International Journal of VLSI System Design and Communication Systems

ISSN 2322-0929
Vol.01,Issue.01,
Agust-2013,
Pages:35-44

www.semargroups.org

TDM Arbitration and Virtual Point to Point Connection in Mesh Networks

G.KAVITHA
Research Scholar, PBR VITS, Nellore, AP-India, Email: Kavitha.gottipati1@gmail.com,

Abstract: Processor allocation in many core MPSoCs is a challenging task, especially when the order and requirements of incoming applications are unknown during design stage. To enhance the performance of network, balance the workload across processing cores or mitigate the effect of hot processing elements in thermal management methodologies. Runtime task migration was first proposed in multicomputer with load balancing as the major objective. Using on-chip interconnection networks in place of ad-hoc global wiring structures the top level wires on a chip and facilitates modular design. With this approach, system modules (processors, memories, peripherals, etc...) communicate by sending packets to one another over the network. Specific NoC properties such as amount of communication buffers, sensitivity to implementation complexity, latency and power consumption constraints bring new challenges in using task migration mechanisms in NoCs. This paper involves efficient methodologies based on virtual point-to-point (VIP for short) connections. These connections provide low-latency and low-power paths for heavy communication flows created by task migration mechanisms. The structured network wiring gives well-controlled electrical parameters that eliminate timing iterations and enable the use of high-performance circuits to reduce latency and increase bandwidth. The area overhead acquired to implement an on-chip network is modest, power savings can be achieved compared to the previously proposed task migration strategy (known as Gathering-Rout-Scattering) for mesh multiprocessors. Silicon nano-photonics is an emerging technology platform for offering high-bandwidth connectivity with extreme energy efficiency for future networks-on-chip. To gain, the low transmission energy and high bandwidth density of waveguides in end-to-end communication as well as in WDM, perform circuit switching as an arbitration mechanism. However, pure circuit-switching requires an electronic control network with heavy loads, low network utilization, and an overhead in power consumption.This paper introduces the concept and design of on-chip networks, and discusses some challenges in the architecture and design of these networks.

Keywords: MPSoCS, Power, Task migration, Virtual point- to- point connection.

I. INTRODUCTION

Old on-chip interconnect schemes, including point-to-point links and shared buses, face major challenges as the number of IP cores increases. The point-to-point links draws the poor performance due to poor scalability and vast area overhead, while this connection scheme presents ideal delay and power profiles. Due to unpredictable latency and increased power, the applicability of shared buses to future many core SoCs (with tens to hundreds IP cores) is also under question. To mitigate area overhead and increase efficiency of on-chip interconnect, packet switched networks-on-chip (NoCs) are extensively employed and evaluated in current multiprocessor systems-on-chip [3,5]. Semiconductor with advanced technology has facilitated for very complex large scale systems on a chip (SoCs) designs. The generation of each new SoC integrates more processing elements (IPs) along with increased functionality. The traditional interconnects, like busses become a bottleneck as and when the number of IPs increases. Networks-on-chip (NoCs) are a modular, scalable interconnect solution. Concrete examples of NoCs are Æthereal, Xpipes, QNoC and Mango. Currently, they tend to become the preferred interconnect solution for large scale inherently multiprocessor SoCs. However, NoCs require sophisticated tools to aid in design-time decisions.

Due to dynamic variation of application behavior, it seems that even optimal mapping may not meet best performance. This necessitates some optimization approach considering such variations. Task migration is such an important mean for load balancing over network resources and thus improving the performance which can efficiently trace dynamic variation of workload

specifications. Simply stated, task migration is transferring a job, task, or process from a core where it is currently running to another core and then resuming its execution there; such that overall system performance is improved with minimum cost overhead. Moreover, due to cost and performance issues, it is desirable to implement embedded systems in a single chip System-on-Chip (SoC), with the use of heterogeneous components such as CPUs, memories and buses. In addition, it is possible that a SoC is composed by more than one processing element (PE), being known as Multiprocessor System-on-Chip (MPSoC). Thus, the requirements and implementation characteristics needed by a given application must be taken into account during the development of an MPSoC causing that customized architectures be designed.

The main goal of this work is presenting different alternatives of architectural support for dynamic load balancing techniques, such as task migration, concerning real-time MPSoCs. It might be possible to distribute the system's load dynamically and homogeneously, and avoiding execution overload, through these techniques. It is mainly due to execution overloads that many permanent failures of the system occur, like electro migration, stress migration and dielectric breakdown. Avoiding overloaded spots may increase the application performance as well as delaying different types of failures.

II. PREVIOUS WORK

To give a flavor for on-chip interconnection networks this section sketches the design of a simple network. Consider a 12mm x12mm chip(figure 1) in 0.1 m CMOS technology with an 0.5 m minimum wire pitch. As shown in Figure 1, we divide this chip into 16 3mm x 3mm tiles. A system is composed by placing client logic (e.g., processors, DSPs, peripheral controllers, memory subsystems, etc.) into the tiles. The client logic blocks communicate with one another only over the network. There are no top-level connections other than the network wires.

The simple network employs virtual-channel flow control [2][5]. Details of an input controller and output controller are shown in Figure 3. Each input controller has an input buffer and input state logic for each virtual channel. When a head flit arrives at each tile, the input controller strips the next entry off the route field and uses these two bits to select one of four output ports. The flit then arbitrates with the other virtual channels on that input port and, if it wins the arbitration, is forwarded to the output port. This arbitration and forwarding takes place in parallel with allocating a virtual channel and checking available buffer space to reduce latency. The output port provides a single stage of buffering for each input port connection. The flits in these buffers arbitrate for the link to the input controller on the next tile. Credits for buffer allocation are piggybacked on flits travelling in the reverse direction.

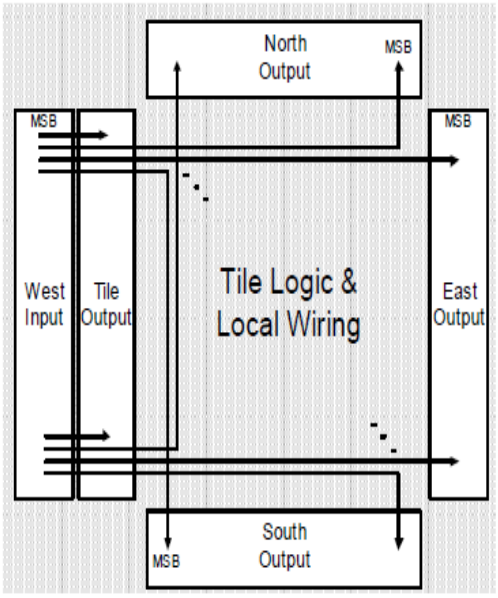


Figure 1: A 12mm X 12mm chip.

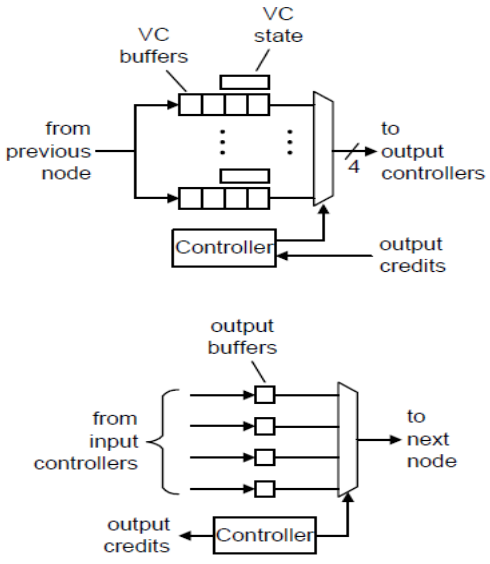


Figure 2: Input and output controllers.

In our example network, the wires are driven at the same frequency as the input and output controllers.

This choice allows for the simplest possible controller logic, at the expensive of more global wiring resources. An alternative architecture could easily drive the network wires at several times the router frequency, taking advantage of their available bandwidth. This approach would reduce global wiring usage with the addition of slightly more controller logic.

III. PROPOSED WORK

Common approaches in system level design methodology use graph-based description of the studied architecture. An MPSoC with a 2D mesh can be denoted as Mesh-NoC=(S,R,BW,V,B) , where S(X,Y) is the mesh size (in X and Y co-ordinates), R(S,D) defines the routing algorithm that gives the order of intermediate routers met by a message from source 'S' to destination 'D'. BW and V represent the maximum bandwidth and virtual channels associated to physical channels, and B is the buffer size of input virtual channels.

Task migration between two sub meshes SM[(X1,Y1),(X2,Y2)] (where (X1,Y1) and (X2,Y2) are respectively the coordinates of bottom-left and top-right corners of the source sub mesh) and SM_[(X3,Y3),(X4,Y4)](where (X3,Y3) and (X4,Y4) are respectively the coordinates of bottom-left and top-right corners of the destination sub mesh) in a 2D mesh NoC is defined as movement of tasks of any IP core in SM to the corresponding IP core in SM_ while other IP cores execute their tasks with no stalled communications. The aim is to reduce the overhead of migration to the normal communications of other IP cores (not involved in the migration). Migration overhead includes migration time and the power consumed for migration that consequently increase the ultimate network latency and power consumption.

Let us define a parameter called average migration latency (AML) as the average time interval between the initiation of a migration and the reception of last migration flits in the destination sub mesh. An ideal task migration scheme between two sub meshes must pose the minimum collision between migration traffic and the normal communication traffic of the application (normal messages have source and destination nodes not residing in the source and destination migration sub meshes).At last, we employ a packed-switched NoC architecture[2] that can provide a number of low-power and low latency dedicated virtual point-to-point (or VIP, for short) connections between any two nodes by bypassing the pipeline of the intermediate routers.

In this paper the architecture of each router involves a multiplexer-tree-based crossbar fabric. For each router of physical channel, one virtual channel (VC0) is dedicated to enable bypassing router pipeline stages. The VIP buffer in each router has a signal called full and it is set to 1 when a flit enters the buffer to be serviced. The full signal is also used to control the arbiter and allocates the output port to the input port of the VIP in the buffer. Each tile interfaces with the network over an input port that is used to insert packets into the network and an output port over which the network delivers packets to the tile1. The input port consists of a 256-bit data field and a control field with the following sub fields:

- Type (2 bits): encodes whether the flit (flow control digit) on the data port is the start of a new packet (head), the continuation of a packet (body), the end of a packet (tail), or an idle cycle (idle). Note that a flit may be both a head and a tail. Amulti-flit packet is inserted by inserting a head flit, zero or more body flits, and a tail flit.
- Size (4 bits): logarithmically encodes the size of the data in the data field from 0 (1 bit) to 8 (256 bits). When a short data field is sent the size field prevents the unused bits from dissipating power.
- Virtual Channel (8 bits): bit mask that specifies which of eight virtual channels this packet may be routed on. This mask identifies a class of service that may be in progress. Though, the injection of a long and low priority packet may be spitted in to short, high-priority packet and then resumed.
- Route (16 bits): A source of the route required two bits for each hop (left, right, straight, or extract). The destination of the route may be one of the sixteen tiles or special network clients including I/O pads and internal network registers. This field is only used on a head flit and can be used to carry data on non-head flits.
- Ready (8 bits): a signal from the network back to the client indicating that the network is ready to accept the next flit on each virtual channel.

Although load balance techniques for general purpose parallel computing have already been studied in some previous works [3], [5] let's focus here on the works concerning the embedded computing issues. In spite of having some similarity with general purpose parallel computing, architecture shave multiple processing elements and MPSoCs present many different challenges within their own design time. These challenges makes even more difficult to implement efficient task migration mechanisms. Nollet et al. [1] proposes a technique that using the processors debug registers, to deal with the initial overhead of a heterogeneous MPSoC task migration. The use of end-to-end monitors collecting network interface statistics is proposed in order to assist the operating system

controlling the NoC. The work focuses on the use of such performance monitors to optimize communication resource usage.

As a motivation example, consider a real-world scenario shown in Figure 3. The scenario considers a 6×7 mesh-based MPSoC running fft application of SPLASH-II benchmark suit onto 21 IP cores (shown in orange). Figure 1.a assumes that lu tasks are interleaved in the free space of the MPSoC without migrating the fft tasks. However, Figure 3.b assumes that fft tasks are first migrated such that a contiguous area of free resources is formed on which lu tasks are mapped. Using simulation (described in section V), the average message latency and total network power when migration is not applied (Figure 3.a) are 28.83 cycles and 5.72 nJ/cycle, respectively. When migration is applied (Figure 3.b), the average message latency and total network power are reduced to 23.78 cycles and 5.1 nJ/cycle. Therefore, using migration can lead to about 13% improvement in message latency and 11% reduction in network consumption power.

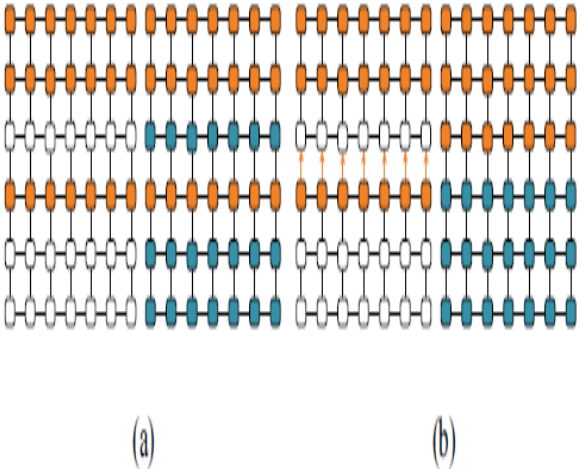


Figure 3: An example of task migration in MPSoC.

Several task migration mechanisms have been proposed for mesh multi-computers [5]. In this paper[5],two migration schemes has proposed. One called Diagonal scheme that exploring all disjoint paths in one phase to migrate a task based on XY routing. The second scheme, called Gathering-Routing-Scattering that reduces the number of task migration paths by collecting the tasks in a limited number of nodes in the source sub mesh and send them to the respective cores in the destination sub mesh which will then scatter the tasks to their final destinations. Two task migration schemes namely ODC-SC and ODCFC are presented in [3][5]. They try rearranging active tasks in the mesh such that a larger contiguous area of free nodes will be available for future allocations. These methods concentrate on locating a destination sub mesh in such a way that when a task migrates towards that sub mesh, it creates the minimum interfere with other tasks migrating in the network.

Task migration in MPSoC context has been rarely addressed in recent years. They mainly concentrate on the mechanism of starting migration on a source core and resuming the task on the destination core. Run-time task migration is addressed in NoCs based on code check pointing. They assume that whenever the user changes requirements or in the case of a mapping failure, the resource management can use run-time task migration for reallocation of resource. The evaluation concerns only performance while ignoring power consumption. Authors in [4] propose a lightweight user-managed migration scheme based on code check-pointing to improve controllability and predictability which are critical features.

The remainder of this paper describes our initial thoughts on the design of on-chip interconnection networks. To provide a baseline, in this section we sketch the design of a simple on chip network. Section 3 revisits the design choices made in this simple network and discusses the challenges and open research issues in the design of such networks. In this paper, we concentrate on the overhead caused by migrating tasks when they are transferring from a source sub mesh to the destination sub mesh and evaluate its impact on the overall performance and power consumption. To reduce task migration time and power overhead, some point-to-point paths are established for heavy loaded task migration paths. These point-to-point connections are prioritized over normal data flows to service migration process quickly.

The logic of the virtual channel router is very simple, and a few thousand gates along each edge of the tile. Hence the area of the router is dominated by buffer space. If we provide four flits (300b per flit) of buffering for each of the eight virtual channels (with overhead), the total buffer requirement is about 104 bits for each edge of the tile. We estimate the logic, driver and receiver circuits, buffer storage, and routing will occupy an area less than 50m wide by 3mm long along each edge of the tile for a total overhead of 0.59mm2 or 6.6% of the tile area. In addition to this area, the router also uses about 3000 of the 6000 available wiring tracks on the top two metal layers for routing differential signals and shields2. This ‘overhead’ replaces the drivers, receivers, repeaters, and wires for global signals. Thus, the net effect on a design will be substantially less than this and may even be positive.

The simple network described above uses two methods to exploit the large available pin count. First, a wide (almost 300-bit) flit is sent broadside across router channels to use the maximum possible number of pins. In contrast, most inter-chip routers use relatively narrow channels (8-16 bits) to fit into the tight pin constraint of a chip package. Second, a folded torus topology is employed. This topology has twice the wire demand and twice the bisection bandwidth of a mesh network. Hence, it effectively converts some of the plentiful wires into bandwidth. In general, the folded torus topology trades a longer average flit transmission distance for fewer routing hops. While these choices of wide, broadside flits and a folded topology increase the wire utilization, much room for improvement remains. There are many alternative topologies and the choice of a topology depends on many factors. For example, if power dissipation is critical, a mesh topology may be preferable to a torus. Simple expressions illustrate the trade-off of power between the folded torus and a traditional mesh where k is the radix of the network (k = 4 in our 16 tile example):

$$P_{total} = hops \cdot P_{hop} + dist \cdot P_{wire}$$

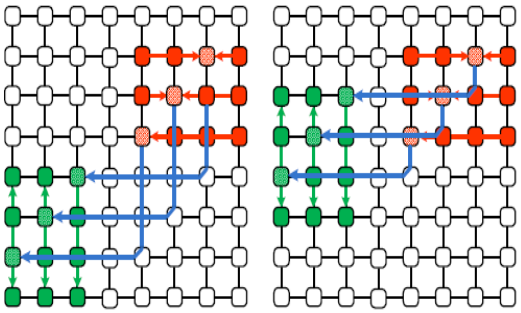
$$P_{mesh} \approx k \cdot \left(\frac{2}{3} \cdot P_{hop} + \frac{2}{3} \cdot P_{wire}\right)$$

$$P_{torus} \approx k \cdot \left(\frac{1}{2} \cdot P_{hop} + \frac{k-1}{k} P_{wire}\right) \quad (1)$$

The proposed low-power and low-latency task migration strategy is based on Gathering-Routing-Scattering algorithm [3] and uses VIPs [5] for the paths involved in task migration to reduce the average migration latency (AML), the total mean network latency and power consumption. To optimize the number of task migration paths, a set of IP cores at the source sub mesh is selected, and same applied to gather the tasks of the IP cores that are in the same row or column. In the same way at destination sub mesh, limited IP cores are used as the intermediate destination nodes of task migration process that further scatter the tasks to their final destination nodes. Since using dimension order routing (or XY routing algorithm), qualified intermediate source and destination cores have to reside in a diagonal arrangement to mitigate possible congestions between different task migration paths, Figure 4 shows two task migration scenarios.

In Figure 4(a), task migration from a source sub mesh (with red cores) to a destination sub mesh (with green cores) that have no overlap in X and Y dimensions is illustrated. Figure 4(b) demonstrates a typical scenario in which source and destination sub meshes are overlapped (in Y dimension, here). The Gathering process at the source sub mesh is shown by red arrows toward selected diagonal cores (hatched cores in red) and scattering process at the destination sub mesh is shown by green arrows from selected diagonal cores (hatched cores in green). The blue arrows represent constructed VIP paths (based on XY routing) between selected diagonal cores in the source and destination sub meshes. Note that paths in the figure are denoted with blue, Scattering phase is identified with green, and hatched cores represent selected diagonal IP cores.

The employed packet-switched NoC architecture [5] uses a setup network to control VIPs establishment. If there is no sufficient free resources for a new VIP connection request, that new VIP can tear down old VIPs. A new VIP is given a weight based on the channels required.



(b)

Figure 4: Different Phases of proposed task migration process

A new VIP can be formed if its weight is greater than the cumulative weight of VIPs to be torn-down. In the proposed task migration strategy, we prioritize VIP connections formed for task migration over other possible VIPs built for normal high volume communication flows. Thus, when setting up a task migration VIP, we let the normal VIP connections (if exist) that disturb immediate migration of the tasks, be torn-down. Note that prioritizing VIP messages over normal messages may lead to starvation at intermediate routers along a typical VIP. To mitigate these potential starvations, each router along the VIP counts the number of consecutive cycles on which the data of a VIP connection are serviced. When this counter meets the threshold Tvip, it stalls VIP message transmission in the upstream VIP latches and resumes normal messages transmission over packet-switched network for at most Tps cycle s, after which VIP transmission is allowed to be resumed. To shorten the migration time, we allocate more bandwidth to migration VIPs compared to the

normal messages (transferred over the packet-switched part). To this end, Tvipis chosen much larger than Tps. This guarantees that the number of consecutive cycles belonging to VIP connection never exceeds this threshold and hence the chance of un-interrupted migration is increased.

A. NoCs and Ethereal

NoCs comprise two components: routers (R) and network interfaces (NI), as depicted in Figure 5. The routers can be randomly connected among themselves and to the NIs (i.e., there are no topology constraints). Note that in principle there can be multiple links between routers. The routers transport packets of data from one NI to another.

The NIs enables end-to-end services to the IP modules and key in decoupling computation from communication [2]. The NI allows the designer to simplify communication issues to local point-to-point transactions at IP module boundaries, using protocols natural to the IP. They are responsible for, implementing the connections and services, and for offering a standard interface (e.g., AXI or OCP) to the IP modules.

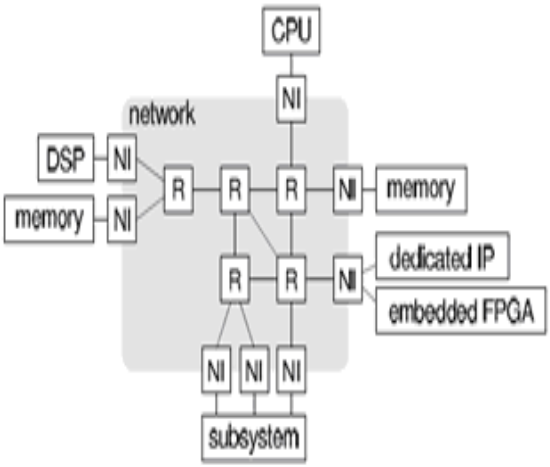


Figure 5: Example NOC.

We use the Æthereal NoC [4, 5] as an example for our work. The Æthereal NoC runs at 500 MHz and offers a raw link bandwidth of 2GB/s in a0.13µm CMOS technology. It is supported by state of the art design time decision tools [1][2].Æthereal offers transport-layer communication services to IPs, in the form of connections, comprising best-effort (BE) and accurate throughput(GT) services. This can be obtained by means of TDMA slot reservations in NIs. Æthereal NoC instances are flexible to reconfigure at run-time. This is achieved by programming the NIs using standard memory-mapped I/O ports.

The current setup uses centralized programming of the NoC and source routing. The Ethereal NoC allows the mapping of potentially multiple IPs per NI and potentially multiple NIs per router with any topology. The interconnected IPs interacts with each other, which are read and write operations from IPs called it as transactions.

The proposed approach results in rapid task migration through the following mechanisms: 1) Each IP core on the diagonal line of the source sub-mesh gathers tasks of the cores in the same row or column, and send them over prioritized VIP connections. Thus, task migration messages are much larger than normal messages which reduces the delay overhead of repetitive VIP establishments. 2) During migration, VIP cycle threshold, Tvip, is chosen much larger than packet-switched cycle threshold, Tps. So, the long migration messages are scarcely interrupted by normal messages. 3) Migration VIPs are prioritized over other potential VIP connections in the network. This reduces VIP establishments delay for migrations and guarantees undesirable interrupts from normal VIPs. As VIP circuits are dynamically reconfigurable and can be configured based on the traffic pattern of the currently running application(s), it would be performance- and cost efficient to have NoC-based MPSoCs running multiple applications which support VIP connections. Hence, by reconfiguring VIP connections, the proposed mechanism can support frequent task migrations occurred when running multiple applications on an MPSoCs.

B. Load balancing

Load balancing issues have been addressed for many years in general purpose distributed and parallel systems [3] [4]. Considering it, it is important to implement the concept of load balancing, which is the amount of work among several nodes of a given group of processing nodes. The main motive is to assure that none or at least the minimum possible number of the nodes neither are overloaded nor under loaded. The main advantages in assuring a balance of the total system workload are having a good utilization of all the system nodes, improving overall performance and minimizing communication delays.

Load-balancing mechanisms can be classified into either global or local policies. Local policies organize each node of the system individually and its own

scheduling issues, while global policies involves into account decisions such as where to execute a process in the available nodes. Furthermore, there is a other main part of the taxonomy, by considers both static and dynamic policies. They regard the time at which the scheduling or assignment decisions are made.

Static load balancing is a method in which depending on the nodes' performance the workload is distributed in the beginning of the system execution. On the other hand, dynamic load balancing determines the distribution of workload at run-time. Based on the recent survey, the master of the system performs the assignment of new tasks to the remaining members. There are many dynamic load-balancing algorithms proposed. They have four basic steps in common:

• Load monitoring, that is monitoring system's components performance;
• Synchronization, that is exchanging this information among nodes;
• Rebalancing criteria, that is calculating new distributions and making the work movement decision;
• Task migration or actual data movement. Regarding to current OSs, like the Linux 2.6 kernel, it is notable their capability of utilizing the power allow by a shared memory architecture, either loosely or tightly coupled.

In RTOS systems, load-balancing mechanisms are desirable due to the same advantages stated above. Here, one important aspect is to maintain transparency of the load balancing mechanisms and the method for achieving it is to configure the RTOS. Through this transparency individual threads can be assigned to run on specific processors based on its availability. Doing so, the processing load can be shared among processors with work automatically assigned to a free processor. The RTOS determines the processor whether it is free or not and, if it is true, a thread can be run on that processor even though the remaining of the system may already be running other threads. In addition, once the RTOS scheduler is designed, priorities consideration is also important to maintain priority execution of all threads. So, the high priority threads are processed before low priority threads. Priority and preemptive schedulers uses multiple cores to run threads when they are in ready state. The threads on available cores are run by scheduler automatically.

IV. TDM ARBITRATION

TDM proposed to arbitrate end-to-end photonic circuit paths in a network of ring-resonator based photonic switches. The basic concept behind this is as follows: during a specified amount of time, or time slot, switches in the network are configured to allow communication between one or more pairs of access points. The length of each time slot is identified as follows.

$$tslot = tsetup + ttransmission + tpropagation$$

Where, tsetup is the time to change the state of aring resonator, ttransmission is the amount of time taken by each node that allowed to transmit data per time slot, and tpropagation is the extreme bad propagation latency between any two valid communicating pairs. If each switch is able to keep track of the current time slot using a global clock, it can be made aware of its correct configuration using control registers for any given timeslot. This allows the control of the switches to be completely distributed.

This approach should be distinguished from TDM in other networks. Usually, all requests to use network resources are arbitrated by sources or individual network nodes to allocate dynamic temporal schedule for usage of virtual channels, links, switches, or virtual circuits, thus providing fairness guarantees to latency and bandwidth. Our method aims at providing the same fairness, but because there is no practical equivalent to a buffer implementation in silicon photonic integrated technology, we must apply TDM arbitration through the entire network by creating end-to-end optical circuit paths. Here, the scheduling of nodes access to network resources is done statically, at design-time. If there are 'nslot' time slots, each of duration 'tslot', then the total TDM period, TTDM is

$$TTDM = nslot \times tslot$$

Our design stipulates that full network communication coverage must be implemented, or that every network node is able to send messages to every other node within TTDM. In this arbitration scheme, the network repeats the cycles through every scheduled time slot. If a network node contains data to send to another node, it waits for the correct time slot to send. If a node has multiple messages to different destinations queued up, it can send them out of order. Moreover, by selecting different values statically for ttransmission, we can vary the granularity of the arbitration. If, for instance, the system architecture specifies that only fixed-length messages may be sent on the network (i.e. cache lines), then we can adjust ttransmission to exactly match that size. The naive way to accomplish the resource scheduling is to assign a single time slot to every communicating pair in the network. Thus, we would require

$$Nslot = N \times (N - 1)$$

time slots to implement full coverage, where N is the number of nodes in the network. Therefore a 64-node network would require 4032 time slots. This naive scheduling of one transmission per time slot in the network achieves the worst-case network utilization. As we will see, it is possible to statically allocate the network to many transmissions during a single time slot.

TDM Scheduling using Genetic Algorithm

We can improve efficiency of this approach on the naive implementation by scheduling more than one transmission per time slot, thus reducing the total number of time slots, and the worst latency of a message waiting for its slot. So that, to maintain appropriate operation we must adhere to the following constraints during a single time slot:

1) Source contention - A node can only send to one destination, assuming a single set of modulators at an access point.
2) Destination contention - A node can only receive from one source, assuming a single set of detectors at an access point.
3) Topology contention - Transmission cannot overlap in the same waveguide.

Thus we are presented with the problem of scheduling in both time and space at least one transmission from every node to every other node in the network. We can accomplish this by searching the solution space using a genetic algorithm.

Genetic algorithms attempt to optimize the solution to a problem by simulating the process of evolution on a population of valid solutions. Starting with an initial population, genetic algorithms iteratively apply selection, reproduction, and mutation methods until a specified number of generations have been simulated, or a proper solution has been found. Candidate solutions to the problem must have a fitness function, which allows the comparison of members of the population. In our case, a solution is a set of time slots containing which collectively implement full network communication coverage. We define the following methods according to problem:

a) Initialization: Here the initial population is generated from the naive solution by merging two time slots randomly. Valid merges must adhere to source, destination, and topology constraints throughout the genetic algorithm process. This is done until the population contains P valid solutions.
b) Selection: We employ tournament selection for ease of implementation [5]. A tournament is begun by selecting a subset kTS of the population at random. Fitness is measured for each solution, which is the number of timeslots in the solution (lower number = better fitness). The best solution is chosen with probability pTS, the second best is chosen with probability pTS×(1−pTS), the third best is chosen with probability pTS× (1 − pTS)2, and so on. Another tournament is started until the selection reaches a proportion pSof the population size P. This selection process has a higher chance of selecting the more relevant solutions, but can still there are some that are not as fit to maintain population diversity.
c) Reproduction: After selection, two solutions are chosen at random to mate, producing one child. This is done according to the pseudo-code in Algorithm 1. To form a child, two parents cycle through every possible communication, taking the larger time slot that contains each. The child's Add function filters out redundant communications. In this way, the reproduction process favours more dense time slots. This process continues until the population count is back to P.
d) Mutation: After the next generation is formed, each solution in the population has a chance pm to undergo the mutation process. Mutation first "unrolls" a random timeslot in the solution, forming a new time slot for each communication contained in it. Then, the solution is iterated over attempting to combine every time slot combination, thereby spreading the unrolled communication back among the other time slots. We test our algorithm by running it multiple times for a mesh topology. Table I shows the parameters to the algorithm. Table II summarizes the results for N=16, 36, and 64.Initial experimentation indicated how many generations were necessary before solutions converged for each network size.

Algorithm 1 Reproduction Pseudo-code

```
Solution child = new Solution();
For i = 1 to N do
For j = 1 to N do
If  i _= j then
Communication c = new Communication(i,j)
if !child.contains(c) then
TS1 = time slot containing c from parent1
TS2 = time slot containing c from parent2
```

```
ifTS1.Count > TS2.Count then
child.Add(TS1)
else
child.Add(TS2)
end if
end if
end if
end for
end for
```

V. EXPERIMENTAL RESULTS

In this section, we evaluate the proposed task migration scheme and compare it to two other migration schemes: Gathering-Routing-Scattering and Diagonal scheme [5]. All considered task migration strategies and the proposed technique are implemented on an NoC architecture simulated by Xmulator [1]. Xmulator [1] is a fully-parameterized discrete event simulator for interconnected networks which is augmented with Orion library to calculate the power consumption of the network. Here, the hardware and software requirements for VIP establishment and networks control are emulated in this environment. Also, the size and working frequency of the NoC process is set to 70nm and 250 MHz.

Figure 4 shows the results for tasks migration of a 5×5 sub-mesh (here SM[(2,2),(6,6)] to another 5×5 sub-mesh SM_[(9,9),(13,13)]) in a 16×16 mesh network. In this case, the source and destination sub-meshes do not overlap. The horizontal axis in the figures represents the traffic generation rate (message/node/cycle) by which an IP core generates and injects messages into the network. The vertical axis shows either the average message latency (in cycles), average migration latency (in cycles), or the total network power consumption (in nJ/cycle). figure 5 reports the same results but for overlapping source and destination sub-meshes (here the source sub-mesh s SM[(2,2),(6,6)] and the destination sub-mesh is SM_[(9,5), (13,9)]).

Both Gathering-Rooting-Scattering [5] and proposed scheme based on VIP connections are completed in three phases. But as can be seen in the figures, VIPs direct large communication volume of the tasks with a much shorter period and less network power. So, as shown in figure 4 and figure 5, the achieved improvement is not considerable for low traffic rates when comparing to Gathering-Rooting-Scattering method. The reason is that the normal messages transmitted over the conventional mesh NoC do not disturb task migration flows.

However, the improvement obtained by the proposed technique is noticeable when message generation rate increases. Also, comparing the total network power consumption confirms that the proposed technique has considerably reduced the network power, especially when traffic is heavy In summary, the proposed VIP-based scheme has reduced the average message latency by 13%, average migration latency by 14%, and total network power by 10%with respect to Gathering-Routing-Scattering method. Although in the second scenario , the source and destination sub-meshes overlap in Y dimension and the distance between corresponding nodes are less than the first scenario, the average message latency and total power consumption of the Gathering-Routing-Scattering and VIP-based schemes are larger, especially for heavy traffic loads and when migration paths are more congested.

VI.CONCLUSION

In this paper, we proposed a task migration scheme in mesh-based NoCs based on low-latency and low-power virtual point-to-point (VIP) connections. Based on the existing proposals, experimental results revealed that the proposed scheme could improve over two other schemes. This improvement was due to a series of strategies used in the proposed method: 1) through VIP establishment, migration VIPs has scheduled as prioritized over normal VIPs. 2) VIP transfer threshold, Tvip, was chosen much larger than packet switched threshold, Tps, which guarantees that large task migration messages can be transferred with no interrupt while no starvation occurs for normal messages. 3) Since a new VIP connection could tear down previously established VIPs (if suitable), could even prioritize the scenarios with multiple task migration. For instance, a task migration process with larger source and destination mesh size may have larger priorities. Moreover, the VIP paths were limited for connections among selected diagonal cores in the source and destination sub-meshes. This can better help reducing stalled normal message flows during VIPs.

VII. REFERENCES

[1] XmulatorNoC simulator, 2008.
[2] L. Benini, and G. De Micheli, “Networks on chip: a new paradigm for systems on chip design”, IEEE Computer, Vol. 35, 2001, pp.70-78.

[3]E.W. Briao, D. Barcelos, and F.R. Wagner, "Dynamic task allocation strategies in MPSoC for soft real-time applications," in Proc. DATE, 2008, pp. 1386-1389.
[4] DALLY, WILLIAM J., "Virtual Channel Flow Control." IEEE Transactions on Parallel and Distributed Systems, March 1992, pp. 194-205.
[5] DALLY, WILLIAM J., AND POULTON, JOHN W., Digital Systems Engineering, Cambridge University Press, 1998.