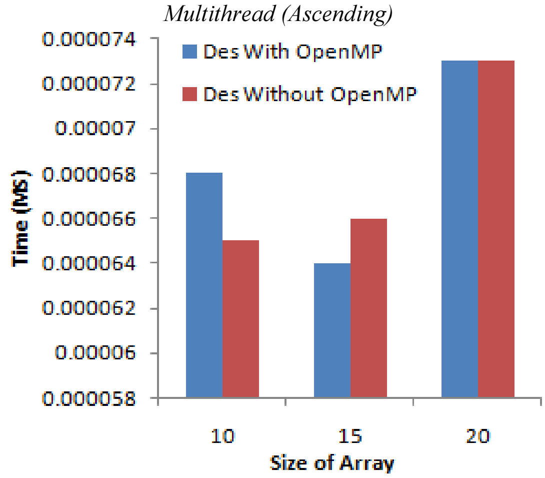

Fig. 7 Time Taken To Sort Number Using Openmp Multithread Descending)

Figure 8 and Figure 9 described the consumed time to sort alphabets using multithread in ascending and descending order respectively.

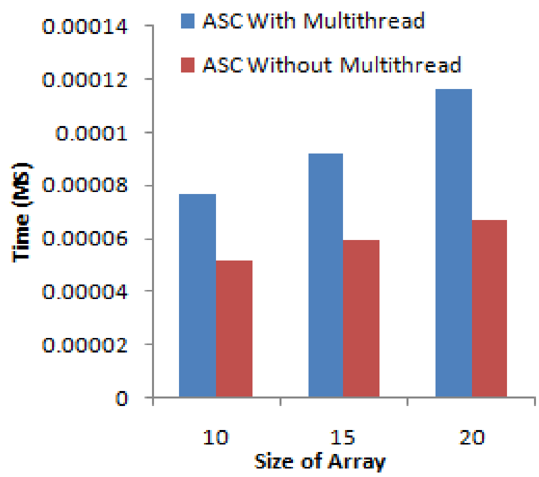

Fig. 8 Time Taken To Sort Alphabets Using Multithread In Ascending Order

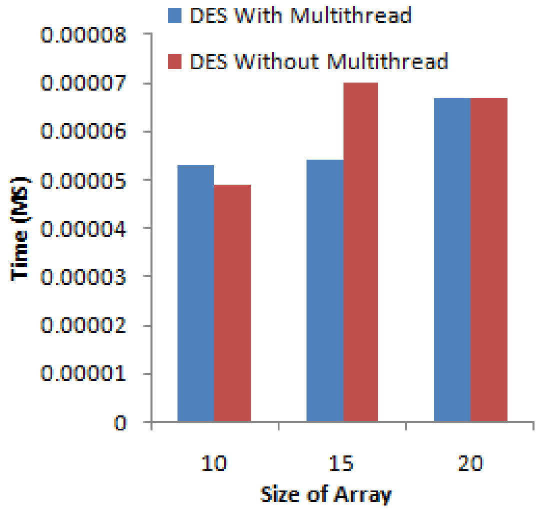

Fig. 9 Time Taken To Sort Alphabets Using Multithread In Descending Order

5. CONCLUSION

This work is concentrated on openMP for multithreading for ascending and descending process to calculate time consumption to sort number and alphabets. The comparison study of ascending and descending order between number (using openMP) and alphabet (using multithreading) is shown the difference to increase speed and reduce power loss in network.

ACKNOWLEDGMENT

I acknowledge the help and facility offered by M/s MicroLogic Systems to carryout the multithreading process and embedded system studies with hardware environment.

REFERENCES

[1] Ryo Mizutani and Kenji Ohmori, "A Design and Implementation Method for Embedded Systems Using Communicating Sequential Processes with an Event-Driven and Multi-Thread Processor", International Conference on Cyberworlds, Darmstadt, Germany, September 25 -September 27, Germany, 2012.
[2] Umar Hamid, Haroon Shahzad and Muhammad Irfan, "Parallel Implementation of 1-D Complex FFT Using Multithreading and Multi-Core Systems", International Journal of Computer and Communication Engineering, Vol. 2, No. 2, March 2013.
[3] Spiros N. Agathos, Vassilios V. Dimakopoulos, Aggelos Mourelis, Alexandros Papadogiannakis, "Deploying OpenMP on an Embedded Multicore Accelerator", IEEE Conference, 2013.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[4] V. V. Dimakopoulos, E. Leontiadis and G. Tzoumas, "A portable C compiler for OpenMP V.2.0", in Proceeding on EWOMP, 5th European Workshop on OpenMP, Aachen, Germany, pp. 5–11, September 2003.
[5] M. Sato, Y. Nakajima, Y. Ojima and Y. Hotta, "OpenMP Implementation and Performance on Embedded Renesas M32R Chip Multiprocessor", in Proceeding of 6th European Workshop on OpenMP (EWOMP'04), pp. 37–42, 2004.
[6] W.C. Jeun and S. Ha, "Effective OpenMP Implementation and Translation For Multiprocessor System-On-Chip without Using OS", in Proceeding of ASP-DAC '07, 12th Asia and South Pacific Design Automation Conference, IEEE Computer Society, Yokohama, Japan, pp. 44–49, 2007.
[7] B. Chapman, L. Huang, E. Biscondi, E. Stotzer, A. Shrivastava and A. Gatherer, "Implementing OpenMP on a high performance embedded multicore MPSoC," in Proceeding of IPDPS '09, IEEE International Symposium on Parallel & Distributed Processing, Rome, Italy, pp. 1–8, May 2009.
[8] Greg Hoover, Forrest Brewer, Timothy Sherwood, "A Case Study of Multi-Threading in the Embedded Space", CASES'06, Seoul, Korea, October 23–25, 2006.
[9] A. Snavely, L. Carter, J. Boisseau, A. Majumdar, K. S. Gatlin, N. Mitchell, J. Feo, and B. Koblenz, "Multi-processor performance on the tera mta", In ACM/IEEE Conference on Supercomputing (CDROM), pp. 1–8, 1998.
[10] Bernhard H.C. SPUTH, Oliver FAUST and Alastair R. ALLEN, "Portable CSP Based Design for Embedded Multi-Core Systems", Communicating Process Architectures, Peter Welch, Jon Kerridge, and Fred Barnes, IOS Press, 2006.
[11] Shah Bhatti, James Carlson, Hui Dai, Jing Deng, Jeff Rose, Anmol Sheth, Brian Shucker, Charles Gruenwald, Adam Torgerson and Richard Han, "MANTIS OS: An Embedded Multithreaded Operating System for Wireless Micro Sensor Platforms", Mobile Networks and Applications, Vol. 10, No. 4, pp. 563–579, 2005.
[12] R. Kumar, V. Zyuban and D.M. Tullsen, "Interconnections in Multicore Architectures: Understanding Mechanisms, Overheads, and Scaling", Proceeding in International Symposium, on Computer Architecture (ISCA05), 2005.
[13] M. Lee, B. Whitney and N. Copty, "Performance and Scalability of OpenMP Programs on the Sun FireTM E25K Throughput Computing Server", WOMPAT, pp. 19-28, 2004.
[14] Y. Zhang, M. Burcea, V. Cheng, R. Ho and M. Voss, "An Adaptive OpenMP Loop Scheduler fro Hyperthreded SMPs", in Proceeding of International Conference on Parallel and Distributed Systems (PDCS-2004), San Francisco, CA, September 2004.
[15] Simon Schliecker, Mircea Negrean, Rolf Ernst, "Reliable Performance Analysis of a Multicore Multithreaded System-On-Chip", in Proceeding CODES+ISSS, ACM, 2008.
[16] P. Crowley and J.L. Baer, "Worst-Case Execution Time Estimation for Hardware-assisted Multithreaded Processors", in Proceeding 2nd Workshop on Network Processors, 2003.
[17] M.A. Kinsy, M. Pellauer and S. Devadas, "Heracles: Fully Synthesizable Parameterized MIPS-Based Multicore System", in 21st International Conference on Field Programmable Logic and Applications. IEEE Conference, September 2011.