Journal of Computer Science 8 (1): 68-75, 2012
ISSN 1549-3636

Effective Utilization of Multicore Processor for Unified Threat Management Functions

Sudhakar Gummadi and Radhakrishnan Shanmugasundaram
Department of Computer Science and Engineering
Arulmigu Kalasalingam College of Engineering
Anand Nagar, Krishnankoil-626190
Tamil Nadu, India

Abstract: Problem statement: Multicore and multithreaded CPUs have become the new approach for increase in the performance of the processor based systems. Numerous applications benefit from use of multiple cores. Unified threat management is one such application that has multiple functions to be implemented at high speeds. Increasing performance of the system by knowing the nature of the functionality and effective utilization of multiple processors for each of the functions warrants detailed experimentation. In this study, some of the functions of Unified Threat Management were implemented using multiple processors for each of the functions. **Approach:** This evaluation was conducted on SunfireT1000 server having Sun UltraSPARC T1 multicore processor. OpenMP parallelization methods were used for scheduling the logical CPUs for the parallelized application. **Results:** Execution time for some of the UTM functions implemented was analyzed to arrive at an effective allocation and parallelization methodology that is dependent on the hardware and the workload. **Conclusion/Recommendations:** Based on the analysis, the type of parallelization method for the implemented UTM functions are suggested.

Key words: Logical CPUs, OpenMP, packet processing, performance analysis, URL filtering, spam filtering, Unified Threat Management (UTM), parallelization method

INTRODUCTION

Network security is one of the most critical issues facing today's internet. Traditionally, for an enterprise, a firewall was used as a first line of defense. With more complicated network environment and mature attack means, the traditional firewall strategy cannot meet the demands of security. For the combined protection against complex and blended threats, multiple security features are integrated into a unified security architecture that results in a Unified Threat Management (UTM) appliance. Unified Threat Management products integrate multiple security features, such as firewall, VPN, intrusion detection and prevention systems, antivirus, spam blocking, URL filtering, content filtering and network monitoring into a single secure appliance (Qi *et al*., 2007). The design challenges of implementing a UTM are: the performance of multiple functions, cost effectiveness, scalability and co-existence with third party software.

With the increase in the network speeds and also the increase in the security threats, the implementation of high performance UTM is essential. Multi-core technology offers high performance, scalability and energy efficiency. UTM processing can be decomposed into parallel activities such as per packet, per flow or type of processing.

A multicore processor (or Chip-level Multiprocessor, CMP) combines two or more independent cores into a single Integrated Circuit (IC) and performs multiprocessing (Lee and Shakaff, 2008). Multicore architecture has become more and more widely used in intensive computing applications as well as in computer networking systems. The amount of improvement in performance by the use of a multicore processor is dependent on the software algorithms and their implementation. Scheduling of parallel activities on the multicore

Corresponding Author: Sudhakar Gummadi, Department of Computer Science and Engineering, Arulmigu Kalasalingam College of Engineering, Anand Nagar, Krishnankoil-626 126, Virudhunagar District, Tamil Nadu, India Tell: +91-4563-289129

processor is very vital to improve the performance of the system. The underlying hardware of the multicore processor has to be effectively used to obtain the optimum performance of the system.

The per chip core counts are increasing significantly. For example, Oracle's SPARC T3 processor features up to 16 cores and 128 threads on a single chip with integrated logic for 1GbE networking and cryptographic coprocessor engines. octeon II CN6880 of Cavium Networks is a 32 core processor with over 85 application acceleration engines that provides high-performance, high throughput solution for intelligent networking applications (Cavium Networks, 2011).

Programming of the multithreaded multicore processor needs a thorough understanding of the hardware and the effective use of the Application Program Interface (API) for parallel programming. The task partitioning and CPU allocation in multicore processors is done based on the application requirement and the time taken for execution of these tasks. OpenMP API is one of the parallel programming models used to exploit the available parallelism of multicore processors.

In a multicore environment, CPUs or a set of CPUs can be assigned to a particular process. Proper performance indicators need to be used for simulation, testing and realization of multicore implementations. Parallelization of UTM functions is considered for generating the load and analyzing the performance of the system.

Multithreaded multicore processor architecture: The UltraSPARC T1 is a chip multicore/multi-threads processor that contains 8 cores and each of the SPARC cores has 4 hardware threads. A single pipeline processes instructions from four threads and completes one instruction in each cycle. All together, the chip handles 32 hardware threads and is addressed as 32 logical CPUs (Weaver, 2008; Leon *et al*., 2006).

Each SPARC core has a 16 KB, 4-way associative, 32B line size of Level 1 instruction cache (I Cache), 8 KB, 4-way associative, 16B line size of Data Cache (D Cache), 64-entry fully associative instruction TLB (Translation Look aside Buffer) and 64-entry fully associative data TLB that are shared by the four hardware threads. The eight SPARC cores are connected through a crossbar to an on-chip unified 3 MB, 4-way associative L2 cache (64B lines). The L2 cache connects to 4 on-chip DRAM controllers, which directly interface to DRAM interface.

Figure 1 show a simplified block diagram of the multicore processor wherein each core has separate L1 instruction cache and L1 data cache. All the cores share the common L2 cache with external shared memory. Each hardware thread of UltraPSARC T1 processor has a unique set of resources in support of its execution. The per-thread resources include registers, a portion of I-fetch data path, store buffer and miss buffer. Multiple threads within the same SPARC core share a set of common resources in support of their execution. The shared resources include the pipeline registers and data-path, caches, Translation Lookaside Buffers (TLB) and execution unit of the SPARC core pipeline due to which the performance of a thread is also affected by other threads running on the same core.

UltraSPARC T1 processor has one Modular Arithmetic Unit (MAU) per core that supports modular multiplication and exponentiation. The hardware thread that initiated the MAU stalls for the duration of the operation, but the other three threads on the core can progress normally.

Open MP: The OpenMP Application Program Interface is a portable, parallel programming model for shared memory multithreaded architectures (Sun Microsystems, 2009; Chapman *et al*., 2009). OpenMP specification version 3.0 introduces a new feature called tasking. By using the tasking feature, applications can be parallelized where units of work are generated dynamically, as recursive structures or while loops. The task directive defines the code associated with the task and its data environment. The task construct can be placed anywhere in the program and whenever a thread encounters a task construct, a new task is generated.

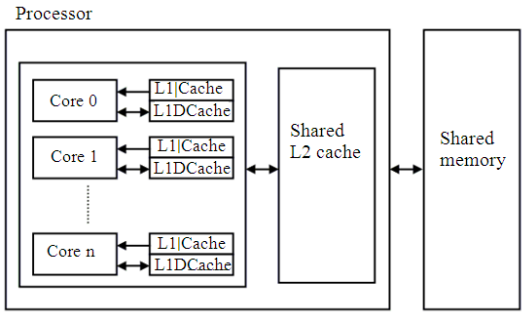

Fig. 1: Simplified block diagram of multicore processor with external shared memory

Ayguade *et al*. (2009) have evaluated the performance of the runtime prototype with several applications using OpenMP tasking feature and have measured the performance in terms of the speedup for different number of CPUs and have proved that OpenMP task implementation can achieve very promising speedups when compared to other established models like OpenMP nested, task queues and Cilk.

Packet processing and parallelization: Packet processing functions have to be done in real time at network line rates. When packet processing functions are implemented in multicore processor based system, the packet processing rate is dependent on the number of threads and cores used for processing and the effective utilization of the hardware resources by the application programs. Packet processing workload is characterized by a large number of simple tasks and large amounts of input/output operations. Typical packet processing applications include forwarding of packets, packet classification, packet scheduling, packet statistics and monitoring and security application. Gigabit data rates that have to be handled by network systems generate significant performance demands (Weng and Wolf, 2009). The overall packet processing tasks are split into three different tasks, namely, receiving, processing and transmitting. Time critical functions take place in the processing task and the nature and extent of parallelism for the processing task and the processor architecture determines the system performance (Sleit *et al*., 2009).

The processing demands on the packet processing system are affected by computational characteristics of all tasks in the system; and by network traffic that exercises the processing system. To derive an optimal allocation of tasks to processing resources at runtime, these factors have to be quantified and considered in the mapping process (Wu and Wolf, 2008).

Weng and Wolf (2009) presented the analytic performance model that could be applied for understanding tradeoffs in the network processor design space to determine suitable network processor topologies and multithreading configurations.

The design challenges of implementing a UTM are: the performance of multiple functions, cost effectiveness, scalability and co-existence with third-party software. Multiple functions of UTM are to be performed simultaneously at required performance levels. Hui (2008) discussed the concept of defining the policies for the flow based on the classification and the implementation of the different policies for the first packet and subsequent packets of the same flow. Classification, rule based policy enforcement and signature based policy enforcement are some of the common processes for UTM. Pattern matching is used in content filtering, URL filtering, spam filtering and intrusion detection functions.

In this study, we present the performance analysis of UTM functions by varying the assignment of CPUs of Sun Microsystems UltraSPARC T1 processor. OpenMP is used for parallelizing the code for execution on the hardware threads referred as CPUs. We also proposed the type of parallelization based on the UTM function for better throughput.

MATERIALS AND METHODS

The performance evaluation is done on SunFire T1000 server having Sun Microsystems UltraSPARC T1 processor. Sun Studio12 Update 1 Integrated Development Environment (IDE) on Solaris 10 Operating System was used to develop the programs in C language and to test the programs. OpenMP parallelizing features are used for implementing parallelism within each process. Libpcap Application Program Interface (API) is used for reading the packets from the physical interface or writing the packets to the physical interface. Furthermore, POSIX.1b Real-time Extension Library is used for message passing, process scheduling and timer options. System V message queues are used for queuing the packets between various stages.

Implementation of UTM requires multiple independent processes. Data is communicated between these processes by use of message queues. Semaphores are used wherever synchronization is necessary. Fork function is used to create the required independent processes and the processor sets are bound to each of these processes. PSet_assign function is used for the assignment of the CPU to a particular processor set. One processor set each is bound to each of the processes. Figure 2 shows the conceptual diagram of allocation of processor sets for different UTM functions. Seven processor sets are created and CPUs are assigned to each of these processor sets.

As shown in Fig. 2, one CPU is assigned to processor set $PSet_R$ for the receive_packets process that receives the packets from the physical interface and enquires the packets to the appropriate queues based on the classification. Another CPU is assigned to processor set $PSet_T$ for transmit_packets process that dequeues the packets from the queues and transmits the packets to the physical interface. For implementation of the UTM functions like the VPN, spam filtering, URL filtering, intrusion detection and virus detection, five processor sets $PSet_1$ to $PSet_5$ are bound to five different independent processes.

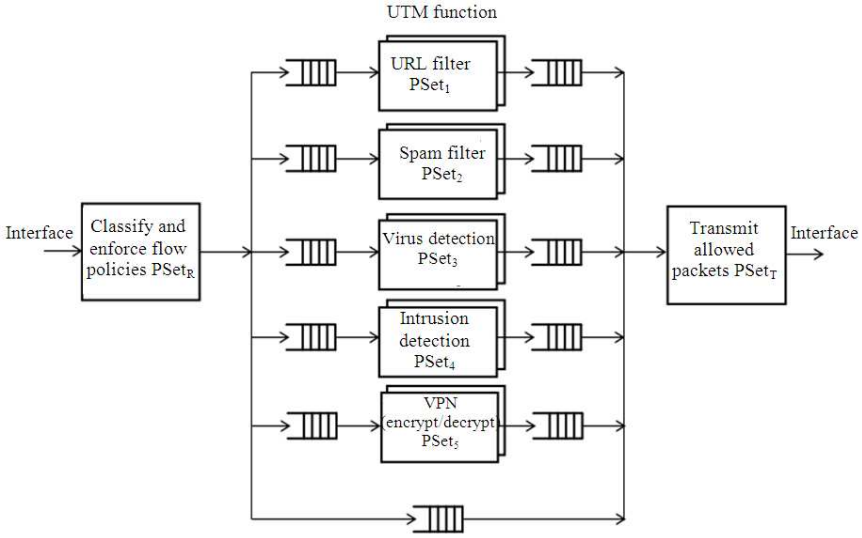

Fig. 2: Allocation of processor sets for UTM functionality

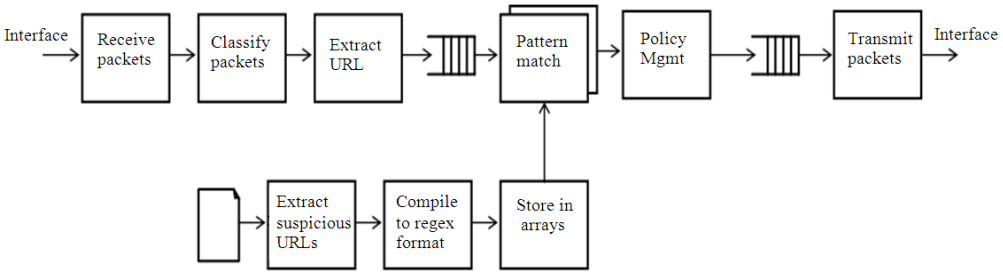

Fig. 3: Block diagram for URL filtering

Incoming packets are first classified and the stream index number is allocated for the packets. All packets belonging to the same stream will have the same stream index value. Policies are defined for the first packet of the stream index based on the five-tuple classification. Subsequent packets of the same stream would have specific policies enforced. Based on these policies to be enforced, the packets are appropriately enqueued for specific UTM function processing or forwarded directly to the transmit processor set.

For parallelization and execution by multiple CPUs assigned to a processor set, OpenMP parallelizing directives are used.

URL filtering: For URL filtering, URL that is extracted from the ftp or the http header field is compared with the list of suspicious sites. Block diagram for URL filtering is shown in Fig. 3. During the initialization phase, each of the 80,000 suspicious URLs are extracted from the file, converted to regex format and stored in an array. The number of arrays for storage of URLs of suspicious sites is equal to the number of CPUs used for parallelization of URL processing. For testing the performance of URL filtering, a maximum of four CPUs are assigned to the processor set. Regex functions are used for matching each of the URLs extracted from the packet with the URLs stored in the arrays. If there is a match, the packet is tagged for being dropped; else the packet is tagged for being transmitted. Packets to be transmitted are queued in the transmit queue for being transmitted by the transmit_packet process.

Parallelization using OpenMP sections feature is used for URL filtering. The number of threads and the number of logical CPUs are set to be equal to the number of OpenMP sections. URLs are extracted from the packet headers and queued for URL filtering. Each CPU in the parallel region compares the URL that is extracted from the queue with each of the patterns stored in the corresponding array concurrently.

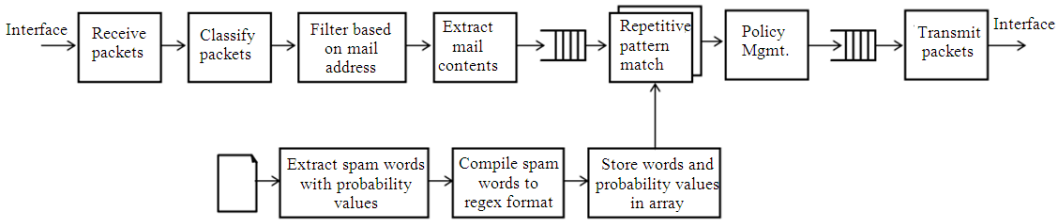

Fig. 4: Block diagram for spam filtering

Once there is a match in any one of the sections, further comparison in all the sections are aborted and the execution time recorded for the matching process.

Spam filtering: Spam filtering is applicable to the packets that carry the email content and use SMTP protocol. The block diagram for implementation of spam filtering is shown in Fig. 4. During initialization, spam words with probability value for each of the words are extracted from the file, the words compiled to regex format and then the words along with its probability value are stored in the array. These words are used for checking the email content and for identifying the mail as spam. The packets that are received from the interface are classified and after filtering, the Email header and content are extracted from the packet and enqueued using the message queues. The spam filtering process is done in the subsequent stage based on which decision is taken to term the mail as spam. In the next stage the mail is transmitted.

As the time taken for the spam filtering process, i.e., repetitive pattern matching of all the spam words and accumulation of the probability value of the matched patterns and terming the mail as spam, takes relatively longer time that receiving and transmitting the packets, the pattern matching process is parallelized. The number of CPUs used for parallelization is defined in the initialization phase.

Experimentation is done using flow based parallelization and pattern based parallelization. In flow based parallelization, each packet is handled independently by each CPU of the processor set assigned for spam filtering. CPU dequeues the message, implements the logic for spam filtering and then enqueues the message. OpenMP *tasking* feature is used for implementing this flow based parallelization.

Experimentation is also done using pattern based parallelization wherein the same message is simultaneously handled for pattern matching by all the CPUs of the processor set assigned for spam filtering. Parallelization using OpenMP *for loop* feature is used for implementing this pattern based spam filtering. For each iteration of the parallelized loop, a word along with the probability value is extracted from the array and the word is compared with the email message and header. In each iteration, repetitive pattern matching for each word is done till the end of the content and for each match the probability value is accumulated. The summed up probability value of all the words is used to check with the threshold value to declare the mail as spam.

There are three scheduling methods that are used for implementation of OpenMP *for loop* parallelization for spam filtering, namely, static scheduling, dynamic scheduling and guided scheduling:

- In static scheduling, the number of iterations of the for loop are equally distributed to the number of CPUs before the parallelization starts
- In dynamic scheduling, iterations equal to the chunk size defined will be allocated by the scheduler to each CPU for parallelization. Once the CPU completes the assigned iterations, it picks up another chunk value for processing. This process continues till the total number of iterations is complete
- In guided scheduling, for each of the CPU available for parallelization, based on the algorithm, the number of iterations greater than the chunk size is allocated initially. Progressively the number of iterations reduces to the chunk size. The advantage of guided scheduling is that the number of times the CPU has to be scheduled reduces compared to the dynamic scheduling

For both the pattern matching methods, experimentation is done by changing the number of CPUs for the parallelized code that determines the mail as spam or not and the total execution time is measured. Study was also done on the performance of the parallelized region by changing the scheduling parameters when using the OpenMP *for loop* feature.

RESULTS

Performance of URL filtering: The total execution time is measured by taking 80 of the 80,000 URLs as inputs for URL filtering. These 80 URLs are taken such that they occur at uniform intervals spread over 80,000 URL records.

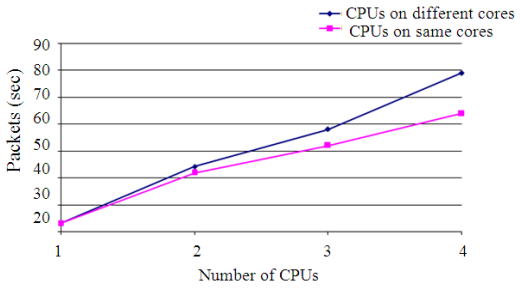

Fig. 5: Packet processing rate for URL filtering with varying number of CPUs

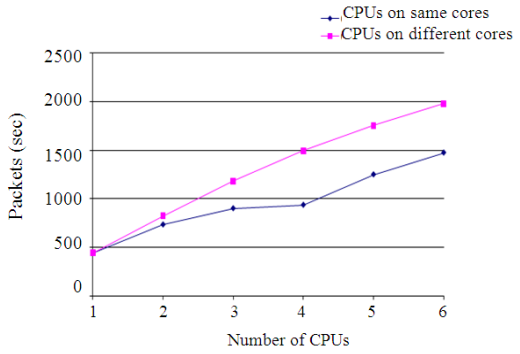

Fig. 6: Packet processing rate for flow-based spam filtering with varying number of CPUs

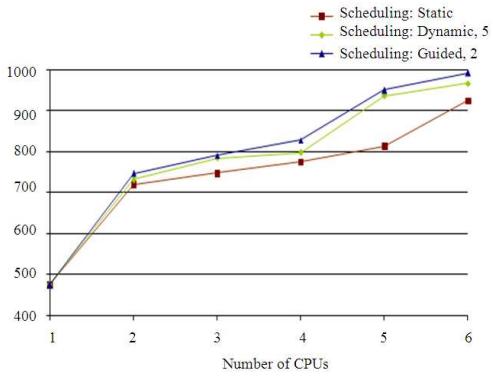

Fig. 7: Packet processing rate for pattern based spam filtering with vary number of CPUs

Packet processing rate is computed from the measured execution time and is as shown in Fig. 5 for varying number of CPUs for the parallelized process of dequeuing, URL filtering and enqueuing for 80 URLs. The execution time is measured for two sets of readings, first by allocating the CPUs on different cores and then by allocating the CPU on the same core. OpenMP *sections* feature is used for parallelization.

Performance of flow based spam filtering: Packets of 1000 bytes each are simulated as email header and content and queued for spam mail filtering. The words that occur in the spam mail with repetitions of some such words are present in the packets. These 80 packets are dequeued, checked for spam and later enqueued with the status information by the CPUs assigned to processor set $PSet_2$ in the parallelized mode.

Figure 6 shows the throughput for flow based spam filtering while processing 80 packets of 1000 bytes each with different number of CPUs in the parallelized region using OpenMP *tasking* feature.

Performance of pattern based spam filtering: Packet processing rate measured by processing 80 packets of 1000 bytes each by varying the number of CPUs in the parallelized region for detection of spam mail is shown in Fig. 7. OpenMP *for loop* is used for the parallelization. The plot shows the packet processing rate for static scheduling, dynamic scheduling and guided scheduling. For dynamic scheduling, the chunk size is set as 5 and for guided, the chunk size is set as 2.

DISCUSSION

For URL filtering, as the number of CPUs increase, there is reduction in the execution time. The relative improvement of the execution time using CPUs of the same core vis-à-vis CPUs of different core is mentioned in Table 1 for different number of CPUs.

URL filtering is normally time consuming and the time taken for detecting the pattern match, if any, depends on the location of the pattern in the array. URL filtering is generally done for only the first packet having the new session index. Policy is generally defined such that subsequent packets of the same session index are not forwarded for URL filtering.

Flow-based spam filtering uses OpenMP *tasking* feature wherein each CPU dequeues one packet, implements spam filtering and enqueues the packet with the status word. Due to the repetitive pattern matching requirement for each word, for the given test conditions, 1,472 packets could be processed per second when six CPUs are allocated for spam filtering.

OpenMP *for loop* feature is used for pattern-based spam filtering wherein all the CPUs handle only one packet in the parallelized zone but would be implementing the repetitive pattern matching with different patterns concurrently. The time taken for each process is non-uniform due the number of matches that would occur for each of the pattern. In static scheduling, based on the initial distribution of the number of iterations for each CPU, due to the non-uniformity of time for processing, there is significant wait time for all the CPUs to complete the processing that adds to the overall processing time.

Table 1: Relative improvement of execution time using CPUs belonging to the same core vis-à-vis CPUs belonging to the different cores

No. of CPUs	Improvement factor
2	1.05
3	1.12
4	1.24

Table 2: Relative improvement using dynamic scheduling methods with reference to static scheduling for pattern-based spam filtering

	Scheduling type	
No. of CPUs	Dynamic, 5	Guided, 2
1	1.00	1.00
2	1.02	1.04
3	1.05	1.06
4	1.03	1.07
5	1.15	1.17
6	1.05	1.07

Table 3: Relative performance of spam filter using flow based approach and pattern based approach

No. of CPUs	Packets per second for flow-based spam filter	Packets per second for pattern-based spam filter
1	440.92	475.70
2	733.68	747.58
3	900.39	791.56
4	935.67	828.92
5	1248.05	951.85
6	1471.67	992.97

This delay is overcome by using the two other scheduling methods, namely, dynamic and guided scheduling. Table 2 shows the relative improvement in the number of packets processed per second using the dynamic and guided scheduling methods. Load balancing is done effectively by these dynamic and guided scheduling methods which show an improvement in performance as compared to static scheduling.

For the same set of conditions, the comparison is done for spam filtering using flow-based and pattern-based approach. Table 3 shows the relative performance of flow-based spam filtering as compared to the pattern-based spam filtering. As the number of CPUs increase, the number of packets processed per second in the flow-based spam filtering is greater as compared to the pattern-based. This is due to the dynamic scheduling overhead for allocation of the number of iterations to each CPU and the overheads in handling multiple CPUs in OpenMP *for loop* parallelization.

CONCLUSION

Various functions of UTM were studied and the URL filtering and spam filtering were implemented using the CPUs of the multicore processor. Different OpenMP parallelization features were tried for URL and spam filtering. OpenMP *sections* feature is appropriate for the URL filtering. For spam filtering, flow-based and pattern-based parallelization methods were tried. Parallelization using runtime scheduling methods like the dynamic and guided were used for pattern-based spam filtering. However results show that flow-based spam filtering performed better than pattern-based spam filtering. Future study will be done in implementing other UTM functions and synchronization of UTM functions.

ACKNOWLEDGMENTS

The researchers acknowledge TIFAC-CORE in Network Engineering (established under the Mission REACH program of Department of Science and Technology, Govt. of India) for providing necessary facilities for working on this project.

REFERENCES

Ayguade, E., N. Copty, A. Duran, J. Hoeflinger and Y. Lin *et al*., 2009. The design of OpenMP tasks. IEEE Trans. Parallel Distrib. Syst., 20: 404-418. DOI: 10.1109/TPDS.2008.105

Chapman, B., L. Huang, E. Biscondi, E. Stotzer and A. Shrivastava *et al*., 2009. Implementing OpenMP on a high performance embedded multicore MPSoC. Proceedings of the IEEE International Symposium on Parallel and Distributed Processing, May 23-29, IEEE Xplore Press, Rome, pp: 1-8. DOI: 10.1109/IPDPS.2009.5161107

Cavium Networks, 2011, Octeon II CN68XX Multi-Core MIPS64 Processors Product Brief.

Hui, M., 2008. Designing UTM with a Multi-Core Processor.

Lee, W.F. and A.Y.M. Shakaff, 2008. Implementing a Large Data Bus VLIW Microprocessor. Am. J. Applied Sci., 5: 1528-1534. DOI: 10.3844/ajassp.2008.1528.1534

Leon, A.S., B. Langley and J.L. Shin, 2006. The UltraSPARC T1 processor: CMT Reliability. Proceeding of the IEEE Custom Integrated Circuits Conference, Sept.10-13, IEEE Xplore Press, San Jose, pp: 555-562. DOI: 10.1109/CICC.2006.320989

Qi, Y., B. Yang, B. Xu and J. Li, 2007. Towards system-level optimization for high performance unified threat management. Proceedings of the 3rd International Conference on Networking and Services, Jun. 19-25, IEEE Xplore Press, Athens, pp: 7-7. DOI: 10.1109/ICNS.2007.126

Sleit, A., W. AlMobaideen, M. Qatawneh and H. Saadeh, 2009. Efficient processing for binary submatrix matching. Am. J. Applied Sci., 6: 78-88. DOI: 10.3844/ajassp.2009.78.88

Sun Microsystems, 2009. Sun Studio 12 Update 1: OpenMP API User's Guide. 1st Edn., Sun Microsystems, Inc, USA., pp: 73.

Weaver, D., 2008. OpenSPARC Internals. 1st Edn., Sun Microsystems, Inc, USA., ISBN: 978-0-557-01974-8, pp: 369.

Weng, N. and T. Wolf, 2009. Analytic modeling of network processors for parallel workload mapping. ACM Trans. Embedded Comput. Syst., 8: 1-29. DOI:10.1145/1509288.1509290

Wu, Q. and T. Wolf, 2008. On runtime management in multi-core packet processing systems. Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, (ANCS' 08), ACM, New York, USA., pp: 69-78. DOI: 10.1145/1477942.1477953