VOL. 12, NO. 1, JANUARY 2017 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences

www.arpnjournals.com

INSIGHT ON MPSOC ENVIRONMENT AND ITS PERFORMANCE RELATED TO LOAD BALANCE

Mays Q. Sedeeq, Muataz H. Salih, Omar F. Yousif and Nada Q. Mohammed
School of Computer and Communication Engineering, Universiti Malaysia Perlis (UniMAP) Perlis, Malaysia
E-Mail: muataz@unimap.edu.my

ABSTRACT

The late need for a fast processor with high performance forced by the modern highly demanding applications has motivated designers to look for efficient solutions. Highly demanding applications such as networking, image processing, communications and multimedia use Multi Processor System on Chip (MPSoC) as a promising solution. MPSoC provides those applications with high functionality achieving Real-Time deadlines and defeating other vital constraints like the consumed power and the limitations of area. In the MPSoC field, the recently reconfigurable multiprocessor which is often FPGA based multiprocessor, is a modern and growingly significant trend. Reconfigurable multiprocessor brings the vantages of rapidly facilitating prototype and permitting study into more efficient architectures and communications techniques. These multiprocessors also exclude the drawbacks of MPSoC ASIC production. So farthe production of MPSoC made major enhancements to fulfil the unique requirements of embedded applications. During this long journey there were many architecture styles, communication and data transferring strategies in building those systems. Also there were many methods to distribute the work among the connected processors to achieve the best load balance. Load balance has always been of great concern because of its strong influence on the performance of the end design. Generally, load balance has the ability to lessen the effect of wasted resources that exists with the occurrence of idle processors. This paper covers two parts, first everything about MPSoC environment pointing the benefits brought by the implementation of reconfigurable computing and the leading part of FPGAs in this matter. The configuration and architecture of MPSoC is covered as well, with comparisons of different utilised styles to point out the strength and weakness in each of them. The second part discusses all strategies and algorithms used in load balancing to achieve the best improved performance.

Keywords: FPGA system design, MPSoC, multiprocessor, VLSI.

INTRODUCTION

Despite the simplicity brought by the utilisation of a uniprocessor and the advantage of having a straightforward hardware not forgetting its simple software. Yet it doesn't satisfy many uprising applications with their need for a processor with high performance. This is where multiprocessors emerge as a solution for such demanding applications where the need for more functionality. The use of a multiprocessor comes with the advantage of minimising processing time, minimising latency and most important is the given high performance. The gained advantages are tightly connected with satisfying a very important feature in the multiprocessing environment which is load balance. Typically, load balancing can be achieved when tasks are evenly scheduled (distributed) across the connected processing elements.

Through the discussion of MPSoC environment there are many controversial concepts that would possibly create some kind of confusion in dealing with them. Concepts like Mechanism and Policy where the first provides the power to execute an action while policy determines what to do with the mechanism. Some times there is a difficulty in distinguishing between parallel and distributed systems, which will be distinguished according to the autonomy of individual node in the system. Autonomy is confirmed in distributed systems, yet this is not the case in parallel systems. When a node has the freedom to behave differently than the rest of the nodes in the system, it is considered an autonomous node. Design autonomy, communication autonomy, execution autonomy and administrative autonomy are the four components to the autonomy of amultiprocessor system. Earlier distinctions were grounded on the possibility of needing interrupt to access some parts of memory like if communication among processors is through shared memory (tight coupling), or through message passing (loose coupling).

The mix in the concept between a job and a task has always been faced in multiprocessing environment. The computing unit in a computing systems is called a task while various tasks functioning to achieve a common goal are known as a job. So in such environments the handed situation is having various jobs each composed of many tasks, and all competing to be served by a processor. The allocation of tasks on a processor is either assigning several processors to a single job which is called space sharing or assigning several tasks to a single processor and that's called time sharing.

In the MPSoC field, the recently reconfigurable multiprocessor which is mostly FPGA-based multiprocessor is a modern and growingly significant trend. Reconfigurable multiprocessor brings the vantages of rapidly facilitating prototype and permitting study into more efficient architectures and communications techniques. These multiprocessors also exclude the drawbacks of MPSoC ASIC production. Reconfigurable Multiprocessor Systems, or often known soft multiprocessor is mainly proposed as a meanto help in prototyping systems for later implementation on an ASIC. At the present, FPGAs opened new horizons not just for implementing prototypes, but for the benefit of the final

designs as well. The development of FPGA capacity permits designers to implement an entire multiprocessor system on one FPGA. The FPGA chief companies provide the possibility of utilising soft-core processors particularly designed to well match in the FPGA. At the same time, FPGAs permit the utilisation of hard-core processors. Moreover, FPGAs are fitted with on-chip memory blocks, peripherals, and interconnection circuitry. One of the FPGA based multiprocessor strengths is the run-time reconfigurability. This characteristic provides multiprocessor systems an adaptation feature to a particular application, making the designed system to earn much of flexibility.

The power provided by the utilisation of reconfigurable processor has been examined by [1-3]. The researchers did confirm that reconfigurable multiprocessor based FPGA provided adaptations for many faced challenges.

Since the second part of this paper will deal with load balancing the confusion between load balancing and load sharing should be cleared. Load sharing permits processors which are busy to offload some of their loads (work) to idle or less busy processors. Load balancing is a particular case of load sharing, aims to keep the load even (or balanced) across all processors and this is the goal of global scheduling algorithms. The threshold load is an important term in load balancing strategies. The totalload to a processor that every load would come further to that processor is a threshold load.

MPSoC substantiate very important and distinct branch of multiprocessors. MPSoC is not just a traditional multiprocessor which was reduced to be on a single chip but it hasbeen designed to satisfy the unique requirements of embedded applications. MPSoCs have been produced for much longer than multicore processors [1]. Many researchers start the discussing of MPSoC systems by classifying them into homogeneous and heterogeneous [1, 2].

In many works there was always a comparison of homogeneous and heterogeneous multiprocessing systems. [3] Showed the drop in performance in both configurations when number of threads was less or equal to the number of connected processors. As the number of threads exceeds the number of processors, the basic condition to get best load balancing is satisfied. The improvement in performance is very much related to absence of an idle processor/s. And since there is a great impact of load balancing (where the distribution of tasks on the connected processors) on the efficiency of MPSoC, load balancing will be discussed in details in the second part of this paper.

RECONFIGURABLE PROCESSORS IN MPSoC ENVIRONMENT

Reconfigurable processor is a microprocessor that has a hardware which can be erased with the ability to dynamically rewire its self. Such features permits the chip to gain an effective adaptation to the programming tasks needed through the particular software that are interfaced at any time [4].

It was Xilinx in the eighties and exactly 1984 who first introduced the programmable logic devices [5]. Firmly after that, the reconfigurable computing made acceptance as a prototyping platform. Then reconfigurable platforms made enhancements in their capabilities especially in terms of their flexibility and performance. It was considered as an expected solution to bridge the gap between processors and Application-Specific Integrated Circuits (ASICs).

Probably some would ask why FPGAs are the selected platform to build reconfigurable multiprocessor. The answer is actually for its many the benefits brought to the end system through the utilisation of this platform as compared to ASICas shown in Table-1.

Table-1. The advantages of utilising FPGAs over ASICs in MPSoC.

1	Flexibility and Reconfigurability	The number of soft core processor which could be connected is set by the FPGA capacitance, with a possibility to independently configure each processor.
2	Minimising time to market	Through the design process there is no need to include the manufacturing of the Integrated Circuit (IC). Reduces design time significantly.
3	Minimising the costs	Cheaper process. Today the FPGA are comparatively cheap. The appearance of an error in the system design is uncritical.
4	Scalability	The FPGA based multiprocessor can put up to larger number of microprocessors or peripherals.
5	Fixing bugs ability	In the case of having available logic resources in the FPGA.

When dealing with reconfigurable computing and its tight connection with FPGAs two terms will pass, which are soft processor and hard processor. Table-2 will shows the difference between both of them in the aspect from very important parameters that could be of a great concern to designers.

Table-2. Comparison between soft and hard processors.

	Soft processor	Hard processor
Implementation	Implemented in FPGA fabric.	Physically implemented as a structure in the silicon.
Speed	250MHz and less (generally less than 200MHz as its limited by speed fabric)	100's MHz up to 1GHz (faster processing speed as they are optimized and not limited by fabric speed)
Modification	Can be easily modified and tuned to specific requirements, more features and custom instructions.	Fixed with no capability to be modified.
Cost	More expensive	Cheaper.
Power consumption	Consumes more power	Consumes less power

Nowadays the two main producers of reconfigurable architecture based FPGA are Altera and Xilinx. The two corporations are competing to provide their products with best resources. Through this they made availability of not just logic blocks, but Digital Signal Processing (DSP) cores, on chip memory blocks and hard IP processor cores as well [6].

Altera Corporation provided at first Nios processor which was a 16 bit embedded processor. Nios was then evaluated to NiosII 32 bit embedded processor. The latter is suited for a broader lay out of embedded computing applications, starting from DSP to system control. The third-party IP provider along with Synopsys design ware made NiosII licenced for standard sell ASICs. And this criteria gives designers the ability to migrate Nios-based designs from an FPGA-platform to a mass production ASIC-device.

There are three versions of Nios II each to address a certain application [7]. For power and cost sensitive applications Nios II economy core (NiosII/e) is the best suited with low logic elements of 600, made this version ideal for microcontroller application. NiosII fast core (NiosII/f) is absolutely deterministic with hardware real-time features. The third version NiosII/s (where s for standard) is the one to keep balance between performance and cost. Altera tends the Nios processor as the most versatile processor, and this claim is supported by Gartner Research.

On the other handXilinx proposed MicroBlaze, PicoBlaze and PowerPC processors. Each has its own criteria in concern to number of bits and type of core. MicroBlaze is a RISC 32 bit soft core processor with an architecture of Harvard memory. PicoBlaze is an 8 bit RISC microcontroller and it's an open source processor. PicoBlaze is very small in size with a performance can reach up to 240 MHz. PowerPC is also 32 bit RISC hard core processor developed by IBM.

Very common open source processors are OpenRisc produced by OpenCores Corporation, and Leon3 produced by Gaisler Corporation. Leon 3 is the one with more elevated functions as compared to MicroBlaze and NiosII. Yet, it suffers from a limitation, which isthe occupation of a large amount of logic resources. Therefor It's difficult to fit on a single FPGA a large number of units.

It's important to point out the ability of FPGAs to be partially reconfigured [8]. This special case of reconfiguring takes place when a critical part of the design will continue its operation when the controller loads a partial design in to a reconfigurable subcomponent. The controller could be on or of the FPGA, and during this operation the FPGA must be held in reset state. Partial reconfiguring is departed in to two types dynamic and static partial reconfiguring. The former is also called active partial reconfiguring; where permission is given to change a part of a component when the rest of an FPGA is still running. On the other hand, in the static type part of the data is transmitted into the FPGA when the device must be in the shutdown mode. When configuring is completed the device is brought up.

It's obvious that FPGA platform is becoming the leader in reconfigurable computing, for its outstanding features and all the enhancements given through vendors.

MPSoC TAXONOMY AND ARCHITECTURE

Through the production of MPSoC in the last decade till now, there has been a classification of those systems and many proposed architecture to make full advantage of the connected processors. There have been also different communication methods with different data transfer styles each has its own advantages and disadvantages that will be discussed in the next paragraphs.

MPSoC Taxonomy

Mainly MPSoC can be classified according to the similarity of the connected processors into homogenous and heterogeneous systems. Homogenous systems utilise identical processors commonly with shared memory architecture and with one operating system to control the whole design. Homogeneous MPSoCs are normally general-purpose systems. In this type of system, the processors count is more likely to be incremented with no need to change the architecture providing scalability property. Asa comparison to heterogeneous systems, it is considered easier to develop software for homogeneous systems. Heterogeneous multiprocessor systems denotes to systems that utilise more than one type of processors or cores. These systems achieve performance or energy efficiency not only through adding the same type of processors, but through adding different types of processors. Usually those systems in order to handle

particular tasks they incorporates specialized processing capabilities [5].

New Heterogeneous System Architecture (HSA) use different types of processors generally CPUs and GPUs on a single chip. These will provide the end system with the best capabilities of both connected processors. With general GPU processing and separately from its reputation as 3D graphics rendering capabilities, GPU can as well execute intensive mathematical computations on huge data sets. As CPUs can run the operating system and only execute traditional serial tasks [9].To effectively use the multicore processor possible performance, the application should be broken into of parallel tasks. It is incessantly demanded that the number of parallel tasks exceeds the number of cores [10] [11]. Yet, some applications cannot be broken into parallel tasks. This in where several executions are demanded and in this aspect heterogeneous multicore processor has the upper hand. The control mechanism in heterogeneous multicore processor schedules the several tasks on the core with high performance. While parallel tasks are executed on the simpler cores. By doing so, the execution time of the several tasks reduces and the overall performance is boosted.

The approach of heterogeneous multiprocessor design has the benefit of making a match between processor cores with application-appropriate features to specific on-chip tasks. The selection of the right processor core or matching the processor core to a particular task provide many advantages. First, more abilities than the requirement of its assigned task set, are not demanded. This characteristic of a heterogeneous-multiprocessor system design with such characteristic lessens the gate counts of a processor by cutting the unneeded features from each processor.

Power consumption is a fundamental challenge in high performance processor. The asymmetric chip multiprocessing has the ability to dynamically switch between different cores and powering down unutilized cores. [12] Has demonstrated that a heterogeneous processor utilizing two core types accomplishes improvement in performance as much as a 63% percent compared to an equivalent area homogenous processor. Alsoa better coverage of spectrum load levels can be gained by heterogeneous multiprocessors.

One of the main drawbacks in heterogeneous multiprocessor design is the necessity to utilise a different software-development tool sets for every different processor cores connected in the system. The software-development tool like the compiler, assembler, debugger, instruction-set simulator, real-time operating system and etc. Also the firmware team must become proficient in using all of the tool sets for the various processors or more likely the team must be split into groups, each assigned to different processor cores.

As a conclusion it is obvious that the handed advantages of heterogeneous chip multiprocessors outweigh that of homogeneous chip multiprocessors. The biggest difficulty stays in teaching parallel programming techniques where most programmers are so versed in sequential programming. Multicore processors are an important innovation in the microprocessor timeline. With competent programmers supported with good knowledge in writing parallelized applications, dramatical increase in multicore efficiency could be achieved.

MPSoC architecture

When designing MPSoC there are three basic cores that should be in the mind of the designer. First is how to organise the processing elements and in what architecture. Second is how to choose a communication method across the connected processors. Finally is selecting the type of data transfer. In the next paragraphs there will be a detailed discussion of each style presenting their advantages and disadvantages in tables.

System architecture

The system architecture is mainly about the organisation of the processors, where basically there are three methods.

A. Master-slave

This model shown in Figure-1 was utilised in the early days of multiprocessors. Where the behaviour of the slave processors is under the masters control. There is an individual data structure which keeps track of ready processes. When a CPU becomes idle, it inquires the operating system for a process to run so it is assigned one. Hence it can never happen that one CPU is idle while another is overloaded. Likewise, pages can be allocated among all the processes dynamically and there is merely one buffer cache, so inconsistencies never occur. Yet this architecture has low reliability because the high dependency on the master where its failure causes the crash of the whole system.

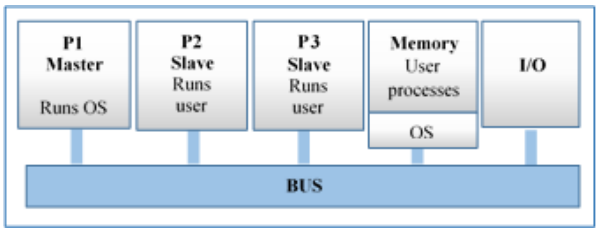


Figure-1. Master-slave multiprocessor [1].

B. Pipelining

The architecture of such multiprocessor is composed by a chain of processors and each processor act as pipeline stage as shown in Figure-2. This architecture is considered to be useful in streaming applications.

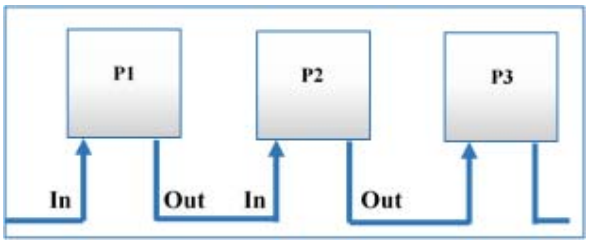


Figure-2. Pipelined multiprocessor [1].

C. Net architecture

This architecture denotes to multiprocessor systems with no hierarchy between processors, and when necessary the connected processors have the ability to communicate with each other. One of the good example of such multiprocessors is Symmetric Multiprocessor (SMP). Table-3 summarises the advantages and disadvantages in each type of systems architecture.

Table-3. System architecture comparison.

System architecture	Adv.	Dis.
Master- slave	Simple scheduling of the operating system. Only the master processor is responsible of performing functions. Only one processor is permitted to access the data structure. Free of conflicts.	Not reliable (The failure of the master causes the crash of the system). Poor use of resource (when having free slave and an occupied master). Increase in the level of interruption (when slaves interrupt the master).
Pipelined	Considerably fitted to data flow nature of multimedia applications. Minimise data transfer and power consumption.	Slow full-system cycle. Electrical loading problems. Restricted bandwidth. Increase in latency and throughput limits.
Net architecture	Large global memory (where data intensive applications can benefit from). Simpler node to node communication.	Memory latency issues. Bandwidth limitations (where the bandwidth of a given node can be affected by other nods) Cache problem may occur in some applications

Communication across processing elements

Is one of the decisions that need to be taken when designing MPSoC, is the way that communications are physically established among the connected processors.

A. Point to point

In this type of connection shown in Figure-3, the processors are directly connected with an access to full channel bandwidth. Although it has a bandwidth advantage and it is efficient in terms of fast data exchange, yet it is not efficient for large systems. The designer is unable to increase the number of processors. Otherwise, this will definitely increase exponentially the complexity of the system and its size, as each resource should have a direct connection with each one of the others resources.

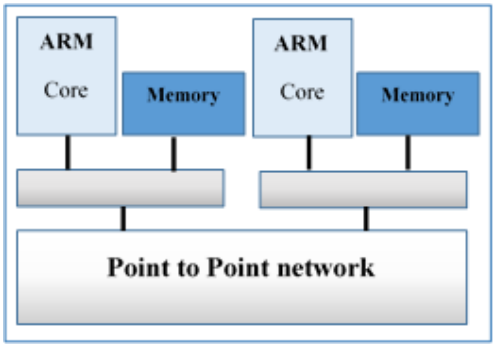


Figure-3. Point to point connected multiprocessor [2].

Despite the debate in that the complexity of the point to point multiprocessor system will increase with large number of resources, some designers faced difficulties with low number of resources [13] like in 7 nods and 10 connections as shown in Figure-4.

Till this point, the discussion was only about connecting nods with a single connection, the complexity will raise when there is a need to connect all nods or when communications are bidirectional.

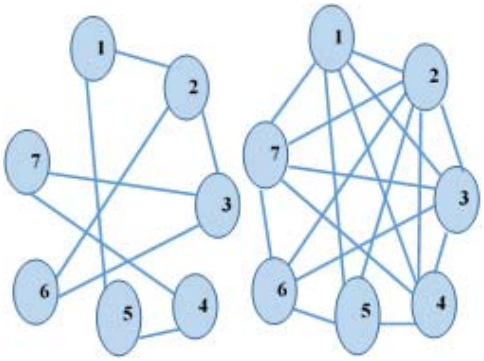


Figure-4. Point to point complexity with small number of nodes.

(a) Point to point connection (b) All connected nodes

Point to point connection really shows a need for a protocol to somehow synchronise the communication through the connected points. In this aspect there are three known protocols:

- Rendezvous protocol: long familiar for point to point communication. In this protocol, the sender must be stopped until the receiver is ready to receive and inversely. This assures that both sides are synchronized before the transfer.
- Coware: is an architecture and a related MPSoC design flow using point to point communication.[14]
- OCP (Open Core Protocol): this is a point to point interface that renders a standard set of data, control and test signals that permit different cores of MPSOCs to communicate [13].

B. Shared bus

It is the most beneficial experienced mechanism to make communication among the connected processors, it's inspired from monoprocessor architecture.

Few enhancements, such as adding priority and arbitration rules for bus access made it more suitable for multiprocessing systems. The arbitration policy has a straight impact on the performance of MPSoC. This architecture comes with the benefit of simplicity since there is a single channel of communication which will reduces design time.

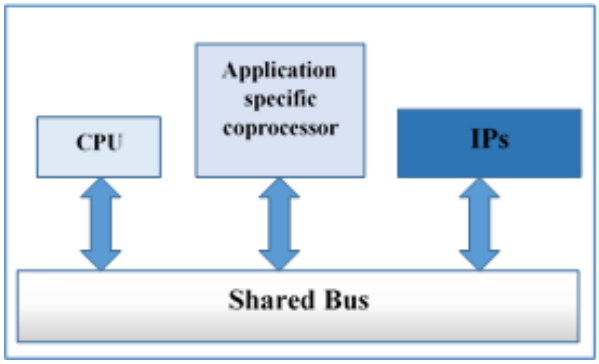


Figure-5. Processors connected via shared bus [2].

Accordingly, this architecture does not require a lot of surface and it's not costing (especially when compared with point to point architecture). Yet it's considered limited by its bandwidth and throughput available between the units on the bus, which is inversely proportional to the number of these units. As a result this architecture can be assumed to be a fine solution for small number of units. On the other hand this type of communication is qualified by its low scalability and/or flexibility also by its high power consumption.

C. Network on chip (NoC)

It is the most promising and late approach in connection trends in an MPSoC environment. The grounds of this approach is to employ network background to the on chip system. Small routers are utilized inside the chip to enable communication between all processors of the system with low latencies.

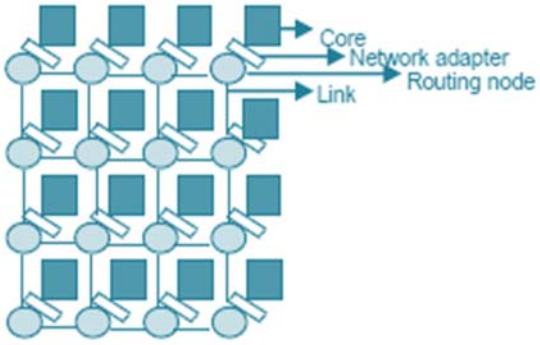


Figure-6. Basic NoC components [15].

NoChas leakage power consumption (when its links are idle they still will consume power in repeaters, due to the dominance of this leakage current at small feature sizes. Among the three communications methods NoC is considered the most promising one with the quickly approaching billion transistors era. Some of the primary issues in deep sub-micron technologies which are qualified by gate lengths in the range of 60-90 nm will arise from non-scalable wire delays, errors in signal integrity and unsynchronized communications. The

mentioned problems could be overcome by the utilization of Network on Chip (NOC) architecture [15]. Table-4 compares these system connection types, giving a brief about the advantages and disadvantages of each type.

Table-4. System connection comparison.

	Adv.	Dis.
Point to point	High bandwidth (where it's not necessary to share communication channels) Renders security and privacy (since communication channel is not shared.	Not area efficient as the system grows (with large no. of connected processors). Limited scalability limited flexibility
Shared bus	Cost effective due to reduced number of channels and interface hardware components. Area efficient.	Limited bandwidth. Security consequences (as a result of sharing single bus for all processors). Low scalability and/or flexibility. High power consumption
NoC	Good performance (similar to point to point). More efficient than shared bus. Less power consumption than shared bus. Less surface demanding than point to point. More scalable than both point to point and shared bus.	Limitations in serial and parallel links utilised in the transportation of packets among the connected elements in NoC system. Not cost effective due to complex router protocols, consuming more area and power. NOC include cost, power consumption, and silicon overhead in terms of the area that is required to support the packet switched network.

Data transfer

It's mainly when dealing with the ways where data is transferred among processors and there are two methods.

A. Message passing

This kind of architecture shown in Figure-7 is utilised to communicate data among the connected processors without the need for global memory. While each processing element possesses its own local memory, messages are used to make some kind of communication among those processing elements. In message passing network there are two important elements that should be taken into consideration:

- The link bandwidth which refers to the number of bits that could be transmitted in per units of time (bit/sec).
- The transfer of a message across the network.

In message passing to exchange messages among processes running on a yielded processor uses what is called internal channels. On the other hand, external channels are utilized to exchange external messages by processes running on different processors. The data exchanges among processors are not for share that's why and via send and receive, messages data is rather copied. This type of data exchange provides a significant advantage which is the elimination of synchronisation conceptions such as semaphores and this will improve performance.

The most important support provided to this type of architecture came through Message Passing Interface (MPI). MPI is a language-independent communications protocol utilised for programming parallel computers. MPI "is a message-passing application programme interface, along with protocol and semantic specifications for how its features should act in any implementation" [16]. MPI's aims are high performance, scalability, and portability also they provide essential virtual topology, synchronization, and communication functionality between the set of processes. MPI is the exclusively message passing library that can be considered a standard. This standardization made it supported on virtually all High Performance Computing (HPC) platform.

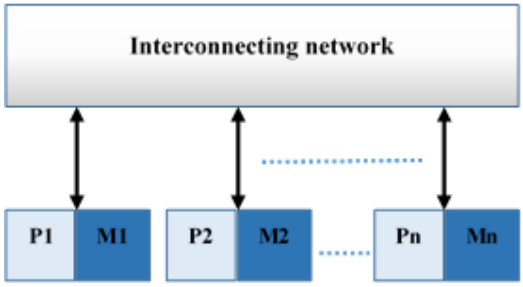


Figure-7. Message passing architecture [2].

Message passing paradigm is demanded for its wide portability which could be utilised in communication for distributed-memory and shared-memory multiprocessors, networks of workstations, and a combination of these elements. The paradigm is applicable in multiple settings, independent of network speed or memory architecture. The MPI basics are:

- Start a process
- Send a message
- Receive message
- Synchronize

The MPI with its portability feature eliminates the need to modify the source code when porting the application to a different platform that supports the MPI standard. Since 115 routines are defined MPI is considered to have the highest functionality. MPI stays the dominant model utilized in high-performance computing today.

B. Shared memory

Shared memory demonstrated in Figure-8 is a type of transferring data across processing elements in a multiprocessing system. Where the processors communicate via shared address space. This type is considered the fastest in the concern of communication but not the easiest especially concern to the synchronisation needed among the connected processors. Typically one processor allocates the shared memory segment, the size and the access permissions to this segment is set when it's created.

In a system with shared memory, the same memory resources are shared among the connected processors; hence, all changes created by one processor to a certain memory location get to be visible to all the other processors in the system. Also different processes are easily capable of exchanging information via shared variables; yet, it demands deliberate covering of synchronization and memory protection. Because of the existed absents of a formulated communication in shared-memory system, it demands synchronization mechanisms like semaphores, barriers. There have been implementations on shared memory thread model like POSIX threads [17] and OpenMP [18] which are two popular ones.

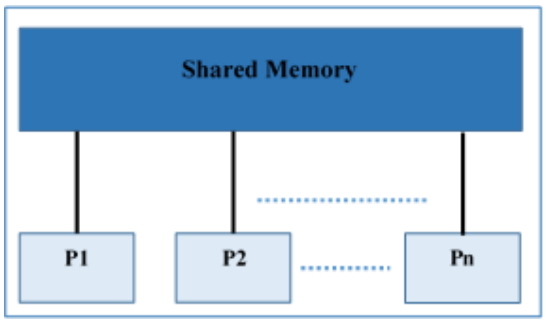


Figure-8. Shared memory architecture [2].

Table-5 points out all the advantages and disadvantages in these two data transfer types.

Table-5. Shared memory vs. message passing.

	Message passing	**Shared memory**
Denotative communication	Via messages (send, receive)	Via memory operation (load, store)
Coupling	Loose coupling of program component	Tight coupling of program component
Machine model	No global address space	All processors see a global shared address space
Cost	More expensive	Less expansive
Delays	May happen and each message could have its own delay.	No occurrence of delays.
Information loss	Lost messages or overflow in the inbox buffer	Occur through overriding
Consistency	Low (because of confusion in the arrival of many messages at the same time)	Higher (the value of a register is always the value that was written last)
Synchronisation	No need for synchronisation mechanisms (all communication and synchronisation are done via messages exchange).	Need for synchronisation mechanisms such as semaphores.

For more scalability, and in the same aspect of creating a communication area among processors, come the Distributed Shared Memory (DSM) systems shown in Figure-9. In such systems, the physically separated memories can be logically addressed as one address space. Since the same physical address on two processors refers to the same location in memory, this would explain the real idea behind the expression "shared" which does not mean the existence of one centralised memory.

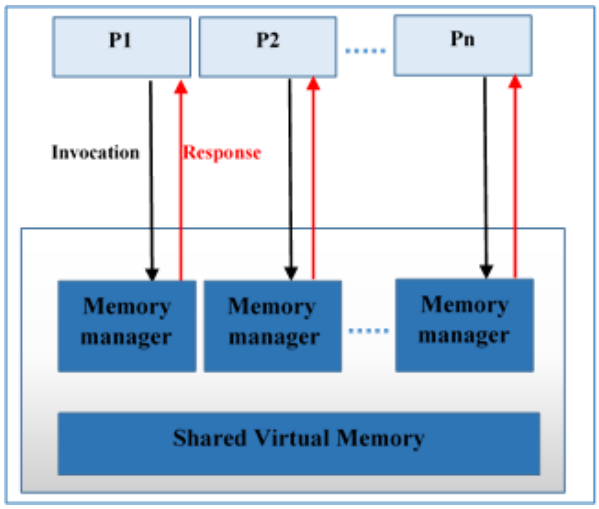


Figure-9. Distributed shared memory (DSM).

Also but in contrast with shared memory there is the Distributed Memory (DM) system. Generally, in Distributed Memory systems generally there is a processor, a memory, and some form of interconnection that permits processes on each processor to interact with each other Figure-10, while message passing protocols are utilised to carry out data transfers.

In Distributed Memory when one processor necessitates an access to data in another processor. Generally, it is the job of the programmer to explicitly determine when and how data is communicated. Similarly, the synchronisation among tasks is the duty of programmers. The interconnection network utilized for data transfer varies widely. This machine is considered more scalable than shared memory system, since only the communication medium may be shared among processors. One of the requirements in Distributed Memory system is the mechanisms for supporting explicit communications between processes. Generally, a library of primitives that permits writing in communication channels. The Message Passing Interface (MPI) [19] is the most popular standard. One of the major advantages of Distributed Memory system is the exclusion of race conditions as each node has its own part of memory to work with

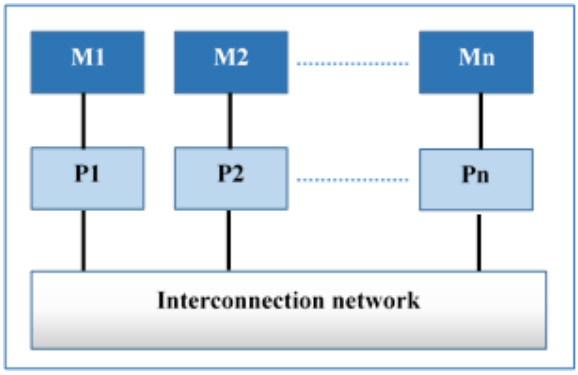


Figure-10. Distributed memory system.

Table-6 highlights the advantage and the disadvantages of both Distributed Shared Memory and Distributed Memory systems, where these factors make enormous effect on the end design.

Table-6. DSM vs. DM.

	Adv.	Dis.
DSM	System scalability. Render large virtual memory space. Ability to deal with complex and large data bases with no replication or sending the data to processes. Memory bottlenecks free, where no single bus. It has are portable programs since they utilise common DSM programming interface. Possible ability to increase performance by speeding data access.	Possibility of causing performance penalty. Locks are demanded for protection against simultaneous access to shared data. Facing difficulties with irregular problems.
DM	Memory is scalable (with the increasein the number of processors the memory size increases). Each processor has the ability to rapidly access its own memory no need for interface and overheads. Cost effective.	A lot of burdens on the programmer, associating data communication between processors. Non uniform memory access times.

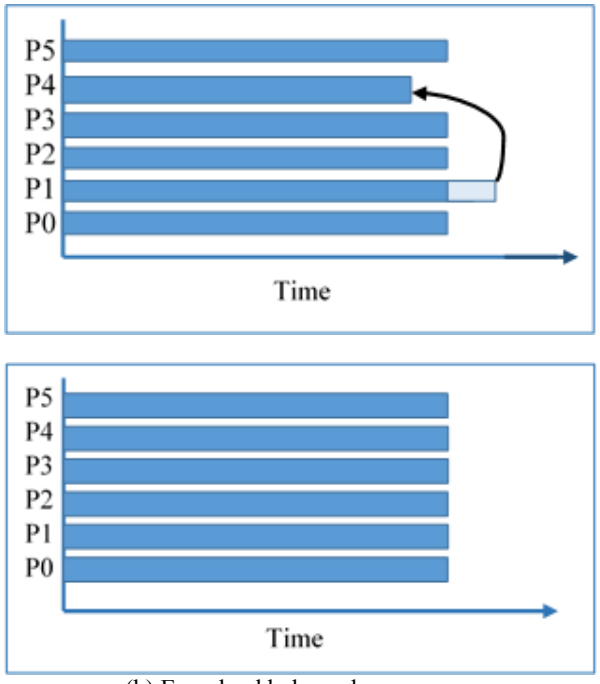
LOAD BALANCING

Since most MPSoC systems handle parallel concurrent application wherein fast and efficient performance is in it basics, load balance appears as the most relevant feature which should be well thought of. When workloads are distributed evenly among the

connected processors parallel systems runs most efficiently and quickly.

So in some cases when there is a need to harness the power of a multiprocessor system specially when dealing with a demanding application. In a selected application the concurrency degree should not be too course. With good load balancing the application makes total vantage of the entire processing power.

To assure not having an idle processor in a multiprocessing system, usually it's preferred to have the number of processors (N) less than number of tasks (T). With satisfying the condition of (T>N) this will assure that each processor is busy with a task on a time. Otherwise, when (N>T) then for sure there will be an idle processor and that is waste of a resource. In terms of the selected concurrency degree, it should not be coarse because generally when (T) gets too large relatively to (N), the amount of synchronisation and communication needed to back up the application increases as well. The consequences of such situation bring down the efficiency as the time needed to achieve the desired synchronisation and communication will exceed the amount of time demanded to do the actual work.



(b) Even load balanced processors

Figure-11. (a) Execution of uneven load balance processors.

As shown in Figure-11, it is obviously that with efficient load balancing there is more potential to create a system where every processor is equally busy, finishing the work approximately at the same time. Generally, load balancing provide many benefits when utilised such as:

- Brings Improvement in to the performance of every node therefore the overall performance improves.
- Reduces the job idle time.
- Ends starvation suffered by small jobs.
- Maximum utilization of available resources.
- Increasing throughput.
- Increasing reliability.
- Lowering cost with an increase in gain.
- Extensibility with an increase in gain

The primarily classification of load balancing is either static balancing or dynamic balancing. Both of the balancing strategies could possibly be farther classified into several differed types, which depend on specific application being run and its characteristics. Some methods of dynamic balancing are discussed.

Static load balancing

In static load balancing strategy and prior to processors runtime, the single processors load is biased. For applications running on a committed system with predefined features, this strategy considered the best. Several tasks are generally partitioned consequently at compile and /or link time. It is possible to determine the assignment of a task by:

- Size of the problem.
- The number of available processors.
- Proportional performance of single processors.

Nevertheless, as the assignment is settled, nothing will be altered at runtime. Static load balancing is the simplest strategy of load balancing and is generally done by mapping tasks to processors. Given an application with graph A and parallel host system with graph H, the goal is to efficiently embed graph A onto graph H (9). In order to allow the application of a quick and smooth running, a cut down in the demanded synchronisation and the intercommunication of a processor should be achieved. This achievement is provided through more beneficial mapping. The better mapping assures of taking the full advantage of all available computing power.

The main disadvantage in this strategy is that when deciding an allocation the current state of the system is not considered. In distributed system where the load fluctuation is, this unconsidered point has a major effect on the overall performance of the system [19]. There are three types of static load balancing algorithms.

A. Round Robin algorithm

In round robin algorithm, there is an even distribution of the load to all nodes, in a round robin order. In circular order, load is assigned to each node equally with no need for any priority. The algorithm will return back to the first node when the last node is reached. The load index of each node is kept independent of allocating from remote node [20]. Round robin has the advantage of easy implementation, simple algorithm and no occurrence of a starving node. In this algorithm there is no requirements for inter process communication. For special purpose applications this algorithm hands the best performance. The only drawback is that it doesn't provide

the anticipated result in general case and when the dealt with jobs are of unequal processing time.

B. Central manager algorithm

There is a central node in this algorithm has the authority to choose the slave node that will transfer the load. The slave node where the least load is selected and the job is assigned to it. The load index of the entire nodes in the system is kept connected to the central node. Then a message is sent through the slave nodes as load is changed [21]. The highly level of inter process communication demanded, is the drawback in this algorithm creating bottlenecks in some cases. As different hosts create dynamic activities, this algorithm will provide more beneficial performance.

C. Threshold algorithm

In this algorithm at the moment where the node is created the load is assigned. Nodes are chosen locally with no demandsfor remote messages to be sent. A private copy of the system's load is kept by each node. There are three characterisations levels for a load. Those levels are:

- Underload.
- Medium
- Overloaded.

Tunder and Tupper are two utilised parameters to trace these levels [22].
Under loaded - load<Tunder
Medium - tunder<= load<=Tupper
Overloaded - load>Tupper

The entire nodes in the system are regarded in under loaded level state at the beginning. When the load state of a node exceeds a load level limit, then it sends messages. The sent messages are concerning the new load state, which is sent to the entire remote nodes so that the nodes are updated on a regular basis to the actual load state. The load is allocated locally when the local state is not overloaded. Differently, a remote node in the underload state gets chosen, if no such node exists it is also can be allocated locally. The low interprocessor allocation along with large number of local process allocations are provided in the Threshold Algorithm. The reduction in overheads of remote process allocation along with the overhead of remote memory access is gained through the large number of process allocations. The handed reductions will conduct performance improvement.

D. Randomized algorithm

In this algorithm the node is chosen randomly, while not having any kind of information about the current or previous load on the node. The static nature of randomised algorithm made it more likely to be best suited for a system with equally loaded nodes [23]. For special purpose application this algorithm provides the most beneficial performance. There is no need for inter process communication as each node keeps the load record of its own. Yet, in this algorithm there is some possibility of having the situation where one node overloaded while the other node is under loaded. Table-7 points out the issues and the drawbacks of both static and dynamic load balancing.

Table-7. Comparison between static and dynamic load balancing.

	Issues to be covered	**Drawbacks**
Static balancing (utilised in homogeneous environment) **Knowledge base**	The response time. The resource utilization. Scalability. Power consumption and Energy Utilization. Makespan Throughput/Performance Prior knowledge base is required about each node statistics and user requirements.	Not Flexible Not scalable Is not compatible with changing user requirements as well as load.
Dynamic balancing (utilised in heterogeneous environment**)** **Knowledge base**	Location of processor to which load is transferred by an overloaded processor. Transfer of task to a remote machine. Information Gathering. Load estimation. Limiting the number of migrations. Throughput Run time statistics of each node are monitored to adapt to changing load requirements.	Complex Time Consuming.

Dynamic load balancing

This scheme of load balancing is done dynamically at runtime [24]. In this strategy there is an equal distribution of tasks. The tasks are distributed out evenly across the held processors and accordingly there will be workload adjustment. Dynamic load balancing lends itself to most parallel applications. This strategy of balancing has the vantage of being customized to the particular system it is being run on at any given time. In a dynamically balanced system, the processor with high

horsepower (where the processor has high computing power) the algorithm would make a full vantage and utilise this power through more workload assignment compared with other processor with less computing power. To adjust and perform future patterns prediction, past execution results may be utilised in some methods. While from the system and at runtime the extracted real time performance characteristics may be utilised in other methods to make application adjustment [25].

Despite all the mentioned advantages, dynamic load balancing has an obvious disadvantage which is the added runtime support needed to run the balancing algorithm. Generally, the algorithms monitor, do the following steps:

- Information exchange among processes.
- Workloads calculation and distribution.
- Redistribution of the workload (in some cases).

Hence, perfect workload balancing algorithm can possibly be a burden itself. This situation takes place as the algorithm consumes lots of cycles to run, affecting the cycles needed to do the actual application. Nevertheless, decently selected algorithm could add overheads but this could be offset by the improved performance of the dynamically allocated and optimized application.

In general, there are two main types where dynamic load balancing strategies can be classified in, which are either centralized or distributed.

A. Centralized dynamic load balancing

In this strategy tasks are passed out to a central location as in Figure-12.

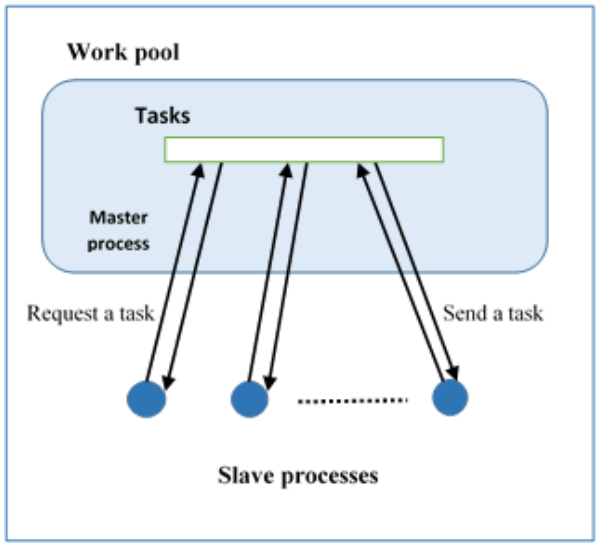


Figure-12. Centralized dynamic load balancing [25].

In single work pool individual tasks are placed. To these individual tasks, all the work wanted to be done is allocated to. To all available processors, tasks are passed out initially in the balancing process. The balancing process initially passes out tasks to all available processors. Through the balancing process, processor is assigned with new tasks to run as it completes its old ones. There is an obvious master slave pattern in this strategy.

Centralized load balancing has the advantage that control is kept in a central location where it may be easily monitored. Nevertheless, the useable of a centralized system is limited. As the number of available processors increases, the amount of communication and synchronization needed to maintain the system as well increases. A bottleneck may be formed at the distributing process, causing idle processors expecting work tasks. Centralised load balancing will further divide into two parts

- Predicate the future algorithm
- Task queue algorithm.

Predicate future algorithm can distinct both task and data by assumption or probability base theory future requirement based on performance of past information. [26] Have used this algorithm in cloud computing where load balance are always a bottleneck. Yet this algorithm is not user friendly.

Task Queue algorithm, synchronously distributes the tasks, they target parallel processing strategy consisting of individual task and scheduling them in the share memory platform [25]. Centralized task queues are simple to implement, however they can quickly become a bottleneck as the number of processing elements increases.

Now a day schedulers have centralized Master-Slaves architecture, Slurm [27], Condor [28-29], PBS [30], SGE [31] are examples of this strategy where a centralized server is in control of the resource supplying and executing job. This architecture has functioned substantially in grid computing scales and coarse granular workloads [32], yet it is known of its poor scalability at the extreme scales of petascale systems with fine-granular workloads [33-34]. The answer to this issue is to move to the decentralized architectures that avoid utilising one component as a manager.

B. Distributed dynamic load balancing

Distributed strategies generally owns many load balancing processors. At the beginning, the master placed in the main work pool, distributes the work to various mini masters. Then the workload is distributed through each of these mini masters who have a cluster of processors beneath them. These extra mini masters are responsible for their share of the work pool Figure-13. A load balancing algorithm is ran by each mini master so that the work being done on the processors beneath them is managed. If the total work has not been allocated. More work could be requested from the master through these mini masters which is considered one of their responsibilities. They could be responsible for requesting more work from the master in case all total work has not been allocated.

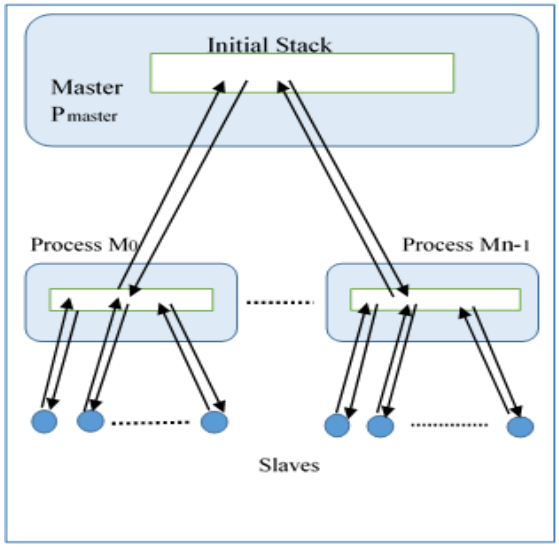


Figure-13. Distributed dynamic load balancing [25].

When utilising this strategy to load balance a system, what should be taken into consideration is the consequences of moving tasks among processors. On load imbalances bases and as the task is moved from one processor to another what should be also considered to actually move a task is the consequences of the communication and synchronization required. Like if there is a necessity to move huge amount of data when there is task migration from one busy processor to another idle processor. Communication and synchronization involved time may actually outbalance the benefits of moving the task in the first place.

Distributed balancing is generally implemented in either hierarchical [34] or fully distributed architectures [35] to cover the scalability problem. Utilising new architectures could deal with the potential single point of failure and make enhancements to the overall performance of the system up to a certain level. But only when tasks are distributed or when load balancing among the nodes problems can arise [36]. Table-8 summarises the important issues to be addressed and the drawbacks in both centralised and distributed load balancing.

Table-8. Comparison between centralised and distributed load balancing.

	Issues to be covered	**Drawbacks**
Centralized load balance (Very useful in small networks with low load). **Knowledge base**	Threshold policies. Throughput. Failure intensity. Communication between central server and processors in network. Associated Overhead. Single node or server is responsible for maintaining the statistics of entire network and updating it from time to time.	Not fault tolerant. Overloaded central decision making node.
Distributed load balancing (Very useful in large and heterogeneous environment) **Knowledge base**	Selection of processor that takes part in load balancing. Migration time Interprocessor communication. Information exchange criteria. Throughput Fault tolerance The entire processors in the network have the responsibility to load balancing store their own local database to achieve an efficient balancing decisions.	Algorithm complexity. Communication overhead.

DYNAMIC LOAD BALANCING ALGORITHEMS

The dynamic load balancing strategy gathers the information needed about the system state and the job information. Decisions are better made with short time as the utilised algorithm collects more information. The utilisation of dynamic load balancing in heterogeneous systems has a major effect on the efficiency of those complex systems. Those algorithms have the ability to deal with the many consisted nodes and their different speeds, different communication link speeds, different memory sizes and variable external loads due to the product.

Nearest neighbour algorithm

In the nearest neighbour algorithm and to do the load balancing, only the immediate neighbour of each processor is considered. According to the processors own load and the load information to its immediate neighbours,

a balancing decision is taken by the processor. Nearest neighbor load balancing algorithms rely on sequentially approximating the global uniform distribution, at each operation. In this algorithm the only concern is the direction of the migrated workload and the consequences of how to allocate excess workloads. There are a number of ways to select the direction of the migrated workload. Among them, researchers are interested in two simple representatives, the Diffusion (DF) and the Dimension Exchange (DE) methods. With diffusion method, a highly or lightly loaded processor balances its workload with all of its nearest neighbor simultaneously. [37] Utilised nearest neighbor algorithm with three typologies, which are 3D Mesh network, 2D Mesh network and chain topology. The comparison of all three results from simulation concluded was that 3D Mesh network performs better as compared to Chain and 2D Mesh network because this network take lesser time for stability, and that the transfer of load to neighbor processors are almost uniform and similar. Therefore it is better than chain and 2D networks.

Random (RAND) algorithm

At the moment where a workload at a certain processor is greater than threshold, migration take place to a neighbour chosen randomly. This algorithm does not implement any kind of checking for the state information of a node. It neither keeps any information of local load nor sends to the other processor any load information. Yet this algorithm is considered simple to design and easy to be carried out. However, it induces significant communication overheads because of the light loaded processor to the nearest neighbours that was randomly selected. [38]Demonstrated the effectiveness of the two phase method of scheduling, wherein task clustering is executed prior to the actual scheduling process. Task clustering finds the optimal or near-optimal number of processors on which to schedule the task graph. For this purpose they have utilised RAND among other algorithms. The foundations in terms of the runtimes of the cluster merging algorithms. RAND was regarded among algorithms known of fast performance. While it does not spend surplus time computing the clusters' workload (like in LB case), it takes significantly extra time to actually merging the clusters [38]. But RAND demonstrates fairly erratic behaviour and as a matter of fact it's considered the worst algorithm for either the one-phase or the two-phase method.

Adaptive contracting with neighbour (ACWN)

At the moment where the workload is newly generated, it migrates to the least loaded processorin the nearest neighbor. The load accepting processor remains the load in its local heap. As the load in its heap is less than its threshold load then there is no problem. Differently load is send to the neighbor processor which its load below the threshold load. Therefore, local load information and the neighbour load information is kept for periodically load exchange and this information is demanded in ACWN algorithm. Hence, the difference between RAND and ACWN is that ACWN fetches its target in the neighbour for the least loaded processor,

CYCLIC algorithm

This algorithm is the result of RAND algorithm after slender modification. To a remote system and in a cyclic style the workload is assigned. CYCLIC algorithm always maintains the information of the last destination where a process was sent. [39] Have utilised this algorithm in shared memory multiprocessor. They have resolved that CYCLIC algorithm is an effective load-balancing algorithm, rendering beneficial workload distribution and well locality of references. It assures good locality of references as commonly each task is executed on the same processor where it was previously executed, the data that was stored in cache memory will be reused. Also the tasks will be migrated only in the case of having load imbalance. Since CYCLIC finds very good workload distribution because the workload is balanced workload is correcting possible imbalanced workload and minimizing the number of migrated tasks to keep a good references locality. This conducts improving the execution times with regard to formal scheduling algorithms.

PROBABILISTIC

Here every node maintains two things, first the load vector and second the subset nodes load. The first art of the load vector is keeping the local load in which it's sent periodically to a node chosen in random. Thus information is revised in this way. The information could be spread in the network with no need for broadcasting. Nevertheless, in concern to as quality this algorithm is not ideal, it has poor extensibility along with delayed insertion.

Prioritized Random (PRAND) algorithm

To get the prioritized RAND (PRAND) and prioritized ACWN (PACWN) there were modifications to deal with the non-uniform workload. In these algorithms index numbers are assigned to workloads in order to weight their heaps. PRAND is same as RAND except that the second largest weighted load is chosen from the heap and transferred to a neighbour chosen in random. In the case of PACWN, it chooses the second largest weighted workload and then it is transferred to the least loaded neighbor.

THRESHOLD AND LEAST

Both of the algorithms get the partial knowledge through exchanging messages. In THRESHOLD a random node is chosen to accept a migrated load. The acceptance of the load depends on one condition, which if the load is below the threshold load. Otherwise, with another node polling is repeated until appropriate node is found and the load will be transferred. Then the process is executed locally when the number of attempts reaches its maximum and no proper recipient is reported. LEAST is an instant of THRESHOLD, where after polling least loaded machine is selected to receive the migrated load. Both THRESHOLD and LEAST have beneficial performance and considered

simple. Moreover, up-to-date load values are utilized by these algorithms. [40] Threshold based job allocation policy for heterogeneous systems was presented. In this, all incoming jobs are compared with the registered nodes and jobs being allocated according to their capability and threshold value. Load imbalance conditions are treated by Sender Initiative (Overloaded Node: Load > Threshold) or Receiver Initiative (Under loaded Node: Load < Threshold) Load Balancing. Algorithm and its analysis were also given. The main objective was to provide a Load Balancing mechanism so that overall performance could be maximized. THRESHOLD and LEAST provide beneficial performance results as considering their simplicity. Moreover, they utilize up-to-date load values, so it's unlikely to have bad location decisions. Actually, those bad decisions occur when disusing load information which is not the case in those algorithms. To summarise THRESHOLD and LEAST they both attain beneficial results when system load between machines is homogeneous. And this can be simply explained if the system load is homogeneous, a small subset of machines constitutes a representative sample: if there is no useable machine to be found the system will be globally loaded after a precise number of trials, when there is no use to keep on searching for an idle machine. However, it's likely to have, and at the same time heavily overloaded workstations while others are idle completely. In the aspect of having partial knowledge on the state of a global system it does not attain the demanded accuracy in heterogeneous load patterns where algorithms based on global knowledge adapts much better.

RECEPTION

In this algorithm overloaded nodes are randomly poled by nodes that have loads below the threshold load to migrate loads from them. Issues like bottlenecks and overloaded sights could be caused through the load distribution in a grid [41]. Where the Grid consists of clusters and each cluster is represented by a coordinator. At first each coordinator tries to load balance its cluster and if this attempt does not succeed, communication take place with the other coordinators to do either transfer or receive a load. This action is periodically repeated. Through simulation made by [41] analyses took place for important parameters of this protocol like correctness, performance and scalability. The outcome was that through diminishing the number of nodes with high loads in a grid environment, balance is achieved successfully in this algorithm.

Centralized information and centralized decision

CENTRAL is a subclass of this algorithm. The server in this algorithm is consequently informed by nodes with the existence or the absence of a light load node. A request is send to the server for a migration of a load from a heavily loaded node.

[42] Suggestedamodification on this algorithm. First that every machine periodically sends its load to the server [42]. [43] Addedto the previous modification that just in the case of when a significant change in the amount of the load, then the machine sends its load to the server. [44] Suggested that the maintenance should be in one shared file controlled through network file system. Before this modification, load information was held in memory by server process. [45] Mentioned that the server calls and through messages for load values periodically.

Generally, in a distributed system Centralized solutions suffers from two possible disadvantages. First is the occurrence of bottlenecks where the server may become one. The second issue is also caused by the server, causing crashing the system because of enlarging the recovery time.

Centralized information and distributed decision

In GLOBAL, the information is collected in a centralised manner while decision making is distributed. The server is responsible of broadcasting the load on the nodes. The over loaded processor fetches for a lightly loaded processor from the load vector that belongs to the overloaded one. So the migration of loads is not done through the server. This algorithm is considered more effective and extensile compared to CENTRAL, this is because it necessitates less number of messages. Moreover, while there are still availability of the process facility during the recovery of the server robustness is better, even with old load values.

In terms of performance GLOBAL performs less than LEAST [40]. This very much related to the larger amount of information utilised by GLOBAL when compared to LEAST. Merely this information is not up-to-date, where high frequency for gathering broadcasting load giving insufferable overheads. While in the LEAST case, it uses less (up-to-date) information but only on a set of machines.

Distributed information and distributed decision

The load situation of each node in this algorithm (OFFER) is periodically broadcasted. There is a global load vector kept by each node. Performance of this algorithm is poor where the mean process response time is larger than with GLOBAL for example. Some researchers proposed some variations, merely it was not sufficient to overcome the cost of systematic broadcasts.

RADIO

The RADIO algorithm it distributes both the information and decision where compulsion in broadcasting is inevitable. In this algorithm, lightly loaded nodes are kept in a list. This list will be distributed to all of the machines in the system, so that each one of them will have full knowledge of its successor and predecessor. Moreover, there is in RADIO a manger which is the head of the available list where every node is aware of. In this algorithm there is a manager that supervise the two types of migrations that take place. The two types of migration are either direct or indirect migration of a process from a heavily loaded node to the lightly loaded one.

The Shortest Expected Delay (SED) strategy

In this strategy the anticipated delay of each job completion is minimize by some efforts. The right way to select the distribution node will have its effect on minimizing the delay. This strategy is presented as a greedy approach, each job aiming to join the queue considers its best interest. This greedy attitude from the jobs can decrease the completion's expected delay and bring it to minimum. [46] Modified this algorithm where the term communication delay was introduced to the SED policy. The consumed time to transfer a job from the source node to the destination node is the communication delay

Another delay is considered in an attempt to make another modification on this strategy. This delay is caused by the limited size of the node's queue. The job will be sent to the best node to be served, but if the node's queue was saturated it will be sent to another best node. When this attempt fails in finding a place in a queue the job will return to the original node. The job will be temporarily stored in its original node's queue (if it's still saturated) waiting for interval of a time to make another attempt. The time delay where the job was wondering to find a place in a queue is also considered in the modified version of SED strategy.

The Never Queue (NQ) strategy

NQ policy is a separable strategy, where the cost of sending a job to be served is estimated before sending. Resulting, that the given job will reach its final destination or a subset of destinations where the job will be laid on the server with the minimum possible cost. That's why this algorithm will consider the fastest uncommitted server where the job is placed [46] Made modification on this algorithm. As NQ strategy demands to find always the fastest idle server. With the absence of an available idle server, the server with the shortest anticipated delay will be selected. Very similar to the SED strategy there is a delay that should be considered which is when a job finds a server with shortest expected delay despite the server's queue is full. The job will also goes to the second server with shortest expected delay. If the dilemma of facing saturated queues continue, it will return to its original server waiting temporarily for interval of a time (if its original server's queue is still saturated) and make another attempt after that. This delay should be considered in modified NQ strategy.

Greedy Throughput (GT) strategy

GT strategy differs from SED and NQ strategies. This strategy is called Greedy Throughput (GT) where it deals with the throughput of the system. The throughput of an algorithm is the number of completed jobs per unit time. This throughput would be maximum before the arrival of new job rather than maximizing only the throughput rate at the instant of balancing. [47] They have addressed this algorithm through considering the problem of job placement in load-sharing algorithms for large heterogeneous distributed computing environments. [47] Presented simulation results using a simple model; the results indicated that, under heavy loads, the usual policy of placing jobs where they will incur the shortest expected delay leads to inefficient system performance.

CONCLUSIONS

MPSoC systems will always be enhanced with new methods and strategies to boost the performance of the connected processors as long as there are demands for a processing system with high functionality. When heterogeneous and homogenous systems were compared it was obvious that the advantages of heterogeneous chip multiprocessors outweigh that of homogeneous chip multiprocessors. The advantages were in aspects of throughput, power consumption and the ability to minimize processor gate which counts through cutting unneeded features from each processor. The only hurdle is the lack of skilled programmers capable of writing parallelized applications, which will be likely defeated in the future as the demand for heterogeneous systems increases.

Reconfigurable computing based FPGAs brings many benefits that would enhance MPSoC making it more flexible and scalable with the ability to fix bugs and excluding the drawbacks in MPSoC based ASIC. It's obvious that FPGA platform is becoming the leader in reconfigurable computing, for its outstanding features and all the enhancements given through vendors.

In the matter of selecting the MPSoC architecture or the way to select a communicating method and finally how to transfer data, itisalways depending on the aim need to be achieved. Each of the presented methods has its own advantages and disadvantages. It's up to the designers to make choices towards achieving their gaols. In fact the designer of such complex systems is always in front of many methods and strategies to select from. The designer should always compromise between what he has and what he aims.

The second part of this paper discusses load balancing and its major influence on the performance of MPSoC. The highly demanded load balancing strategies are responsible on the evenly distribution of the workload across all connected processors in an attempt to improve the performance along with minimising the overheads. The important goal of bringing response time to maximum, is done through knowledgeable selection of a suitable load balance strategy which will bring waiting time to minimum. Through the survey of different load balancing algorithms it was clear that static load balancing algorithms have more stability than dynamic algorithms. But in distributed systems and in concern to which is more capable of having an accurate performance, dynamic load balancing slows preferable dealing with demanding systems than static load balancing algorithms.

REFERENCES

[1] Wolf W., Jerraya A. A. and Martin G. 2008. Multiprocessor system-on-chip (MPSoC) technology.

Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 27(10), 1701-1713.

[2] Dorta T., Jiménez J., Martín J. L., Bidarte U. and Astarloa A. 2010. Reconfigurable multiprocessor systems: a review. International Journal of Reconfigurable Computing. p. 7.

[3] Becchi M. and Crowley P. 2006, May. Dynamic thread assignment on heterogeneous multiprocessor architectures. In Proceedings of the 3rd conference on computing frontiers. pp. 29-40. ACM.

[4] Kumar A., Hansson A., Huisken J. and Corporaal H. 2007, April. Interactive presentation: An FPGA design flow for reconfigurable network-based multi-processor systems on chip. In Proceedings of the conference on Design, automation and test in Europe. pp. 117-122. EDA Consortium.

[5] Keller E. R. and Patterson C. D. 2004. U.S. Patent No. 6,725,441. Washington, DC: U.S. Patent and Trademark Office.

[6] Ghazanfari L. S., Airoldi R., Nurmi J. and Ahonen T. 2012, August. Reconfigurable multi-processor architecture for streaming applications. In Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on (pp. 477-478). IEEE.

[7] Nios I. I. Nios II Processor Reference Handbook. Altera-2003. ISO 690.

[8] Bonamy R., Chillet D., Sentieys O. and Bilavarn S. 2011, June. Towards a power and energy efficient use of partial dynamic reconfiguration. In Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2011 6th International Workshop on (pp. 1-4). IEEE.

[9] Shan Amar. 2006. Heterogeneous Processing: a Strategy for Augmenting Moore's Law. Linux Journal.

[10] Mittal S. and Vetter J. S. 2015. A survey of CPU-GPU heterogeneous computing techniques. ACM Computing Surveys (CSUR). 47(4): 69.

[11] Kumar R., Tullsen D. M., Ranganathan P., Jouppi N. P. and Farkas K. I. 2004. Single-ISA heterogeneous multi-core architectures for multithreaded workload performance. ACM SIGARCH Computer Architecture News. 32(2): 64.

[12] Balakrishnan S., Rajwar R., Upton M. and Lai K. 2005, June. The impact of performance asymmetry in emerging multicore architectures. In ACM SIGARCH Computer Architecture News. 33(2): 506-517. IEEE Computer Society.

[13] Lee H. G., Chang N., Ogras U. Y. and Marculescu R. 2007. On-chip communication architecture exploration: A quantitative evaluation of point-to-point, bus, and network-on-chip approaches. ACM Transactions on Design Automation of Electronic Systems (TODAES). 12(3): 23.

[14] Verkest D., Van Rompaey K., Bolsens I. and De Man, H. 1996. CoWare-A design environment for heterogeneous hardware/software systems. Design automation for embedded systems. 1(4): 357-386.

[15] Agarwal A., Iskander C. and Shankar R. 2009. Survey of network on chip (noc) architectures & contributions. Journal of engineering, Computing and Architecture. 3(1): 21-27. ISO 690.

[16] Gropp W., Lusk E. and Skjellum A. 1999. Using MPI: portable parallel programming with the message-passing interface (Vol. 1). MIT press.

[17] Zea N., Sartori J. and Kumar R. 2008. Servo: a programming model for many-core computing. ACM SIGARCH Computer Architecture News. 36(2): 28-37.

[18] Chapman B., Jost G. and Van Der Pas R. 2008. Using OpenMP: portable shared memory parallel programming (Vol. 10). MIT press.

[19] Stone J. E., Gohara D. and Shi G. 2010. OpenCL: A parallel programming standard for heterogeneous computing systems. Computing in science and engineering. 12(1-3): 66-73.

[20] Abubakar H. R. and Aftab U. 2004. Evaluation of load balancing strategies. In National Conference on Emerging Technologies. p. 67.

[21] Sharma S., Singh S. and Sharma M. 2008. Performance analysis of load balancing algorithms. World Academy of Science, Engineering and Technology. 38: 269-272.

[22] Grosu D. and Chronopoulos A. T. 2005. Noncooperative load balancing in distributed systems. Journal of parallel and distributed computing. 65(9): 1022-1034.

[23] Salehi M. A., Deldari H. and Dorri B. M. 2009. Balancing load in a computational grid applying adaptive, intelligent colonies of ants. Informatica. 33(2).

[24] Khan R. Z. and Ali M. F. 2014. An Efficient Diffusion Load Balancing Algorithm in Distributed System. International Journal of Information Technology and Computer Science (IJITCS). 6(8): 65.

[25] Pandey M. S. K., Tiwari R. and Bhilai C. 2013. The Efficient load balancing in the parallel computer. International Journal of Advanced Research in Computer Engineering and Technology (IJARCET). 2(4): 1507.

[26] Zaki M. J., Li W. and Parthasarathy S. 1996, August. Customized dynamic load balancing for a network of workstations. In High Performance Distributed Computing, 1996. Proceedings of 5th IEEE International Symposium on. pp. 282-291. IEEE.

[27] Anitha R. and Kavitha V. 2015. Hierarchical Model with Future Prediction Algorithm. Int. J. Adv. Eng. 1(3): 354-358.

[28] Yoo A. B., Jette M. A. and Grondona M. 2003, June. Slurm: Simple linux utility for resource management. In Job Scheduling Strategies for Parallel Processing (pp. 44-60). Springer Berlin Heidelberg.

[29] Thain D., Tannenbaum T. and Livny M. 2005. Distributed computing in practice: the Condor experience. Concurrency and computation: practice and experience. 17(2-4): 323-356.

[30] Frey J., Tannenbaum T., Livny M., Foster I. and Tuecke S. 2002. Condor-G: A computation management agent for multi-institutional grids. Cluster Computing. 5(3): 237-246.

[31] Bode B., Halstead D. M., Kendall R., Lei Z. and Jackson, D. (2000, October). The Portable Batch Scheduler and the Maui Scheduler on Linux Clusters. In Annual Linux Showcase & Conference.

[32] Gentzsch W. 2001. Sun grid engine: Towards creating a compute power grid. In Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on. pp. 35-36. IEEE.

[33] Dumitrescu C. L., Raicu I. and Foster I. 2005. Experiences in running workloads over grid3. In Grid and Cooperative Computing-GCC 2005 (pp. 274-286). Springer Berlin Heidelberg.

[34] Raicu I., Zhang Z., Wilde M., Foster I., Beckman P., Iskra K. and Clifford B. 2008, November. Toward loosely coupled programming on petascale systems. In Proceedings of the 2008 ACM/IEEE conference on Supercomputing. p. 22. IEEE Press.

[35] Raicu I., Zhao Y., Dumitrescu C., Foster I. and Wilde M. 2007, November. Falkon: a Fast and Light-weight tasK executiON framework. In Proceedings of the 2007 ACM/IEEE conference on Supercomputing. p. 43. ACM.

[36] Melnik S., Gubarev A., Long J. J., Romer G., Shivakumar S., Tolton M. and Vassilakis T. 2011. Dremel: interactive analysis of web-scale datasets. Communications of the ACM. 54(6): 114-123.

[37] Diekmann R., Frommer A. and Monien B. 1999. Efficient schemes for nearest neighbor load balancing. Parallel computing. 25(7): 789-812.

[38] Liou J. C. and Palis M. A. 1997, April. A comparison of general approaches to multiprocessor scheduling. In Parallel Processing Symposium, 1997. Proceedings. 11th International. pp. 152-156. IEEE.

[39] García-Dopico A., Pérez A., Rodríguez S. and García M. I. 2013. CYCLIC: A Locality-Preserving Load-Balancing Algorithm for PDES on Shared Memory Multiprocessors. Computing and Informatics. 31(6): 1255-1278.

[40] Saxena A. B. and Sharma D. 2011. Analysis of threshold based centralized load balancing policy for heterogeneous machines. International Journal of Advanced Information Technology. 1(5): 39.

[41] Resat Umit Payli, Kayhan Erciyes, Orhan Dagdeviren. 2011. Cluster-Based Load Balancing Algorithms for Grids. International Journal of Computer Networks & Communications (IJCNC). 3(5).

[42] Hagmann R. B. 1986, May. Process Server: Sharing Processing Power in a Workstation Environment. In ICDCS. pp. 260-267.

[43] Theimer M. M. and Lantz K. A. 1989. Finding idle machines in a workstation-based distributed system. Software Engineering, IEEE Transactions on. 15(11): 1444-1458.

[44] Nichols D. 1987, November. Using idle workstations in a shared computing environment. In ACM SIGOPS Operating Systems Review. 21(5): 5-12. ACM.

[45] Litzkow M. J. 1987, June. Remote Unix: Turning idle workstations into cycle servers. In Proceedings of the Summer USENIX Conference. pp. 381-384.

[46] Kabalan K. Y., Smari W. W. and Hakimian J. Y. 2002. Adaptive load sharing in heterogeneous systems: Policies, modifications, and simulation. International Journal of Simulation, Systems, Science and Technology. 3(1-2): 89-100.

[47] Weinrib A. and Shenker S. 1988, March. Greed is not enough: adaptive load sharing in large heterogeneous systems. In INFOCOM'88. Networks: Evolution or Revolution. Proceedings of 7th Annual Joint Conference of the IEEE Computer and Communications Societies. pp. 986-994. IEEE.