Efficient CPU Core Usage and Balanced Bandwidth Distribution using Smart Adaptive Arbitration

M. Nishat Akhtar and Junita Mohamad-Saleh*

School of Electrical and Electronics Engineering, Universiti Sains Malaysia, 14300 Nibong Tebal, Penang, Malaysia; nishat_akhtar2000@yahoo.com, jms@usm.my

Abstract
Software industry has been revolutionized due to the introduction of multi-core chips and high performance computing. The current trend shows that number of cores per chip will double every alternate year without any modification in the processor clock speed. Sophisticated parallel computation needs to resolve the access of bus in an optimum manner by keeping in mind the parameter of efficiency. In any multi-core system, an arbiter is the one which receives numerous bus access requests which various processors (masters) generates. This article proposes a new strategy of arbitration known as Smart Adaptive Arbitration (SAA) to enhance the CPU cores usage to enable parallel computing along with balanced bus bandwidth distribution. The designed SAA is for heterogeneous masters which compute with respect to different behavior of data traffic to attain task parallelism. Latest benchmark program has been used to evaluate the performance of SAA. The results shows that SAA stands better if compared with other arbitration techniques as it tries to enable high degree of task parallelism and a balanced bandwidth distribution to the masters requesting bus access. SAA has a potential to be a promising arbitration strategy for solving future on-chip resource necessities.

Keywords: Arbiter, Bandwidth Distribution, CPU Utilization, Multi-Core, Smart Adaptive Arbiter, Task Parallelism

1. Introduction
Parallel computing is coming out to be an enhanced paradigm of programming to gain balanced multiproccessing to uncover odd parallelism using an effective manner. For any system with multi-processors, task parallelization can be accomplished if every computing device is meant to perform distinct thread execution for any comparable or a diverse data. Uniform parallel processing has turned into a functional alternative for calculating in the realm of implanted systems where innovation is mixed with complicated chips that fuse numerous computing device devoted for particular computing needs. With a specific end goal to understand complex System-on-Chip (SoC) with protected innovation based techniques, correspondence design assumes a noteworthy part. In any SoC, arbitration algorithm assumes a noteworthy part with a specific end goal to tackle bus contention. Fairness is an essential parameter that assumes an exceptionally pivotal part between different arbiters to resolve the problem of bus access. The execution of computing machines with numerous processors depends more on productive correspondence amongst them and the fair time share to compute their associated processes, as opposed to an immaculate speed of the processor. It is to be noted that for any data exchange on the bus the arbiter is the one which comes into action at first. Therefore an arbiter is at the critical path of bus-based computing and should be outlined with extraordinary care. A productive bus access plan is required to give fine-grained control of the correspondence data transfer capacity dispensed to individual processor and maintain a strategic distance from starvation of lower priority transactions. Refining task parallelism is still a challenge in the world of parallel computing. Nonetheless, languages and tools such as MPI, OpenStream, Java, OpenMP, Jasmine, Hadoop and so forth could just make an advancement of applications slightly straightforward. With respect to symmetric...
multiprocessing, the design of internal bus is a crucial part as it connects other resources using shared channel. As a result, data transfer capacity of bus turns into an overwhelming obstruction due to improper bandwidth distribution. To keep up the bus data transfer capacity in an effective way, the procedure with respect to memory arbitration cannot get ignored as it is one of a fundamental element of simultaneous computing. These days with an extensive arbitration on SoC, the correspondence engineering ought to be sufficiently reasonable to give out better efficiency to a variety of masters as per traffic behaviour, since several masters connected to the bus may generate different demands at any time. Subsequently, an arbiter assumes a critical part to choose which master ought to be given access to bus at first place. Henceforth, this paper introduces a SAA technique, outlined in a manner that it is highly compatible system by keeping efficient throughput and less starvation rate between the diverse masters and accomplishing task parallelism along with balanced bus bandwidth distribution. Remaining manuscript is arranged as follows; Section 2 gives some brief review of related works carried out by several researchers. In section 3, an idea of masters composed as per the traffic behaviour is shown. Section 4 expounds on proposed SAA method. At last, section 5 and section 6 presents the results, discussion and conclusion accordingly.

2. Related Previous Work

In recent years, numerous scientists concentrated upon creating multi-layered arbitration algorithm to reduce system latency and to accomplish reasonable bandwidth allotment. Very frequently utilized arbitration algorithm is Static Fixed Priority (SFP) algorithm, where master upon bus is allocated a definite priority value. The master accomplishing most astounding preference dependably takes the bus access with a definite bandwidth. It is to be noted that SFP is to be executed using non-preemptive way. It is also to be noted that for any preemptive execution, a continuous low priority data exchange with a master is ended promptly by not getting finished if and only if a demand for bus usage is requested from a master having higher preference. However for non-preemptive execution, a continuous low preference data exchange from any master is permitted to finish prior the access of bus is given to a high preferable master. SFP algorithm is easy to execute and can give high output by guaranteeing crucial data exchange, for example, among processor, this algorithm must get implemented precisely, because it can prompt to starvation of low preference masters, which may not at all have the capacity to possess the bus only if there are regular bus access by higher priority masters. Wiseman and Feitelson\(^1\) implemented SFP algorithm using, a strict gang scheduling in a parpar cluster of eight computational node in parallel and measured CPU usage rate which got turned up to 20%. Nonetheless, on a single node, SFP scheme can’t be implemented in parallel. Time Division Multiple Access (TDMA) arbitration algorithm can ensure a definite high bandwidth distribution of bus to various masters having high data exchange necessities and furthermore guarantees that low preference masters don’t starve. In this algorithm, every master is given out vacancies of multiple lengths, contingent upon the bandwidth necessities of the master. A decision of number of availabilities to assign out to every master is critical. The slots assigned to higher preference masters ought never to be in a manner that the lower preference priority begins to starve. Length of every time span ought to be sufficiently long to finish no less than a distinct data transfer. TDMA scheduling separates executable time in the bus to various partitions and designates time slice to all the masters within which each master has to complete its execution. Let’s say if any master does not utilize its designated partition of time, then the availability of bus could get squandered. Therefore, there comes a need of second stage arbitration to remove this drawback where the role of second stage arbitration is to give access of the bus to master which was fighting for it. In any case, now there is adaptability to set up dynamic runtime connection for guaranteed network bandwidth\(^4\).

The algorithm of round-robin guarantees that there does not exist any starving master. In this algorithm, access to bus is allowed using a circular fashion and ensures each master on the long run will access the bus. Once the master is done with the job from the bus, it surrenders its access and passes the bus ownership to the arbiter. The round robin algorithm is easy to execute, and guarantees equal dissemination of bandwidth to various master, however it experiences one drawback contrasted with the SFP algorithm, due to the fact that critical data exchanges may need to hold up for quite a while before they may continue. Round robin algorithm can be executed in both preemptive (appropriate for clusters) and non-preemptive way\(^1\). An adjusted adaptation of round robin algorithm was demonstrated by Yaashuwanth and Ramesh\(^5\) which changes every one of the disadvantages of
a straightforward round robin algorithm by diminishing the larger waiting and response time with the help of non-preemptive strategy. A Programmable Priority Encoder (PPE) based arbiter using thermometer encoding was introduced by Vilas and Shyam. Their design comprised of an 8X8 switch and a round robin strategy. It is to be noted that due to high context switching in round robin algorithm, there arises system latency. To enhance parallelism using efficient arbitration strategy, local search algorithms have also been developed by several researchers. To reduce the task execution time, some programmers additionally apply local search algorithm to exploit task parallelization.

Some efficient arbitration techniques which keeps user oriented as well as system oriented requirements into consideration are static lottery bus architecture and dynamic lottery bus architecture. Static lottery bus architecture is the probability based arbitration algorithm which gets implemented in the central lottery manager. This system does not possess a fixed topology, as a result various on-chip components can be interconnected in a network of shared channels. Multiple master requests for the ownership of the bus to the lottery manager. Each master is statically allotted a number of lottery tickets, as shown in Figure 1.

![Figure 1. Static lottery bus-based architecture.](image)

The lottery manager chooses the winning master by calculating the number of lottery tickets of each master. Finally a fair share of bus bandwidth is given to the master who possesses highest amount of lottery tickets. In order to avoid monopolizing of bus from a master, a maximum transfer size has been allocated to each master to limit the amount of bus cycles which each granted master can utilize. In order to minimize the idle bus cycles, static lottery bus architecture pipelines lottery manager processes with actual data transfers, thus the latency is reduced. The lottery manager takes the input in the form of sets of requests and the number of lottery tickets each master has. The output is in the form of a set of grant lines which indicates that the selected master is permitted to transfer data across the bus. In order to make sure regarding any pending request, the lottery manager polls the incoming request lines after every bus cycle. For dynamic lottery bus design, set of request lines forms the input in the order $r_0, r_1, r_2, r_3$, which could be seen in Figure 2. In order to generate tickets to various masters, a separate module of ticket generator is attached to the lottery manager.

Since the ranges of various values of ticket are dynamic, as a result, for each component in every lottery a partial sum is required to be computed which is represented by the following equation:

$$\sum_{j=1}^{n} r_j \cdot t_j$$

where $r_j$ represents number of request lines and $t_j$ represents number of ticket generators.

The aforesaid equation is executed using tree and bitwise AND adder. Then the result ($T$) is obtained from the specified range i.e. $T = r_0 t_0 + r_1 t_1 + r_2 t_2 + r_3 t_3$ where all random number resides. This design follows the architecture of static lottery manager only with slight modifications. In this architecture, the allotment of the random numbers is not uniform due to which there arises a slight limitation. The positivity of this architecture is that none of the masters suffers starvation as the access to bus is given to all the masters with a balance bandwidth. However, the drawback of this architecture is due to its sequential style of computing. In addition to this, if the pseudo random number value is greater than the sum of total tickets, then not even a single master gets the permission to access the bus. Moreover, this architecture of arbitration technique also could not be executed in parallel due to the sequential ticket generator.

Yi et al. introduced a new arbitration technique known as adaptive dynamic arbiter. In their arbitration, an approach of lottery bus algorithm was used where an automated adjustment of bus bandwidth proportion is done by the arbiter so that the divided bandwidth could be allotted to the processors. If compared with previous arbitration strategies, their proposed arbitration architecture reduces the task execution time however it does not allot a balanced bandwidth to its assigned processors. In addition to this, their designed architecture does not maximize the usage of numerous CPU cores since they follow sequential style of programming.
Aravind came up with a different algorithm which brings completely distributed programming solution for an arbitration issue where there is an involvement of systems with multi-port memory. The algorithm is absolutely in view of first in first out and least recently used fairness method. In any case, the algorithm does not manage reasonable bandwidth distribution to the diverse masters which may turn into an obstruction to get better execution. In addition, their arbitration procedure is implemented using the approach of sequential programming and accordingly is unable to utilize the numerous cores of the CPU which in turn leads to a delay in the execution of task.

Massimo and Poncino introduced a unique technique of automated synthesis of adaptable bus arbiters including task procedures with dynamic priority. They accentuated specific arbitration strategies which has the capability to be realized on silicon resulting in a digital circuit, as opposed to being worried that how the chosen arbitration approaches could influence the execution of a system with multiple processors. Their arbitration procedure was reasonable with respect to bandwidth and latency. In any case, it puts the slightest concern to CPU core usage as it didn’t adopt parallel programming strategy into account. The significant burden of multi-processor system with a common-bus is the diminishment of throughput created by conflict between different processors requesting access to the shared memory. Realistically, throughput must increase specifically as quantity of processors increases however the bus contention lessens this expanding pattern.

Chen et al. came up with a promising arbitration algorithm approach called RT_Lottery which is real-time oriented and provides guaranteed bandwidth too. RT_Lottery algorithm uses two level policy of arbitration which includes TDMA and lottery based arbitration. They designed masters are indicated by traffic conduct of data stream which comprises of both substantial traffic masters and light traffic masters. As their designed arbitration used sequential programming, therefore synchronization between their masters were not observed. Moreover, with respect to bandwidth distribution, RT_Lottery algorithm cannot compete with adaptive arbitration designed by Li et al. which has the potential to give an automated best bandwidth to various masters which requests bus access. They demonstrated that it is conceivable to assign reasonable bandwidth to a given arrangement of processors with a high level of fairness. The automated continuous reordering of the priority between masters is the prominent function of adaptive arbitration. Their arbiter functioned admirably with respect to fair bus bandwidth distribution, however it didn’t exploit multi-core parallelism.

With an end goal to support stress testing, Nejati et al. investigated the CPU core usage in safety critical embedded systems. Their primary objective was to research whether their system could help designs in inferring experiments for intensive CPU use. In order to accomplish their objective, two target functions were used to record high utilization of CPU execution time. The first computes normal CPU core usage and is indicated by \( f_{\text{usage}} \). The second target function figures the aggregate length of the schedule, and is indicated by \( f_{\text{makespan}} \). Their developed system was useful for application specific tasks computed using heterogeneous masters. Shuai et al. presented another benchmark known as Rodina intended for heterogeneous computing systems to encourage computer designers to examine building stages, for example, multi-core CPU and GPU systems. Using their designed benchmark, various algorithms have been tested. Their research portrays the assorted qualities of the Rodina benchmark to demonstrate that its every component shows novel quality. Nonetheless, bandwidth dissemination stayed unexplored with respect to their investigation. Enzo et al. worked on hybrid parallel programming using a multi-core cluster for DNA sequence alignment. They emphasized on the method to apply the computer power of various parallel platforms to speed up the execution process of sequence alignment without losing result accuracy. Kittisak and Nittaya implemented parallelization on multi-core processors. They employed a single
program multiple data (SPMD) approach on the principal of message passing model. Their work demonstrated a high degree of parallelism along with fault tolerance. It is to be noted that cache optimization and parallel computing runs hand in hand. In order to optimize level-2 cache, Nakhoon and Hwanyong tried to execute matrix multiplication on the multi-core CPU architecture using OpenMP and windows thread. The overall performance of their result in terms of execution time was promising. However, optimization of level-2 cache could be done at a more depth to generate better results.

In order to improve the scalability of distributed computing, Doug Cutting and Mike Cafarella developed a Java written framework called Hadoop. This framework is being used widely in various clusters to build up high performance systems. Moreover, this Hadoop cluster has got a potential to scale-up computation capacity, I/O bandwidth and storage capacity by leveraging on the commodity machines. The prominent feature of the Hadoop framework is its Map-Reduce programming model. Hadoop has got an inbuilt file system called Hadoop Distributed File System (HDFS) which is inspired by the design of propriety Google File System (GFS). HDFS possess a master-slave architecture, where the master could be considered as an arbiter manages. Manages the entire task by mapping it to its respective slaves.

Akhtar and Sidek also came up with a unique strategy of arbitration called Intelligent Adaptive Arbitration (IAA) which takes a shot at the guideline of parallel computation and also sorts out the bandwidth constraints. IAA follows a promising arbitration strategy if compared with aforesaid discussed arbitration methods. Similar to RT_Lottery arbitration, IAA also uses masters so as to come up with an efficient task synchronization. Nonetheless, there are constraints at the synchronization degree of masters due to tight granularity of its masters which still stays unexplored. This in turn affects the processor utilization and the latency. By taking all these factors into consideration, it is not suitable to implement IAA on high end cloud platforms.

In a nutshell, Table 1 summarizes the above discussed arbitration techniques.

3. Design of Masters Related to Traffic Behavior

For the proposed arbitration method, three types of heterogeneous masters were designed as per the behaviour of traffic data to execute an efficient arbitration method which can utilize the numerous cores of the CPU and can allot a balanced bandwidth to the processors. High emphasis was done on synchronization of masters to handle real-time data necessities by ensuring the high CPU core usage. Moreover, in terms of critical real-time requirements, the designed masters have the potential to be put into implementation. Similar masters were also used by Chen et al. For the RT_Lottery algorithm designed by Chen et al., the weight function was used to decide the bus access priority by each master. Their designed masters were beneficial for real-time bandwidth guaranteed applications. For the proposed arbitration technique, the following sub-section elaborates the designed heterogeneous masters.

3.1 Master of D_Type

For the master of D_Type, D represents dependent. With respect to the master of D_Type there are not any real-time requirements to satisfy. Any upcoming request for this type master solely relies on the completion time of any ongoing request. An interval time or time interval is defined as the period from the initiation of the process till the time it gets finished. Figure 3 gives an example of the time interval. There is a 4 beat processing burst which gets generated at cycle 13. However, the permission to process this 4 beat process burst is granted between cycle 16 and cycle 20. Let us assume the time interval to execute the 4 beat process burst is 10 cycles, therefore, a new request by any process could only get issued at cycle 30. In Figure 3, uni-directional line demonstrates that none of the new request is supposed to get issued between 20th cycle and 30th cycle.

Figure 3. D_Type master.

3.2 Master of DR_Type

The master of DR_Type is similar to the master D_Type. It just has got one exception related to the real time parameter R. Let us assume that 10 cycles is allotted to R and its related demand is issued at 5th Cycle. This implies the
demand issued at 5th cycle must get finished by 15th cycle or else an ongoing infringement happens. The working of DR_Type master is shown in Figure 4. As shown in figure, the bi-directional line denotes the primary request which needs to complete before the 15th cycle. In addition to this, the uni-directional line denotes the secondary request which got issued continuously during the implementation of primary request. However, it is to be noted that the permission to the secondary request is given once the primary request gets completed at 12th cycle.

3.3 Master of NDR_Type (ND for Not Dependent)

The master of NDR_Type shows fine granularity by maintaining high synchronization between other masters. Request time associated to the master of NDR_Type of master is independent of the completion time of its previous request. An interval time here is defined as clock cycles between two progressive requests. Figure 5 shows the demonstration of master NDR_Type. As depicted from the figure, the expected time interval amounts to 15 cycles. It could also be observed from the figure that at 21st cycle, a secondary request is allowed authorization and its process gets executed as denoted by the uni-directional line. Moreover, this request specifically relies upon the 7th cycle of the principal request however not on its fulfillment time at the 14th cycle as denoted by the bi-directional line. For this situation R should be lesser than least possible time interval on the grounds that the present request must be completed before the next request gets issued.

| Method                          | Specific drawback                              | Parallel/Sequential Programming | Latency     | Bandwidth allotment |
|---------------------------------|------------------------------------------------|---------------------------------|-------------|---------------------|
| Static Fixed Priority           | Lower priority masters gets least chance       | Sequential/Parallel             | High        | Unfair              |
| Time Division Multiple Access   | Degree of inequality is high between masters   | Sequential                      | Low         | Fair                |
| Round Robin                     | High context switching                         | Sequential/Parallel             | High/Moderate| Fair               |
| Dynamic Priority                | High cost Overhead                             | Sequential                      | Low         | Unfair              |
| Programmable Priority           | High context Switching                         | Sequential                      | High        | Fair                |
| Static Lottery Bus              | Unsuitable for high performance multi-core system | Sequential                      | Low         | Fair                |
| Dynamic Lottery Bus             | Unsuitable for high performance multi-core system | Sequential                      | Low         | Fair                |
| RT_Lottery                      | Unsuitable for high performance multi-core computing system | Sequential                      | Low         | Fair                |
| Smart Adaptive Arbitration      | Thread synchronization limitation              | Parallel                        | Low         | Fair                |
| Adaptive Arbitration            | Unable to harness multi-core parallelism       | Sequential                      | Low         | Fair                |
| Rodina                          | suffers cache misses due to lack of thread synchronization | Parallel                        | Moderate    | Distribution of bandwidth unexplored |
| CPU usage using COMET           | Lacks synchronization of higher level heterogeneous masters | Parallel/Sequential             | Moderate    | Distribution of bandwidth unexplored |

Table 1. Summarization of arbitration techniques

Figure 4. DR_Type master.
4. Smart Adaptive Arbitration (SAA)

SAA technique can maximize the CPU cores utilization using multiple thread synchronization and can provide balanced bandwidth to the various masters as per their bandwidth needs. SAA maintains a high level of parallelism with a balanced bandwidth distribution to its masters and ensures lower task execution time. If suppose any thread worker grows with a notion of $O(n^2)$, then the cost of communication only grows with the pattern $O(n)$, where $O$ is the order function. If suppose the communication time is denoted by $C$, then the total time could be written as:

$$T = \frac{P}{n} + C$$  \hspace{1cm} (2)

Where $P$ denotes parallel component and $n$ represents the total number of threads.

Figure 6 illustrates how the different designed heterogeneous masters are connected to the arbiter. In order to evaluate the performance of the designed masters, a simple synthetic benchmark called “STREAM” (sustainable memory bandwidth in high performance computers) has been used which is designed to measure the computation rate and the sustainable bandwidth of the memory.

The function of STREAM modules are defined in the following points:

1. “stream_copy” measures the data transfer rate without the involvement of any arithmetic operations.
2. “stream_scale” applies a simple arithmetic operation.
3. “stream_sum” takes in one more operand to enable various load and store ports on the vector machines to be evaluated.
4. “stream_triad” does chained, fused, overlapped, add and multiply operations.

To evaluate the performance of SAA with respect to the designed masters of heterogeneous type as per their traffic behaviour, the platform of Amazon EC2 instance comprising of C language was used along with necessary profiler tools and OpenMP libraries to measure the balanced bandwidth allotment and CPU usage.

The specs of the Amazon EC2 machine were a single quadcore machine with 3.40 GHz clock frequency and 8 GB RAM running on Ubuntu 14.04-Linux 64 bits. With an aim to harness the numerous core usage, the designed heterogeneous masters were synchronized so as to run the STREAM modules in parallel.

For SAA, the master of D_Type has been allotted the highest priority and is responsible for task initiation. There is no additional real-time parameter associated with the master of D_Type as the next request can only be executed once the current request has completed its job. The thread process associated with this type of master is tightly time constrained as any new method invoked must execute till a value is returned.

Similarly, the master of DR_Type is same if compared with the master of D_Type. The only difference is that the master of DR_Type possesses an additional real-time parameter as discussed in the section 3.2. To execute the master of DR_Type in an optimum style, there is a data sharing portion defined within the master. The additional real-time parameter of this master has two phases i.e. dependent and independent. For an independent parameter, there is a specific count value defined before which the task execution has to get complete and for the dependent parameter there is a different criteria as it has to synchronize with the master of NDR_Type and has to complete its process until a new request gets issued by the master of NDR_Type.
Initiation for the master of NDR_Type does not rely on the completion time associated with the master of DR_Type. Nonetheless, the master of DR_Type is the one who is responsible to initiate the master of NDR_Type so that the dependent parameter for the master of DR_Type can get executed. The capacity of the master of NDR_Type is to execute the modules of STREAM in parallel with the master of DR_Type. This execution gets done in the OpenMP area and high level of parallelization is kept up by enabling omp_parallel function of OpenMP.

The style of synchronization shared between the master of D_Type and the master of DR_Type is sequential as the master of D_Type is responsible for passing the input data to the master of DR_Type. Accordingly there is also an information reliance requirement between the master of DR_Type and the master of NDR_Type. Figure 7 demonstrates the synchronization associated with all type of masters.

Figure 7. Synchronization between masters.

The pseudo code used for the process of arbitration in SAA is shown in Algorithm I. It is to be noted that the arbiter gives the first preference to the master of D_Type as it is the master with highest priority in the proposed SAA. Moreover, the master of DR_Type and the master of NDR_Type has been assigned with the second and the third preference by considering the synchronization factor in the proposed SAA.

Algorithm I. Arbiter

- Arbiter ()
- Preference set-up:
  - Preference I: Master D_Type
  - Preference II: Master DR_Type
  - Preference III: Master NDR_Type
- if incoming request = Master D_Type
  arbiter grant permission with high preference
  && master of D_Type synchronize with
  master of DR_Type
- else
  - if incoming request = Master DR_Type
    arbiter grant permission with second preference &&
    master of DR_Type synchronize with
    master NDR_Type
  - else
    - if incoming request = Master of NDR_Type
      arbiter grant permission with third preference &&
      master NDR_Type synchronize with
      master DR_Type
    - End if
  - End

5. Results and Discussion

The graphs in Figure 8 gives a contrast between the CPU core usage of SAA and IAA. It is to be noted that for both SAA and IAA, the STREAM benchmark was implemented to test the performance. From the graph it is clear that SAA emphatically beats IAA as SAA has an average CPU core usage of 86.64% where as for IAA the average CPU core usage is 74%. Even with respect to the task execution time, SAA has an edge over IAA.

Figure 8. Comparison of CPU cores utilization rate for SAA and IAA.

Three noteworthy benchmarks must be managed at the same time; CPU core usage, bandwidth utilization of the bus and task execution time. Processor designers need to trade-off the speed versus a great deal of the elements with respect to the specs of processor. Keeping in mind the end result of maximizing the instructions per
clock cycle, the instruction, operands and the destination must be available simultaneously in parallel manner rather than following the sequential style. Li et al.\textsuperscript{13} gave an exploratory study to analyze the bandwidth distribution by taking 4 masters into consideration using their proposed arbitration technique called adaptive arbitration. The arrangement of masters were taken in the order $M_1, M_2, M_3, M_4$ and every 1 in 4 masters required distinctive bandwidth with the proportion of 40, 30, 20 and 10 percent. In Figure 9, the diagram demonstrates the fluctuation in bandwidth with respect to the distinctive arbitration techniques when there is a request generated for the bus access in terms of variable bandwidth by all the 4 masters. It could be seen in the figure that for adaptive arbitration algorithm the gap between the bandwidth requested and the bandwidth allocated is $\pm 3.5\%$ whereas with respect to round robin the gap is $\pm 4.75\%$, for IAA its $\pm 3.4\%$, whereas for lottery bus algorithm the gap is $\pm 3.7\%$ and for SFP the gap is minimum which is just $\pm 2.15\%$.

It can be clearly observed that bandwidth fluctuation in the case of SAA is just $\pm 0.25\%$. This shows that the difference between requested bandwidth and the allotted bandwidth is relatively low compared to other arbitration techniques. This clearly implies high level of synchronization between the designed heterogeneous masters so that the tasks could be implemented in parallel manner by keeping in mind that the bandwidth requirement by different masters may differ from each other.

With respect to balanced bandwidth distribution if compared with other arbitration algorithms, SAA comes out to be better. After analyzing all of the above graphs, it can be said that the SAA is good arbitration strategy for parallelizing the task as it maximizes the CPU cores utilization with balanced bandwidth distribution.

6. Conclusion

An attempt is made to design an efficient arbitration technique which can maximize the usage of numerous cores of CPU to accomplish the problem of task parallelization by keeping balanced bus bandwidth distribution. SAA has a high CPU core usage rate of 86.64\% compared to IAA which is only 74\%. With respect to bandwidth fluctuation, SAA demonstrates moderate change of just $\pm 0.25\%$. SAA could be utilized in high performance application where the parallelization of task plays a major role. More or less, SAA method helps in accomplishing a reasonable bus bandwidth distribution for critical needs by harnessing the CPU cores to its maximum level to accomplish high level of task parallelization. Moreover, SAA reduces the system latency by a wide margin using synchronization of masters. In any case, SAA must be tried on different exceptional image/video processing application for upgraded performance testing.

7. Acknowledgement

"The authors would like to acknowledge School of Electrical and Electronics Engineering and Institute of Postgraduate Studies (IPS) of Universiti Sains Malaysia for Global Fellowship financial support to conduct this research".

8. References

1. Pasricha S. and Dutt N. Communication Architectures-System On-Chip Interconnect. USA: Morgan Kaufmann: 2008.
2. Poletti F, Bertozzi D, Benini L and Bogliolo A. Performance Analysis of Arbitration Policies for SoC Communication Architectures. Design Automation for Embedded Systems. 2003 June; 8(1):189-210. Available from: https://doi.org/10.1023/B:DAEM.000003962.54165.5c.

3. Wiseman Y and Feitelson DG. Paired gang scheduling. Parallel and Distributed Systems, IEEE Transactions. 2003; 14(1):581-92.

4. Samman FA. Runtime connection-oriented guaranteed-bandwidth network-on-chip with extra multicast communication service. Microprocessor and Microsystem. 2014; 38(1):170-81. Available from: https://doi.org/10.1016/j.micpro.2013.07.006.

5. Yaashuwanth C and Ramesh. Intelligent time slice for round-robin in real time operating systems. IJRRAS. 2010; 2(1):126-31.

6. Vilas N and Shyam L. Time Efficient Arbiter in the design of Scheduler embodying ISLIP algorithm for on chip interconnection. IJAST. 2010; p. 2005-13.

7. Davidovic T. and Crainic TG. Parallel Local Search to schedule communicating tasks on identical processors. Parallel Computing. 2015; 48(1):1-14. Available from: https://doi.org/10.1016/j.parco.2015.04.002

8. Chang Hee P, Chi Ho L, Hi Seok K and Jong Wha C. The efficient bus arbitration scheme in SoC environment. 3rd Workshop IEEE System-on-Chip for Real-Time Applications. 2003; p. 311-5.

9. Lahiri K, Raghunathan A and Lakshminarayana G. LOTTERYBUS: A new high-performance communication architecture for system-on-chip designs. Proceedings, Design Automation Conference. 2001; p. 15-20.

10. Yi X, Li L, Ming-Lun G, Bing Z, Zhao-Yu J, Gao-Ming D. An Adaptive Dynamic Arbiter for Multi-Processor SoC. 8th International Conference on Solid-State and Integrated Circuit Technology. 2006; p. 1993-6.

11. Aravind A. An arbitration algorithm for multiport memory systems. IEICE Electronics Express. 2005; 2(1):488-94. Available from: https://doi.org/10.1587/elex.2.488.

12. Massimo E and Poncino M. The design of easily scalable bus arbiters with different dynamic priority assignment schemes. Italy: IEEE-Signals, Systems and Computers. 1995; p. 211-3.

13. JR. Lopez-Blanco, Reyes R, Aliaga JI, Badia RM, Chacon P and Quintana-Orti ES. Exploring large macromolecular functional motions on clusters of multicore processors. Journal of Computational Physics. 2013 Aug; 246(2):275-88. Available from: https://doi.org/10.1016/j.jcp.2013.03.032.

14. Chien-Hua C, Geeng-Wei L, Juinn-Dar H and Jing-Yang J. A real-time and bandwidth guaranteed arbitration algorithm for SoC bus communication. Design Automation. Asia and South Pacific Conference. 2006; p. 24-7. Available from: https://doi.org/10.1109/aspdac.2006.1594751.

15. Li H, Ming Z, Wei Z and Dongxiao L. An Adaptive Arbitration Algorithm for SoC Bus. NAS International Conference on Networking, Architecture, and Storage. 2007; p. 245-6. Available from: https://doi.org/10.1109/nas.2007.9.

16. Nejati S, Alesio SD, Sabetzadeh M and Briand L. Modeling and analysis of CPU usage in safety-critical embedded systems to support stress testing. Innsbruck, Austria: The Proceedings of the 15th international conference on Model Driven Engineering Languages and Systems. 2012; p. 759-75. Available from: https://doi.org/10.1007/978-3-642-33666-9_48.

17. Shuai C, Boyer M, Jiayuan M, Tarjan D, Sheaffer JW, Sang-Ha L. Rodinia: A benchmark suite for heterogeneous computing. IEEE International Symposium on Workload Characterization, IISWC. 2009; p. 44-54.

18. Enzo Rucci AD, Franco Chichizola, Marcelo Naïouf, Laura de Giusti. DNA sequence alignment: hybrid parallel programming on a multicore cluster. Recent Advances in Computers, Communications, Applied Social Science and Mathematics, WSEAS. 2011; p. 183-90.

19. Kerdprasop K. Parallelization of K-Means Clustering on Multi-Core Processors. Selected topics in applied computer science. WSEAS. 2010. p. 472-77.

20. Baek N and Lee H. Implementing Matrix Multiplications on the Multi-Core CPU Architectures. Hangzhou, China: Proceedings of the 6th WSEAS International Conference on Applied Computer Science. 2007; p. 431-5.

21. Cutting D and Cafarella M. Hadoop Framework. USA Patent. 2009.

22. Akhtar MN, Sidek O. An Intelligent Adaptive Arbiter for Maximum CPU Utilization, Fair Bandwidth Allocation and Low Latency. IETE Journal of Research. 2013; 59(2):48-52. Available from: https://doi.org/10.4103/0377-2063.110628.

23. McCalpin JD. Memory bandwidth and machine balance in current high performance computers. IEEE Computer Society Technical Committee on Computer Architecture, TCCA. 2016; p. 19-25.