TELKOMNIKA Indonesian Journal of Electrical Engineering
Vol.12, No.3, March 2014, pp. 2227 ~ 2235
DOI: http://dx.doi.org/10.11591/telkomnika.v12i3.4491

Architecture and Task Scheduling of Video Streaming on Multi-core Platform

Jun Li*[1,2], Hong Ni[1], Lingfang Wang[1], Jun Chen[1]
[1]National Network New Media Engineering Research Center, Institute of Acoustics, Chinese Academy of Sciences, Beijing, China 100190
[2]University of Chinese Academy of Sciences, Beijing, China 100190
*Corresponding author, e-mail: lij@dsp.ac.cn

Abstract

Compared with traditional video streaming server, streaming on multi-core platform has many advantages: flexible and configurable on the number of executing core according to system requirements; fault-tolerant; and fitting well to future process technologies, more cores will be available in advanced process technologies, meanwhile the complexity per core does not increase. In this paper, we focused on the video streaming issues on multi-core processor, including architecture and task scheduling. We proposed a pipeline-parallel hybrid multi-core architecture and service migration based task scheduling strategy on multi-core processor to improve the efficiency of video streaming and increase the number of concurrence. We implemented the task scheduling algorithm with proposed architecture, and provided evidences of 48% outperformance based on Cavium OCTEON CN5860 multi-core processor than full parallel architecture, meanwhile, request success rate higher than REM algorithm.

Keywords*: multi-core platform, pipeline-parallel hybrid architecture, service migration, task scheduling*

1. Introduction

VoD system provides service to terminal users by streaming traditional video contents. It is usually composed of center task scheduler and a number of multimedia streaming servers. The center scheduler is responsible for task assignment and management, user interactive operation and service status maintenance, which includes session parameters, video contents distribution and playing statistical information. Meanwhile, streaming server takes change of multimedia streaming with instruction from scheduler. As a main component, video streaming server always is the key module in current interactive on demand industry standards, like ISA (Interactive Service Architecture) [1], NGOD (Next Generation on Demand) [2] and so on. Both in ISA and NGOD, video streaming system is designed and used to push multimedia to STB (Set-Top-Box). VSS (Video streaming system) works with others modules like SM (Session Manager), VRM (Video stream Resource Management) STB (Set-Top-Box) and so on. Terminal users send their VCR (Video Cassette Recording) signaling to VSS by STB, and then VSS parses the signaling, replaces current streaming with corresponding content.

For a video streaming application, like VSS, the server may service thousands of users concurrently, which brings in huge pressure to VoD system. Meanwhile, computer hardware manufacturers have moved decisively to multi-core and are currently experimenting with increasingly advanced many-core architectures. And more and more network acceleration devices use multi-core processor to achieve much higher targets. But on multi-core platform, how to integrate those cores and scheduler modules on each core is still under research, which is also the most significant to figure out. In the long term, writing portable, efficient and correct parallel programs targeting multi-core architectures must become no more challenging than writing the same ones for sequential computers. To date, however, most applications running on multi-core machines do not exploit fully the potential of these platforms. In this paper, based on multi-core platform, we proposed a pipeline-parallel hybrid streaming server multi-core architecture to address problems like concurrence, stability and so on. There existed several tools and libraries available for constructing streaming applications, but many of them are oriented to coarse grain computations, such as StreamIt [3], Brook [4], and CUDA [5]. Some other tools, as TBB (Threading Building Block) [6], provide explicit mechanisms for parallel

Received August 22, 2013; Revised September 26, 2013; Accepted October 17, 2013

paradigm, while some others, as openMP [7] and Cilk [8] mainly offers mechanisms for data parallelism and Divide&Conquer computations. These mechanisms can also be exploited to implement video streaming system, as we shall compare our method with them in Section 3. Some other efforts also have been made to study streaming architecture on multi-core platform. As in [9] proposed a design of a flexible dataflow architecture aimed at addressing many of the shortcomings of existing systems including a unified execution model for both demand driven and event driven models; it designed a resource scheduler that can automatically make decisions on how to allocate computing resources; and support for more general streaming data structures which include unstructured elements. As in [10] introduced and discussed FastFlow, a programming framework specifically targeting cache-coherent shared-memory multi-cores. FastFlow is implemented as a stack of C++ template libraries. The lowest layer of FastFlow provides very efficient lock-free and memory-free synchronization base mechanisms. Since video streaming application can be constructed based on FastFlow and some other multi-core tools, but all of those methods did not optimize. In this paper, we proposed pipeline-parallel hybrid system architecture, by analyzing and making full use of features of video streaming application, to promote system concurrency performance.

Video streaming servers usually works like a cluster, which has to be load banlancing and fault tolerant. When the scheduler receives a new request from user, it has to allocate the request to a video server with some strategy which is used for load balancing. In traditional task scheduling algorithm, the scheduler migrate request to another server when a server has reached its service capacity, this method is called last-minute migration [11]. In REM algorithm, Yingqing Zhan [12] proposed double thresholds, *min_th* and *max_th*, for service migration. When the load of current video server is larger than *max_th*, new request must be migrated out because of that current video server is in hearvy load. The request will be migrated out with a kind of possibility, which is the function of system load, when load of current video server is between in *min_th* and *max_th*. New request will be accepted by video server if its load is lower than *min_th*, since it has enough service capacity. Based on pipeline-parallel hybrid architecture on multi-core platform, we introduced the service migration based task scheduling strategy. And implemented them on Cavium OCTEON CN5860 [13] multi-core processor.

2. Video Streaming on Multi-core Platform

2.1 Pipeline-parallel Hybrid Architecture

SMP (Symmetrical Multi-Processing) systems are tightly coupled multiprocessor systems with a pool of homogeneous processors running independently, each processor can executing the same program or different programs and operating on different data and with capability of sharing common resources (memory, I/O device, interrupt system and so on) and connected using a system bus or a crossbar. In this paper, we will construct a pipeline-parallel hybrid video streaming architecture based on SMP platform.

Data streaming procedure in VoD system can be considered as a data transmission unit under signaling control. There may be thousands of users require movies on demand si multaneously, video streaming server should parse the signaling, allocate session and streaming resources, obtain content from CDN or local disk caches and reply all of those requests in a very short time, such as less than 500ms. With the objective of video streaming server, we partition the system into two parts: *control-plane* and *data-plane*. C*ontrol-plane* responsable for control related operations, like task construct, signaling parsing, and resources allocation and soon. And *data-plane* takes charges of data related operations, like data obtaining, caching and transmission. In control-plane, each signaling is processed in pipeline style. Under pressure from large concurrence, data-plane is not easily to perform well. With multi-core processor, more cores can be easily configured to run data-plane programs according to the system load situation.

Almost all existed VSS systems, such as ISA and NGOD, have used RTSP protocol to transfer signaling between video streaming server and other modules like STB, Session Manager and so on. After the scheduler receiving a signaling from STB, it cannot do any scheduling until parameters parsed from signaling. Figure 1 gives the parsing time of a single RTSP message, and total 100 test times with zero system loads. RTSP message parsing time under different system load is shown in Figure 2(i3-2310M CPU @2.1GHz, 512M memory).

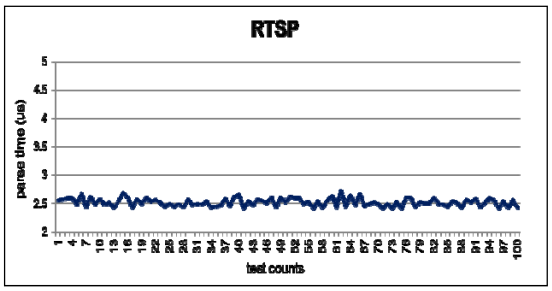


Figure 1. Time Used for Parsing a RTSP Message

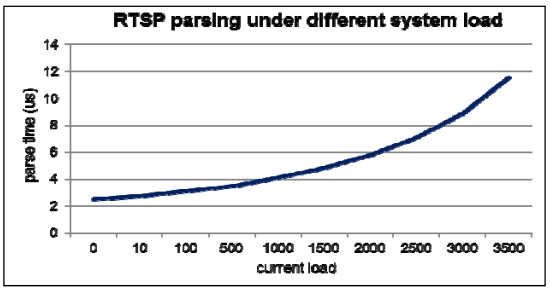
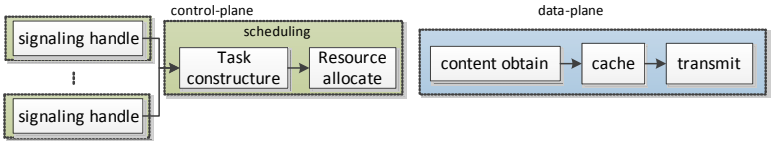


Figure 2. RTSP Message Parsing Time on Different System Load

RTSP message parsing time increases with the growth of system load. The first RTSP protocol content parsing costs 2.52 microseconds with zero system load, and when system load rising to 2500, a normal RTSP request parsing costs almost 7 microseconds. Assuming that video streaming system requires a RTSP request or response parsing time less than 5 microseconds to guarantee user experience, it needs another core to process RTSP protocol parsing when the number of concurrent user rise up to 1600.

In this paper, we assign appropriate number of core to each part of video streaming system. Multi-core architecture is given in Figure 3. Figure 3(a) and Figure 3(b) show details of control-plane and data-plane respectively. In control-plane, cores can be configured to process signaling flexibly. After RTSP signaling parsed, scheduling section do task construct or task scheduling according to the parameters encapsulated in signaling, and data-planes do responses to those schedules by stopping streaming or changing contents operations. Data obtained by data-plane will be cached for anti-jitter purpose, and then, transmitted to STB through NICs (Network Interface Card). Figure 4 gives a full parallel architecture of video streaming system. With the advantages of SMP system, each core on the multi-core platform runs the same program. For a video streaming system, it includes streaming task establishing, RTSP message parsing, resources allocating, content obtaining and data transmission. Each core can process video streaming task independently. In this paper, we will make the comparison between full parallel architecture with pipeline-parallel hybrid architecture.



(a) control-plane procedure (b) data-plane procedure

Figure 3. Detail of Pipleline-parallel Hybrid Architecture

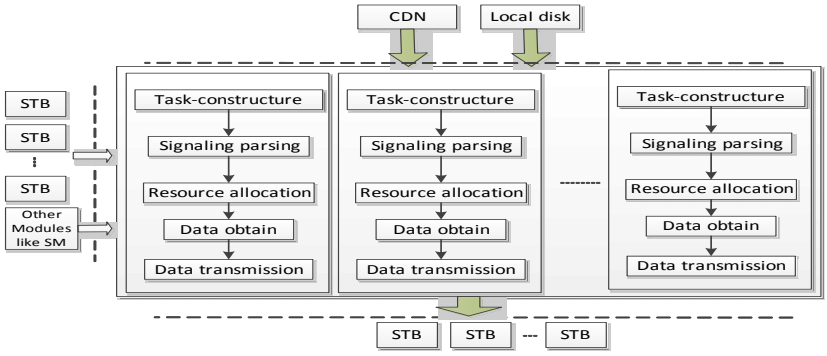


Figure 4. Full Parallel Architecture for Video Streaming with Multi-core Platform

```
task execution procedure
{   if(receive new signaling)
    {       if(is SETUP)
            {   //extract parameter from signaling
                signaling_handle(signaling);
//establish video streaming task by allocating
resources
                streaming_task = task_construct();
            //add streaming_task to task_list on current
core
                task_add(streaming_task);
            }else{   streaming task parameter update;
    }
    //iterator every task in task_list, doing streaming
job
            for each task in task_list
            {   apply_data;//from CDN or local disk;
                if data recved,
                add data to task_data_cache;
                sending_data, using funnel method to
ensure stability of rate;
            }
    }}
```

(a) control-plane execution procedure

```
while(1)
{   if(receive new data transmission task)
    {       create new data transmission task;
            add new task to data transmission task list;
            apply data from CDN or local disk;
    }
    if(receive data task parameter update)
    { update task parameter;}
    if(data received)
            add data to task_data_cache;
    iterator( task in data transmission task list)
    {       sending data to STB using funnel method;
            update task parameter;
    }
}
```

(b) data-plane execution procedure

Figure 5. Each Execute Procedure of Hybrid Architecture

Figure 5(a) and Figure 5(b) provide the pseudo-code of video streaming with pipeline-parallel hybrid architecture.

2.2. Service Migration Based Task Scheduling

Streaming server always has a kind of storage ability, but limited. So its service capacity is restricted. Based on pipeline-parallel hybrid architecture on multi-core platform, we introduced the service migration based task scheduling strategy, which is used for system load balancing and video server fault tolerant. Table 1 is the summary of key variables.

Table 1. Key Variables

symbol	description
M	Total number of content
N	Number of streaming server
L	Service capacity of a streaming server
R(j)	Request for content *j*
l_i	Current load of server *i*
$\overline{L}$	Average load of all streaming server
L_{th}	Service load threshold of service migration
H	Content distribution matrix
Λ	Streaming server connection matrix
p	Migration possibility
μ	Request success rate
T_i	Service delay of request *i*
τ	Handle time per request on streaming server
λ	Length of migration path
Γ	Maximum of migration path length
$\omega(i)$	Set of migration path which starts from server *i*

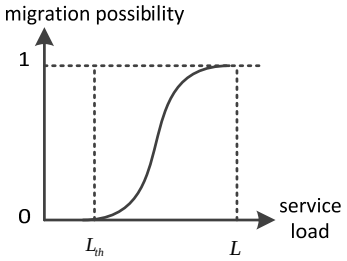


Figure 6. Releationship between Migration Possibility and Service Load in Our Apporach

When the scheduler receives a new request, it decides which video server the request will be assigned to based on current system service load. If loads of the servers that contains requested content are lager than L_{th}, service migrates out with the possibility of p_i, and path length λ ($\lambda < \Gamma$) is limited. Calculation of p_i is shown in Equation (1).

$$p_i = \begin{cases} 0 & l_i < L_{th} \\ \dfrac{1}{1 + B(l_i - L_{th})^{-n}} & L_{th} < l_i < L \end{cases} \quad (1)$$

Where *B* and *n* are system related parameters, we will discusse later.

When load of current video server $l_i < L_{th}$, it is means that current server is in light load, and can continue to service. If $l_i > L_{th}$, it indicates that current server is serving many request. In this case, service migration is needed for system load balance. Since migration not only affects the requests in service, but also increases the service delay of new request. We migrate request out with a possibility when $l_i > L_{th}$ by using modified sigmoid curve. From Figure 6, it is obvious that the p_i is much litter when load is less than L_{th}, and p_i grows quickly if $l_i > L_{th}$, which means that migration happens with much higher possibility at high load. If p_i is grater than generated random p, service migration happens. Long migration path implies higher service delay. We limit migration path length to ensure the quality of service. If path length is longer than the longest tolerant length Γ, new request will be rejected without service migration.

Grater *n* is, much faster the improved sigmoid curve trends to 1. In our approach, *n* is a function of migration path length, and it decrease rapdly with λ grows up. The negative exponential function has been used to deliver the releationship between *n* and λ, shown in Equation (2).

$$n = B \times e^{-\lambda} \quad (2)$$

Because of migration path length directly affects user experience in service; migration path length should be limited in practice to guarantee quanlity of service.

For the calculation of migration path, every possiable path can be computed before system beginning service under the situation that the multimedia content distribution and video server connection are fixed. Video server is taken as a node, servers containg the same content have a path connected each other, which shapes a graph. We assuming that the system is stable, that is content distribution and server connection never change after system is running. By using Dijkstra algorithm [14], we can get the shortest path to all others nodes in the connection graph, and recursive operate for each node. During service migration, we only need to find the optimal path in existed set without recalculation.

Based on pipeline-paralle hybrid architecture, when the scheduler receives a new request $R(j)$, it will run the following steps to dispatch the request.

Step 1: The scheduler decides whether service migration is needed for $R(j)$ according current content distribution matrix *H* and each server load l_i:

a. If S_i contains content j and $l_i < L_{th}$, then S_i will service $R(j)$, update load of S_i, goto step 6;
b. If each servers S_i which contains content j has $l_i > L_{th}$:
   a) If there is at least on path in set $\omega(i)$ in which the load of the node that after S_i is less than *L*, then computer every possiable path length and migration possibility p_i according formula 1 and 2;
   b) else goto step 5;

Step 2: The scheduler generates a random p, $0 \le p \le 1$. If $p_i \ge p$, then goto step 3, else goto step 4.

Step 3: The scheduler send migration message to each server in migration path, and service migration implemented along the path while $R(j)$ is hold at scheduler.

Step 4: S_i begin service new request. Update load status of each server, goto step 6.

Step 5: Reject $R(j)$.

Step 6: The scheduler waiting for next new request.

In our algorithm, the optimal path should be found before each migration. Since the new request in hold in scheduler, the migration path must be able to minisize service delay. We define variance of load balancing σ as formula 3 in following:

$$\sigma = \frac{\sum_{i=1}^{N} (l_i - \overline{L})^2}{N} \tag{3}$$

Where, $\overline{L} = \frac{1}{N}\sum_{i=1}^{N} l_i$, is the average load of all video servers. When $\frac{l_1}{L} \approx \frac{l_2}{L} \approx \cdots \approx \frac{l_N}{L}$ in a determinated VoD system, σ is minimum, which means system is in a good load balancing.

In this paper, linearity weighted aggregation method is used to poise migration path length and variance of load balancing, as shown in formula 4.

$$mp = \alpha \times \sigma + (1-\alpha) \times \lambda \tag{4}$$

In which λ is the length of each possiable migration path, and $\lambda < \Gamma$. The path with smallest mp will be chosed for migration.

3. Results and Analysis

3.1. Experiments of Pipeline-parallel Hybird Architecture

We take Cavium OCTEON CN5860 [13] series multi-core processor as our target platform. Cavium OCTEON CN5860 multi-core processor has 16 cores, with 800MHz per core. It has a 10GB SPF+ port, and its memory can be configured as 4GB or 8GB. In order to test the performance of the proposed architecture, we implement pipeline-parallel hybrid architectures using C, and compared it with full parallel architecture using openMP [7].

Figure 7 displays the inter-core communication overhead between control-plane and data-plane in pipeline-parallel hybrid architecture. Based on Cavium OCTEON CN5860 multi-core processor, an interactive operation among different core cost increases with growth of the load on the core. Where the concurrency rises up to 3500, inter-core communication cost is almost three times of light load.

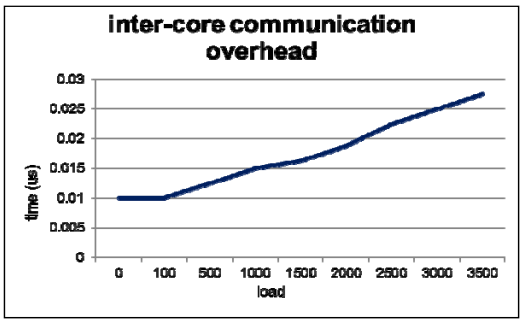


Figure 7. Inter-core Communication Overhead

The performance of control-plane and data-plane in pipeline-parallel hybrid architecture is shown in Figure 8. For data-plane, each core can handle 230 data transmission task, with bitrate of 3.75Mbps per each task. And the concurrent performance of control-plane on a single core up to 2300 streaming tasks. So, the situation of one core executing control-plane can work with 10 cores running data-plane, and the total throughput of video streaming reaches to 10Gbps, the top capacity of the platform. Figure 9 shows performance comparison between pipeline-parallel hybrid architecture and full parallel architecture, which implemented by using openMP, and it is clear that the prior method outperformance 48% than the later one.

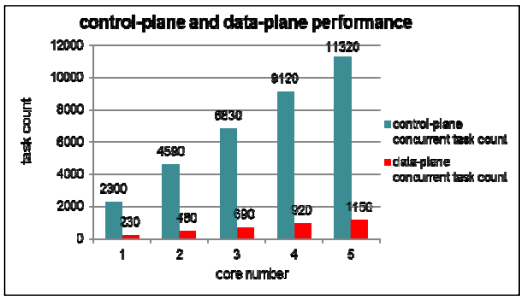


Figure 8. Task Number per Core Using openMP

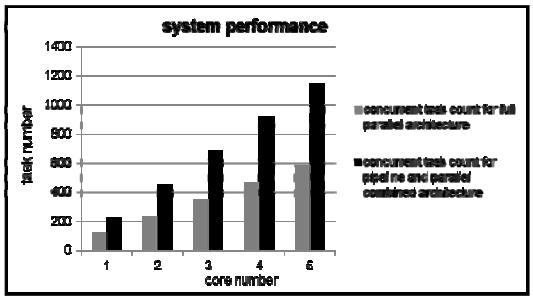


Figure 9. System Concurrent Performance Pared with Full Parallel Architecture

3.2. Task Scheduling Experiments

We implemented the proposed task scheduling alogrithm on pipeline-parallel hybird architecture. In experiments, we simulated 140 video contents and 12 video servers. Each server has 20 content storage capacity and service capacity is 120 concurrent requests. The user request pattern is a Poisson process with an arrival rate of ξ request/minute, where ξ varies from 36 to 44. The access possibility of each of 140 video contents is modeled by Zipf-like distribution [15] with $\theta = 0.27$.Besides, we set σ=0.5, which means that load balancing and service delay have the same weight in migration path selection. At the same time, after a lot of testing, *B* is set to 2500, in this case, the improved sigmoid curve changes most close to real VoD system. In order to prove the effectiveness of service migration based task scheduling strategy, we compared it with REM algorithm, Figure 10 shows the request success rate in the simulation period for service migration based task scheduling method and REM, and the prior performance better than REM algorithm. Figure 11 demostrates differnecy of the service delay between our proposed task scheduling strategy and REM. Since the precomputation of migration path, service migration based task scheduling algorithm has much lower service delay and with smaller delay jitter.

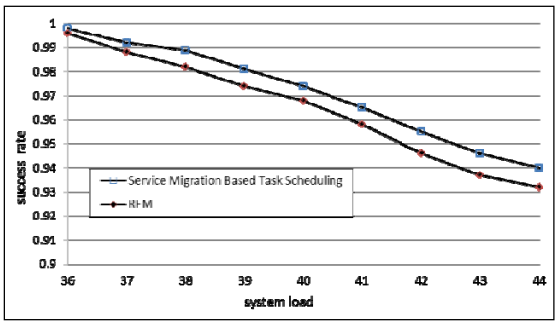


Figure 10. Success Rate as a Function of System Load

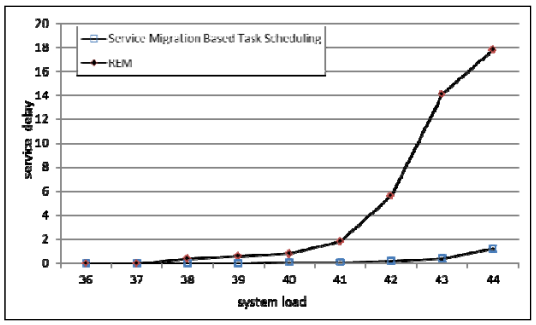


Figure 11. Service Delay as a Function of System Load

4. Conclusion

The widespread use of multi-core processor is pushing explicitly parallel high-level programming models to the forefront. In this paper, we proposed a multi-core architecture for video streaming system combined pipeline with concurrent modules. One of the main issues of getting an even distribution of computation across processors is dealt in an integrated fission and partitioning step that breaks up computation units just enough to span the available processors. The issue of signaling control overhead is overcome by an intelligent assignment, which overlap all communication with computation. Total video streaming task is split into control-plane and data-plane. High performance requires much more core resources; both control-plane and data-plane can be allocated more cores to satisfy system requirements. We compared the performance of our architecture with full parallel ones, and the experiment results show that hybrid architecture gains much higher concurrence than the compared one. Meanwhile, the proposed service migration based task scheduling algorithm achieved higher request success rate and smaller service delay jitter than REM.

Acknowledgements

We thank the team leader, my supervisor and other members for their continuous help in discussion and suggestion on the task scheduling strategy in this paper. We are indebted to members of the National Network New Media Engineering Research Center for their contributions to this work. This work is supported by National High-tech R&D Program of China (2012AA011703), National Key Technology R&D Program of China (2012BAH02B01) and the Key Program of Chinese Academy of Sciences (KGZD-EW-103-4, KGZD-EW-103-2).

References

[1] Ellis, Leslie. *Inside Time Warners's Interactive-Services Architecture*. Time Warners. Report number: 22. 2001.
[2] Comcast Corp. 2.0. NGOD Overall Architecture. Philadelphia. Comcast Corp. 2006.
[3] Thies W, Karczmarek M, Amarasinghe S. *StreamIt: A language for streaming applications.* 11th Conference on Compiler Construction. London. 2002: 179-196.
[4] Buck I, Foley T, Horn D. Brook for GPUs: stream computing on graphics hardware. *ACM Transactions on Graphics (TOG). ACM*. 2004; 23(3): 777-786.
[5] David Kirk. *Nvidia cuda software and gpu parallel computing architecture*. In Proc. of the 6th Intl. symposium on Memory management (ISM). New York. 2007: 103-104.
[6] Pheatt C. Intel® threading building blocks. Journal of Computing Sciences in Colleges. 2008; 23(4): 298-298.
[7] Chapman, Barbara, Gabriele Jost, Ruud Van Der Pas. Using OpenMP: portable shared memory parallel programming. Cambridge: The MIT Press. 2008.
[8] Cole C, Herlihy M. Snapshots and software transactional memory. *Science of Computer Programming*. 2005; 58(3): 310-324.
[9] Vo HT, Osmari DK, Summa B. *Streaming-Enabled Parallel Dataflow Architecture for Multicore Systems. Computer Graphics Forum*. Blackwell Publishing Ltd. 2010; 29(3): 1073-1082.

[10] Aldinucci M, Danelutto M, Kilpatrick P. Fastflow: high-level and efficient streaming on multi-core. Programming Multi-core and Many-core Computing Systems. *Parallel and Distributed Computing*. 2012.
[11] Mundur P, Simon R, Sood AK. End-to-end analysis of distributed video-on-demand systems. *Multimedia, IEEE Transactions on*. 2004; 6(1): 129-141.
[12] Zhao Yingqing, Kuo CCJ. *Video server scheduling using random early request migration*. Multimedia systems. 2005; 10(4): 302-316.
[13] Cavium OCTEON CN58XX. http://www.cavium.com/OCTEON-Plus_CN58XX.html. (2013).
[14] Dijkstra EW. A note on two problems in connexion with graphs. Numerische mathematic. 1959; 1(1): 269-271.
[15] Jiang Q, Tan CH, Phang CW. Understanding Chinese online users and their visits to websites: Application of Zipf's law. *International Journal of Information Management*. 2013; 33(5): 752-763.