Meshed Bluetree: Time-Predictable Multi-Memory Interconnect for Multi-Core Architectures

Haitong Wang, Neil C. Audsley, Xiaobo Sharon Hu, Wanli Chang

***Abstract*—Multi-core architectures are widely adopted in the emerging real-time applications, such as autonomous vehicles and robotics, where latency is required to be both bounded in the worst case (i.e., time predictability) and low. With the number of processors growing, the conventional memory interconnects, i.e., shared bus, crossbar, and network-on-chip (NoC), suffer high latency due to the increasing logic size of their centralised arbiter, which is deployed for time predictability. In this paper, we introduce a novel distributed multi-memory interconnect, Meshed Bluetree, and explain its operation. Constructed by coupling a router network with multiple Bluetree-based memory architectures in parallel, Meshed Bluetree allows simultaneous access to multiple memory modules. We present the analysis for the predictable timing behaviour of memory access to bound the worst case. Evaluation on FPGA with synthetic memory workloads and real-world benchmarks demonstrates the effectiveness of our work, i.e., as the number of memory modules increases, the latency is reduced with the same scale. This work reports the first time-predictable distributed multi-memory interconnect, significantly contributing to multi-core real-time systems.**

***Index Terms*—multi-core architecture, memory interconnect, time predictability**

I. Introduction

In the emerging real-time application scenarios, such as highly automated driving and robotics, there is a stringent requirement on the latency being both bounded in the worst case (hence time predictability) and low. To deal with complex functionality and achieve high performance, multi-core architectures are widely deployed, where multiple processors share one memory module. With the trend of integrating more processors into the multi-core architectures, the contention over memory access aggravates and multiple memory modules are getting engaged.

The conventional multi-core architectures employ shared bus to connect processors and the shared memory modules, e.g., AHB (Advanced High-Performance Bus) [1] in the SoC (System-on-Chip) design. Communication between processors and access to the memory must be delivered through the shared bus. Once a single access occurs, the bus is blocked, which leads to severe contention. Alternatively, the crossbar design, e.g., AXI Interconnect (Advanced Extensible Interface) [2], alleviates the contention issue with a set of switch boxes. It uses dedicated links to replace the shared bus, which allows multiple accesses to occur simultaneously. The NoC (Network-on-Chip) architecture employs a packet switching network [3][4], where each processor is connected through a router and experiences less contention compared to the shared bus. The shared memory modules are commonly located on the edge of the network.

In order to achieve time predictability, the above conventional interconnects on multi-core architectures typically implement an arbitration scheme, such as priority-based, time-division multiplexing (TDM), or round-robin, on a centralised arbiter. As the number of processors grows, the logic size of the arbiter hardware increases, which limits the maximum synthesisable clock frequency. One promising approach recently investigated is to employ distributed memory interconnects, where the tree-based structure with pipelined stages (Figure 1 as an example) can break the critical path of multiplexing into multiple shorter steps with small logic size. Although this introduces additional clock cycles, the latency is reduced, as higher clock frequency can be synthesised, pipelining is supported, and scaling to a large number of processors gets enabled.

The distributed memory interconnects are classified as *locally arbitrated* and *globally arbitrated*. The *locally arbitrated* interconnect is constructed with a distributed binary arbitration tree that multiplexes the memory requests from processors to the shared memory module. Based on this architecture, the *globally arbitrated* interconnect integrates global scheduling to the distributed data paths, and thus can be considered as the *locally arbitrated* interconnect with traffic shaping. In general, the *locally arbitrated* interconnect allows the average-case latency to be much lower than the worst case, making the time predictability analysis challenging. By contrast, the *globally arbitrated* interconnect essentially limits the average-case behaviour to be similar to the worst case, facilitating the time predictability analysis. However, the processor is slowed down, degrading the overall system performance. In addition, the *globally arbitrated* interconnect requires complex scheduling as well as strict coordination, and potentially suffers the synchronisation issue.

Main contributions: In this paper, we introduce a novel distributed memory interconnect, Meshed Bluetree, and explain its operation. Constructed by coupling a router network with multiple *locally arbitrated* Bluetree-based memory architectures in parallel, Meshed Bluetree enables multiple processors to simultaneously access multiple memory modules. We present the analysis for the predictable timing behaviour of memory access to bound the worst case,

This article was presented in the International Conference on Hardware/Software Codesign and System Synthesis 2020 and appears as part of the ESWEEK-TCAD special issue. W. Chang is the corresponding author.

H. Wang, N. C. Audsley and W. Chang are with the Department of Computer Science, University of York, UK (email: hw963@york.ac.uk; neil.audsley@york.ac.uk; wanli.chang@york.ac.uk)

X. Hu is with the Department of Computer Science and Engineering, University of Notre Dame, USA (email: shu@nd.edu)

which can be extended to other architectures than Bluetree. Evaluation on FPGA with synthetic memory workloads and real-world benchmarks demonstrates the effectiveness of our work, i.e., as the number of memory modules increases, the latency is reduced with the same scale. This work reports the first time-predictable distributed multi-memory interconnect, significantly contributing to multi-core real-time systems.

The remainder of this paper is structured as follows. Section II reviews the related work on distributed multi-core memory interconnects. Section III describes the basic Bluetree-based architecture. The proposed Meshed Bluetree is presented in Section IV with its predictable timing analysis in Section V. The experimental results are reported in Section VI and Section VII makes the concluding remarks.

II. Related Work

In contrast to the conventional centralised design, distributed memory interconnects are emerging in multi-core architectures, especially for real-time applications. In this section, we discuss the related works on *locally arbitrated* and *globally arbitrated* distributed memory interconnects, and make a comparison. Whilst the *locally arbitrated* interconnects have the potential for general applicability and good performance, the resource contention is difficult to resolve and analyse, which is the main contribution of this work.

Among the *locally arbitrated* distributed memory interconnects, [5] develops an arbitration tree with globally synchronised timestamps. The arbiter at each distributed multiplexing stage applies the first-come-first-served (FCFS) scheme, where memory requests relay to the next stage according to the increasing order of their timestamp values. It is feasible for very limited platforms, mainly those having AXI bus [6] with small numbers of outstanding memory requests.

Alternatively, Bluetree [7][8][9] is initially developed for the NoC architecture as the external memory tree, to provide a second network exclusively for accessing the shared memory module. It separates memory traffic from the processor router network, hence preventing memory access from interfering with communication between processors. Bluetree is constructed by a set of pipelined multiplexers using a local round-robin arbitration scheme. The Bluetree-based memory architecture does not require full synchronisation and allows multiple memory requests to be transferred through the tree network simultaneously. This aids further scalability towards good average-case performance. However, the *locally arbitrated* Bluetree interconnect demands complicated analysis of the predictable timing behaviour.

Among the *globally arbitrated* distributed memory interconnects, TDM Tree [10] is built upon the integration of global TDM scheduling components with a tree-based multiplexing architecture. When a TDM time slot arrives, one memory request from a specific processor is allowed to relay to the tree network. With the global scheduling interval, there is no contention to the shared resources, neither the data paths nor the root memory module. No interference exists between memory accesses. However, TDM Tree requires strict synchronisation and complex coordination. In addition, it does not support work-conservation, which potentially leads to a considerable waste of bandwidth.

Based on the global scheduling interval, Globally Arbitrated Memory Tree (GAMT) [11][12] extends the distributed multiplexing tree with priority-based rate control schemes. This aims to better utilise the bandwidth with flexibility. When a time slot arrives, successive memory requests from a specific processor are allowed to relay to the tree network. The behaviour of memory access is affected by the priority-based rate control scheme, such as Frame-based Static Priority (FBSP) and Credit-Controlled Static Priority (CCSP). GAMT could only benefit specific applications, as it is generally hard to model the memory requests on hardware, unlike task scheduling in operating systems. In addition, the synchronisation suffers.

The contention over memory accesses aggravates with an increasing number of processors integrated. The *locally arbitrated* memory architectures allow multiple memory requests in transfer simultaneously, leading to contention over either the shared root memory module or the overlapped data paths in the tree-based interconnect. Once the root memory module is occupied, the entire request flow is blocked. By contrast, the *globally arbitrated* memory architectures provide contention-free request paths, avoiding memory access interference. However, this reservation-based method fails to alleviate memory workloads. Instead, it budgets memory bandwidth and slows down a processor, consequently degrading the overall system performance.

Such resource contention as discussed above has been widely studied on multi-core architectures. For example, [13] proposes message combining to reduce resource contention within the tree-based architectures. As for the memory interconnects with multiple pipelined stages, the requests simultaneously arriving at one arbiter stage can be merged, and the memory response is then split to multiple individual ones along the response path. This reduces contention on the overlapped data paths. However, it leaves the burden to the shared root memory module and requires an increasing logic size for each pipelined stage.

The alternative method is to invest additional hardware resources to increase bandwidth. For example, a virtual channel [14][15] can be employed to the shared router in NoC, which alleviates the router contention from multiple communication flows and provides flexibility in the channel utilisation. For the tree-based architectures, [16] proposes that multiple memory banks can be deployed at the root of the *locally arbitrated* Bluetree architecture, which increases bandwidth and potentially supports mixed-criticality systems with diverse memory features. However, it moves the design burden to the shared memory controller, and this centralised design at the Bluetree root inevitably limits the maximum clock frequency of the synthesised hardware.

Following this idea of having multiple memory banks in parallel, we propose the design of Meshed Bluetree with distributed data paths and local arbitration. The aim is to provide a multi-memory interconnect for multi-core architectures, towards predictable timing behaviour (i.e., with the memory

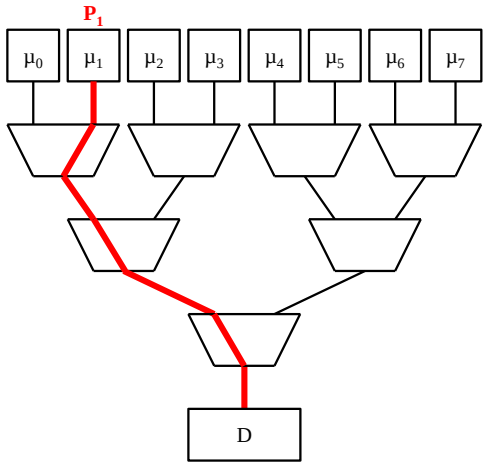

Fig. 1. A Bluetree-based architecture with 8 clients sharing one memory module.

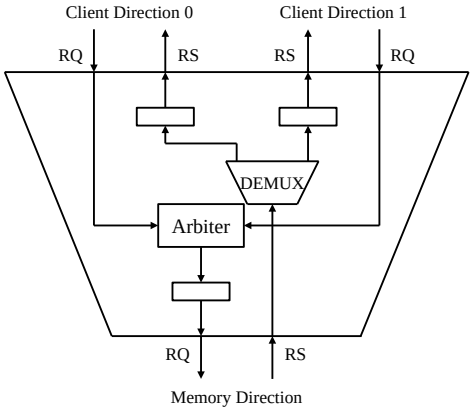

Fig. 2. The Bluetree multiplexer.

latency analysable and worst case bounded), latency reduction in the average case, as well as scalability.

III. Basic Architecture

Our proposed distributed multi-memory interconnect, Meshed Bluetree, is built upon the *locally arbitrated* Bluetree-based architecture, and can be extended to other architectural configurations, such as by reconfiguring with different local arbitration schemes or integrating with global scheduling schemes. In this section, we describe and analyse the conventional Bluetree-based architecture.

Figure 1 illustrates an 8-client Bluetree-based architecture, consisting of 8 clients, the Bluetree interconnect, and the shared memory module. A client can be a single processing core or a multi-core processor, and denoted by μ_i, where i is the client index. Each client has a memory access path P_i, with P_1 for the client μ_1 highlighted in the figure. The Bluetree interconnect B employs multiple stages of 2-to-1 Bluetree multiplexers to construct the tree network, connecting clients at the leaves to the shared memory module D at the root. Across this bi-directional Bluetree network, memory requests issued by the clients are multiplexed and relayed to the shared memory, and memory responses return to the corresponding clients. As the number of clients grows, the tree network scales with more Bluetree multiplexer stages, which increases the Bluetree depth N_β. In Figure 1, N_β is equal to 3.

Figure 2 shows the Bluetree multiplexer with requests coming from two client directions. Arbitration occurs in the request path (RQ) to decide which direction of request to be relayed to the memory direction, and the next Bluetree multiplexers. The blocking factor α of the internal arbiter is defined such that every α requests from *Direction 0* can be blocked by at most one request from *Direction 1*, where *Direction 0* can be considered as the local high-priority path, and *Direction 1* is the local low-priority path. Starvation can be prevented by allowing one request from the low-priority path to be relayed for every α requests from the high-priority path. If there is no request from *Direction 0*, the arbiter imposes no blocking on *Direction 1* with outstanding requests. The implementation of the local arbiter requires an internal blocking counter. When the blocking factor is set as $\alpha = 1$, Bluetree can be considered as the distributed binary tree stages with a local round-robin scheme, which provides relatively fair access to the shared memory module for all clients.

On the other hand, the response path (RS) is non-blocking. The internal demultiplexer simply decides the route direction of the memory response as shown in Figure 2. Besides, a buffer is implemented along each direction as a common pipeline design practice. The Bluetree multiplexer interface is designed to operate in the client-server manner, which allows each local Bluetree multiplexer to function independently, without requiring the operating state knowledge of any other Bluetree multiplexer nearby. The Bluetree interconnect does not require full clock synchronisation.

The Bluetree-based architecture is initially designed to provide good average-case performance and guarantee the worst-case memory latency. With the *locally arbitrated* data paths, memory accesses show predictable behaviour. However, the shared root memory is the architectural bottleneck. As shown in Figure 1, closer to the Bluetree root, more memory access paths overlap, where the memory requests from different clients have to share the common hardware paths, as well as the shared root memory. This shared interconnect architecture inevitably causes resource contention during simultaneous memory accesses.

IV. Meshed Bluetree

In this section, we introduce the Meshed Bluetree distributed memory interconnect. The topology of our design is based on the Mesh-of-Trees (MoT), similar to [17][18][19], where [17] and [18] focus on the topology research. In [19], the MoT is developed with single-clock-cycle data paths, using a set of switches coordinated by a global control signal to establish a complete memory access path dedicated for a specific client at a time. This MoT operates in the circuit-switched round-robin manner with centralised control, allowing data transfer between clients and memories within one clock cycle and enabling relatively simple timing analysis. However, with the

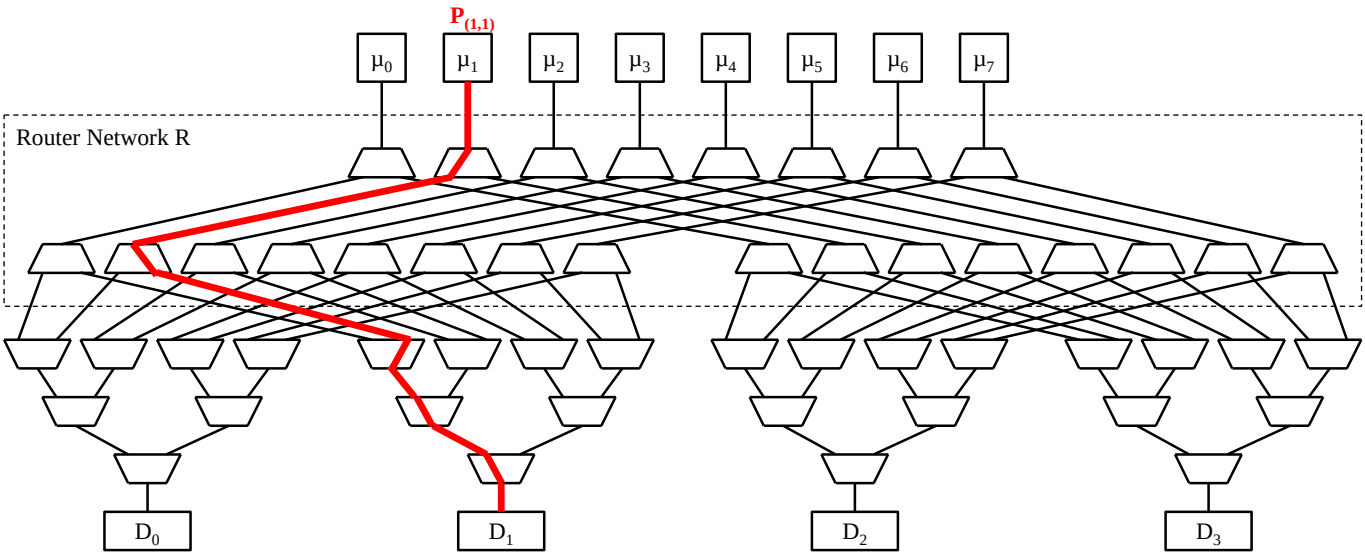

Fig. 3. An 8×4 Meshed Bluetree with 8 clients and 4 memory modules.

expanding system configuration (i.e., the number of clients and memories), the logic size of the centralised design increases logarithmically, which severely limits the maximum synthesisable clock frequency.

By contrast, our Meshed Bluetree employs distributed data paths with local arbitration. Although additional clock cycles are introduced, it allows a much higher clock frequency, enables pipelining, and scales to a large system. There have been works quantitatively comparing the centralised and the distributed design, such as in [12]. According to its experimental results, the centralised design fails to scale with the number of processors, and the results can be even worse with more complex arbitration schemes. By comparison, the maximum synthesisable clock frequency of the distributed design remains high and constant with the increasing number of processors, demonstrating scalability.

With distributed data paths, Meshed Bluetree is proposed to resolve the resource contention on the conventional Bluetree-based memory architecture as discussed in Section III and to enable multiple processors to share multiple memory modules. Our aim is to achieve good and scalable average-case performance, whilst providing predictable timing behaviour across the pipelined multiplexing stages, i.e., with an analysable memory access latency bound.

Figure 3 illustrates the architecture of Meshed Bluetree, which is constructed by coupling a distributed router network (the upper half) with multiple Bluetree-based architectures in parallel (the lower half). In this particular example, eight clients share four memory modules. Each client μ_i has a memory access path $P_{(i,j)}$ to connect to the memory module D_j, where j is the memory module index. The path $P_{(1,1)}$ for the client μ_1 to connect to the memory module D_1 is highlighted in the paper. The memory modules can be paralleled memory banks within one DRAM module as analysed in [16]. The design can also be extended with paralleled scratchpad memory, cache, or mixed types of memory components. Meshed Bluetree allows sufficient design flexibility to support multi-core applications.

When a client μ_i issues a memory request, the router network R first decides the routing path and relays the request to a specific Bluetree-based architecture. Then the corresponding Bluetree interconnect B_j further multiplexes and relays this request to the destination memory module D_j. Here, the same subscript j indicates a one-to-one relationship between a Bluetree interconnect and a memory module. The memory response returns across the bi-directional meshed interconnect in a reverse process. As the memory address range can be partitioned across these paralleled memory modules, the simultaneous accesses to different memory modules can be processed concurrently. This significantly reduces the contention over a single memory module and increases the system bandwidth.

The router network R is constructed with multiple stages of Bluetree routers. With the number of memory modules N_D growing, the router network R scales with more pipelined router stages, which increases the router depth N_R in the tree-based architecture. In Figure 3, N_R is equal to 2. The design of the Bluetree router is shown in Figure 4. The local request path (named RQ as before) of the Bluetree router is non-blocking, and the internal demultiplexer simply decides the route direction of memory requests. Pipelined buffers and client-server interfaces are also implemented, similar to the Bluetree multiplexers.

Arbitration occurs in the local response path (named RS as before) to decide which direction of memory response to be relayed to the client, and the next Bluetree routers. An applicable local arbitration scheme can be round-robin, which provides locally fair access for both Bluetree directions. It is also feasible to employ static priority-based arbitration at the local router stage, always allowing the memory response from one direction to have higher priority and get relayed first. The consecutive responses along one path have time intervals

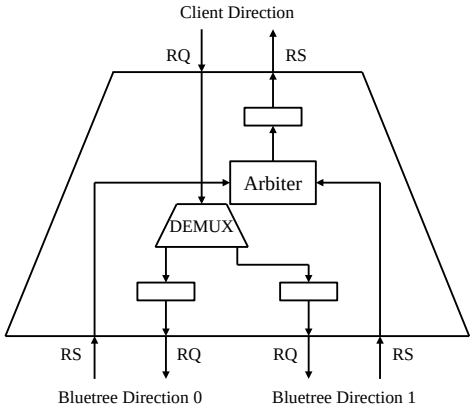

Fig. 4. The Bluetree router.

in between, related to the responding speed of the memory modules. Therefore, a memory response will not be blocked at one router stage for long, even with a lower priority. The exact amount of blocking along the response path depends on the number of responses ahead in transfer. This Meshed Bluetree architecture allows memory modules with different response time, potentially supporting mixed-criticality applications.

The term *system cardinality* can be introduced to describe the configuration of the Meshed Bluetree architecture. It is expressed as the product of the number of clients N_μ and the number of memory modules N_D. For example, the system cardinality of the Meshed Bluetree in Figure 3 is 8×4. With an increasing system cardinality, the Meshed Bluetree scales with either higher router depth N_R or higher Bluetree depth N_β, indicating larger hardware consumption.

Below we provide an analysis on the number of components required to construct the Meshed Bluetree interconnect, including Bluetree multiplexers, Bluetree routers, and Bluetree wires. For a single Bluetree memory architecture as in Figure 1, the number of Bluetree multiplexers N_{mux} increases with the number of clients N_μ, considering the tree topology. For the Meshed Bluetree, the total number of Bluetree multiplexers N_{mux} also increases with the number of memory modules N_D. To sum up,

$$N_{mux} = (N_\mu - 1) \times N_D. \quad (1)$$

Taking Figure 3 as an example, the number of Bluetree multiplexers N_{mux} is equal to $(8-1) \times 4 = 28$. Similarly, the number of Bluetree routers N_{router} increases with the number of memory modules N_D in the tree-based router network and the number of clients N_μ. To sum up,

$$N_{router} = (N_D - 1) \times N_\mu. \quad (2)$$

Taking Figure 3 as an example, the number of Bluetree routers N_{router} is equal to $(4-1) \times 8 = 24$. Bluetree wire refers to the data bus for the communication between any two Bluetree components within the interconnect, i.e., clients, memory modules, multiplexers, and routers. The number of Bluetree wires N_{wire} is,

$$N_{wire} = (N_\mu - 1) \times N_D + (N_D \times 2 - 1) \times N_\mu, \quad (3)$$

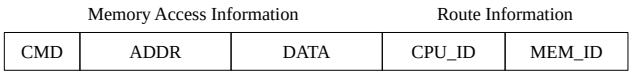

Fig. 5. The packet format for the communication across the Meshed Bluetree interconnect.

where for each Bluetree multiplexer, there is one Bluetree wire (pointing towards the memory modules), thus $(N_\mu - 1) \times N_D$; for each Bluetree router, there is one Bluetree wire (pointing towards the clients), thus $(N_D - 1) \times N_\mu$; and the rest $N_D \times N_\mu$ connects the multiplexers with routers. Taking Figure 3 as an example, the number of Bluetree wires N_{wire} is equal to $(8-1) \times 4 + (4 \times 2 - 1) \times 8 = 84$.

The width of the data bus within the Meshed Bluetree interconnect depends on the communication packet format, which generally includes the memory access information and the route information as shown in Figure 5. The memory access information is generated or received by the client or the root memory, including the 1-bit command field CMD (i.e., the memory command type such as memory read or memory write), the 32-bit address field ADDR, and the 32-bit data field DATA. In the memory request packet, CMD '0' indicates a read request, and CMD '1' indicates a write request. In the memory response packet, CMD '0' indicates a read response, and CMD '1' indicates a write acknowledgement.

The route information is required for the packet transfer across the interconnect, and it is used for for each distributed multiplexing stage to track or decide the route. The route information includes the 8-bit client identifier field CPU_ID and the 8-bit memory identifier field MEM_ID, which supports a maximum Bluetree depth $N_\beta = 8$ and a maximum router depth $N_R = 8$. When a client issues a request, the corresponding CPU_ID is encoded by the local arbiter at each Bluetree multiplexer to track the route: left shift by 1 bit with '0' for the local high-priority path, or left shift 1 bit with '1' for the local low-priority path. CPU_ID is also used by the demultiplexer along the response path to decide the route back to the corresponding client, decoded by the right shift operation at each local stage. Similarly, MEM_ID is required by Bluetree routers.

In the above design, the total bit-width of a packet is 81, which is also the width of the data bus as well as the multiplexers and routers. It is to be noted that this design is reconfigurable and allows flexible extension. For example, a priority field can be employed in the route information for the priority-based arbitration scheme. An extra interface is needed for the conversion of the packet format (e.g. converting the packet format between the Meshed Bluetree interconnect and the AXI bus). In addition, our design is independent of the memory addressing scheme.

In general, a single memory access over the Meshed Bluetree architecture incurs higher delay, considering the longer pipelined data path with the router network. However, simultaneous memory accesses can be processed by the parallel memory modules concurrently, which increases the bandwidth and effectively alleviates the contention over one shared memory module. Latencies for intensive memory accesses

can be reduced and hence the overall system performance is improved. In addition, the Meshed Bluetree architecture supports memory isolation, potentially simplifying software or OS (operating system) development for multi-core systems, and provides sufficient flexibility for mixed-criticality systems with diverse memory bandwidth and latency requirements. The challenge is how to analyse the timing behaviour of the memory access to bound the worst-case latency, which is particularly important for the real-time applications and will be provided in the next section.

V. Predictable Timing Analysis

Real-time systems must guarantee the response within the specified timing constraints. Multi-core architectures are typically designed for good average-case performance, where software components contend for the shared hardware resources. Memory accesses over the distributed tree-based interconnect cause contention to both the overlapped data paths and the shared root memory modules. In this section, we present the predictability analysis of the Meshed Bluetree architecture. Our method defines the general analytical flow to the multi-core architectures with *locally arbitrated* interconnects, and can be extended to other architectural configurations than the Meshed Bluetree.

A. Timing Behaviour

Our analysis aims to compute and bound the memory access latencies across the Meshed Bluetree architecture. In general, the latency t of the memory access ω consists of three parts, the request path latency t_{RQ}, the root memory latency t_D, and the response path latency t_{RS},

$$t(\omega) = t_{RQ}(\omega) + t_D + t_{RS}(\omega). \tag{4}$$

When there is no contention, i.e., in the best case, it takes 1 clock cycle to cross each pipelined stage, along both the request and response paths. Therefore, the best-case request path latency $t^{BC}_{RQ}(\omega)$ and the best-case response path latency $t^{BC}_{RS}(\omega)$ are both equal to $N_R + N_\beta$. The root memory latency t_D is taken as constant. The best-case overall latency t^{BC} of the memory access ω is then,

$$\begin{aligned} t^{BC}(\omega) &= t^{BC}_{RQ}(\omega) + t_D + t^{BC}_{RS}(\omega) \\ &= 2 \times (N_R + N_\beta) + t_D. \end{aligned} \tag{5}$$

The best-case latency $t^{BC}(\omega)$ gives the minimum latency that a memory access experiences across the Meshed Bluetree architecture. It is based on the assumption of no contention, i.e., every pipelined stage is always in the idle status, ready to accept the request and the response without any delay. When there is resource contention to either the data path or the shared root memory, the request or the response may be blocked, which leads to increasing path latency $t_{RQ}(\omega)$ or $t_{RS}(\omega)$, and consequently the total latency $t(\omega)$.

Blocking whitin the Meshed Bluetree architecture can be classified as *inter-path blocking* and *intra-path blocking*. The *inter-path blocking* occurs when a request or response crosses an arbiter stage and gets blocked by the other local path. Therefore, the *inter-path blocking* is affected by the local arbitration scheme. On the other hand, the *intra-path blocking* occurs when a request or response is blocked by any other request or response ahead of it, from either the same client or the other clients. In addition, the interaction between the *inter-path blocking* and *intra-path blocking* needs to be considered. For example, when a request ω_1 experiences *inter-path blocking* from ω_2, ω_2 might overtake ω_1 and get ahead in the same data path, which potentially leads to additional *intra-path blocking*.

Based on the above blocking analysis, the memory access across the Meshed Bluetree architecture exhibits predictable behaviour. If the exact memory access profiles are known, the detailed status of the memory flow and the local arbiter at every pipelined stage can be derived. Hence, the accurate timing can be computed. It is to be noted that such exact analysis becomes more complicated as the router depth N_R or the Bluetree depth N_β increases, due to the following reasons. First, a larger number of pipelined buffers in the data path potentially leads to more *intra-path blocking*. Second, the *inter-path blocking* could increase with the the number of arbiters. Third, there is interference between the pipelined stages. As the nature of tree-based architectures, if there is any blocking in the stage close to the root, the entire tree will be affected. For example, if the Bluetree root stage is blocked, the request flow within this Bluetree-based architecture stalls. Similarly, with more *inter-path blocking* close to the Bluetree leaf stage, there will be more consequent *intra-path blocking* in the overlapped paths.

In practice, there is often uncertainty with the memory access profiles, such as on the number of memory requests and the memory issuing time instants. In this case, the exact timing analysis is not valid. Below, we will provide the worst-case latency analysis on the memory access across the Meshed Bluetree architecture.

B. The Worst-Case Memory Access Latency

Similar to the analysis in Section V-A, the calculation on the worst-case latency t^{WC} of the memory access ω also consists of the worst-case request path latency $t^{WC}_{RQ}(\omega)$, the worst-case response path latency $t^{WC}_{RS}(\omega)$, and the constant root memory latency t_D,

$$t^{WC}(\omega) = t^{WC}_{RQ}(\omega) + t_D + t^{WC}_{RS}(\omega). \tag{6}$$

Along both the request and response paths occur the *inter-path blocking* and *intra-path blocking*.

1) The Worst-Case Request Path Latency: Each blocking that the request ω experiences in the request path induces an amount of path latency proportional to the root memory latency t_D within the corresponding Bluetree-based architecture. Essentially, the request flow stalls until the memory is idle again to accept the next request. This latency caused by waiting for the root memory masks the path latency across the pipelined stages. Therefore, the maximum blocking number denoted as $N^{WC}_{RQ}(\omega)$, which the request ω experiences across

the corresponding request path $P_{(i,j)}$, can be used to calculate the worst-case request path latency $t_{RQ}^{WC}(\omega)$,

$$t_{RQ}^{WC}(\omega) = N_{RQ}^{WC}(\omega) \times t_D. \tag{7}$$

For the request path within the router network R, the memory request ω can only be stalled due to the *intra-path blocking*. With the router depth N_R, the maximum blocking number in the router request path is equal to N_R, under the assumption that all the buffers are occupied at every pipelined stage. These blockings within the router network aggravate the *inter-path blocking* in the overlapped Bluetree request paths, which gets more severe closer to the root memory modules.

For the request path within the corresponding Bluetree interconnect B_j, the blocking analysis complicates, involving both the *inter-path blocking* and the *intra-path blocking*. The term *priority path* is introduced here to analyse the maximum blocking number. It is used to track the local priority at each Bluetree stage β_k across the request path, where k is the stage index. Referring to the interconnect in Figure 3, the priority path $P_{(1,1)}$ for the client μ_1 to the memory module D_1 can be $P_{(1,1)} = \{L, H, H\}$, for example, where L is for the local low-priority and H for the local high-priority. Therefore, the path $P_{(1,1)}$ within the Bluetree interconnect B_1 is across the local low-priority path at the Bluetree stage β_2, the local high-priority path at β_1, and the local high-priority path at the Bluetree root stage β_0, eventually to the memory module D_1. The related local priority can be expressed as $P_{(1,1)}(\beta_2) = L$, $P_{(1,1)}(\beta_1) = H$, and $P_{(1,1)}(\beta_0) = H$.

By tracking the local priority, the calculation of the maximum blocking number $N_{RQ}^{WC}(\omega)$ across the corresponding Bluetree request path is iterative, based on the calculation of the maximum blocking number at each Bluetree stage β_k. Intuitively, the blocking number at any given Bluetree stage β_k is dependent on (i) the amount of blocking that has occurred at previous stages along the request path, and (ii) the amount of blocking that can occur at the current stage, which is dependent on the local blocking factor α. Following this idea, $N_{RQ}^{WC}(\beta_k)$ is defined as the iterative blocking up to and including the Bluetree stage β_k, and the maximum arbiter blocking number $N_{\alpha}^{WC}(\beta_k)$ is to represent the blocking at the Bluetree stage β_k only. The iterative calculation can be expressed as,

$$N_{RQ}^{WC}(\beta_k) = N_{RQ}^{WC}(\beta_{k+1}) + N_{\alpha}^{WC}(\beta_k) + 1, \tag{8}$$

where $+1$ indicates that the local buffer is occupied. At the Bluetree leave stage, $N_{RQ}^{WC}(\beta_{k+1}) = N_R$, which is the amount of blocking that has accumulated in the router network, according to our previous analysis.

The maximum arbiter blocking number $N_{\alpha}^{WC}(\beta_k)$ is locally decided by the blocking factor α at the corresponding Bluetree stage β_k. With the local arbitration scheme discussed earlier in Section III, every α requests from the local high-priority path can be blocked by at most one request from the local low-priority path, and every single request from the local low-priority path can be blocked by up to α requests from the local high-priority path. Given $N_{RQ}^{WC}(\beta_{k+1})$, $N_{\alpha}^{WC}(\beta_k)$ can be calculated with the local priority $P_{(i,j)}(\beta_k)$,

$$N_{\alpha}^{WC}(\beta_k) = \begin{cases} \lceil \frac{(N_{RQ}^{WC}(\beta_{k+1})+1)}{\alpha} \rceil & \text{H} \\ (N_{RQ}^{WC}(\beta_{k+1})+1) \times \alpha & \text{L} \end{cases}, \tag{9}$$

where $+1$ is to include the request ω and determine the total amount of requests to cross the local arbiter at this Bluetree stage.

Taking the local high-priority path as an example, if there are $N_{RQ}^{WC}(\beta_{k+1}) + 1$ (the number of requests accumulated till the upper stage plus the request under study itself) requests going through the high-priority path, the maximum blocking from the low-priority path is this number divided by α and then applied a ceiling function. Other types of arbitration may be applied as well and our interconnect makes no specific requirement.

To summarise the above analysis, the maximum blocking number up to and including any given Bluetree stage β_k can be computed with (8) and (9). The maximum blocking number that the request ω experiences across the request path $N_{RQ}^{WC}(\omega)$ can be calculated iteratively, starting with the value N_R from the router network R to the Bluetree root stage β_0 within the interconnect B_j. Finally, the maximum blocking number in the request path $N_{RQ}^{WC}(\omega)$ is

$$N_{RQ}^{WC}(\omega) = N_{RQ}^{WC}(\beta_0), \tag{10}$$

and the worst-case request path latency $t_{RQ}^{WC}(\omega)$ can be calculated with (7). The worst-case assumption is that the request path gets flooded by interfering requests — (i) all pipelined buffers across the data path are occupied, and (ii) the local arbiter always harms the request flow.

With the increasing Bluetree blocking factor α, the maximum blocking number in the request path $N_{RQ}^{WC}(\omega)$ decreases with more local high-priority tracks. According to the Bluetree arbitration design in Section III, when the blocking factor $\alpha = 1$, Bluetree can be considered as distributed tree stages with the local round-robin scheme and provides fair accesses for all requests regardless of the client index. This design is implemented in our experiments.

2) The Worst-Case Response Path Latency: The analysis for the blocking in the response path is different from that for the request path discussed above. According to our design of the Meshed Bluetree architecture in Section IV, the consecutive memory responses are separated by certain time intervals, depending on the responding speed of the memory modules. Therefore, a response path will not be flooded by interfering responses. The maximum blocking that the memory access ω experiences in the response path is much less than that in the request path. In general, the response path is non-blocking within a Bluetree interconnect B_j, and the memory response can experience blocking in the router network R. The blocking analysis within the router network varies, depending on whether the root memory modules have homogeneous latency.

If all the paralleled memory modules have the identical root memory latency t_D, there will be no blocking within the router network R. The memory requests from the same

client are always issued successively. Therefore, there is only one response arriving at each arbitration stage at a time, hence no *inter-path blocking*. If the root memory latency t_D varies on different memory modules in the paralleled Bluetree-based architectures, the *inter-path blocking* occurs in the router network R. One response may stall in each pipelined stage for at most 1 clock cycle due to one contending response from the other local path. Referring to our analysis in Section IV, the Bluetree router could locally employ either the round-robin arbitration scheme or the static priority-based arbitration scheme. Below we analyse the maximum blocking number with both schemes. The worst case occurs when the local arbiter along the response path always harms the response flow.

With the round-robin scheme at each router stage, one response can be blocked by at most one other response from the other local path. Considering the response intervals from the memory modules and the basic pipelined data path latencies (crossing routers and multiplexers without blocking), such *inter-path blocking* will not lead to any *intra-path blocking* of the responses behind. Therefore, the maximum blocking number in the response path is determined by the router depth N_R as $N_{RS}^{WC}(\omega) = N_R$. The worst-case response path latency $t_{RS}^{WC}(\omega)$ can be calculated as the sum of the basic pipelined path latencies (through the router network and the Bluetree interconnect) plus blocking,

$$\begin{aligned} t_{RS}^{WC}(\omega) &= N_\beta + N_R + N_{RS}^{WC}(\omega) \\ &= N_\beta + N_R + N_R \\ &= N_\beta + 2 \times N_R. \end{aligned} \tag{11}$$

The local static priority-based arbitration could lead to more *inter-path blocking*. With the static priority at each router stage, the internal arbiter will always allow memory responses from one local path with higher priority to block the other local path. Following the architectural characteristics, the responses in one path are separated with intervals, and one response experiences the basic pipelined data path latencies. Therefore, one response will not be stalled at a local router stage for long. The *inter-path blocking* does not cause any *intra-path blocking* to the memory responses behind in the same path, as the clients process responses immediately, unlike the memory modules that take t_D to process requests.

When a response ω crosses the leaf stage of the router network, there will be only one interfering response from the other local path considering the memory responding intervals. Then the response ω experiences more *inter-path blocking* at the subsequent router stages closer to the client. The maximum blocking number in the response path can be bounded as $N_{RS}^{WC}(\omega) = N_D$, with the assumption that the response flow is always interfered. Based on the above analysis, the worst-case response path latency $t_{RS}^{WC}(\omega)$ can be calculated as,

$$t_{RS}^{WC}(\omega) = N_\beta + N_R + N_D. \tag{12}$$

3) The Worst-Case Memory Access latency: Below we summarise the worst-case memory access latency analysis and calculation for our proposed Meshed Bluetree configurations. The round-robin arbitration is deployed for the Bluetree multiplexers and the static priority-based arbitration for the Bluetree routers in the implementation. The worst-case latency t^{WC} of the memory access ω can be computed from the worst-case latency across the request path $t_{RQ}^{WC}(\omega)$ and the response path $t_{RS}^{WC}(\omega)$. The worst-case request path latency can be calculated with (8), (9), and (10), where the local blocking factor α is set to 1 as in the implementation. The worst-case response path latency with local static priority can be calculated using (12). The overall equation for the worst-case memory access latency is,

$$\begin{aligned} t^{WC}(\omega) &= t_{RQ}^{WC}(\omega) + t_D + t_{RS}^{WC}(\omega) \\ &= N_{RQ}^{WC}(\beta_0) \times t_D + t_D + t_{RS}^{WC}(\omega) \\ &= (N_{RQ}^{WC}(\beta_0) + 1) \times t_D + N_B + N_R + N_D. \end{aligned} \tag{13}$$

We illustrate the above calculation with an example, on the 8×4 Meshed Bluetree architecture shown in Figure 3. The router network R has two stages and hence its depth is $N_R = 2$, which is equal to the maximum blocking number along the router request path, based on our analysis. A Bluetree architecture B_j has three stages and hence its depth is $N_\beta = 3$. The maximum blocking number along the request path within the Bluetree interconnect can be computed iteratively with (8), as discussed before. At the Bluetree leaf stage β_2, $N_{RQ}^{WC}(\beta_2) = N_{RQ}^{WC}(\beta_3) + N_\alpha^{WC}(\beta_2) + 1$, where $N_{RQ}^{WC}(\beta_3) = N_R = 2$. In this example, we assume α to be 1, which is effectively a local round-robin arbitration scheme, where every request from one path can be blocked by at most one request from the other path. The maximum arbiter blocking number can then be calculated following (9) as $N_\alpha^{WC}(\beta_k) = (N_{RQ}^{WC}(\beta_{k+1}) + 1) \times 1$. Therefore, at the Bluetree stage β_2, $N_\alpha^{WC}(\beta_2) = (N_{RQ}^{WC}(\beta_3) + 1) \times 1 = (2+1) \times 1 = 3$. The maximum blocking number at this stage is then $N_{RQ}^{WC}(\beta_2) = 2 + 3 + 1 = 6$.

Similar calculation can be performed for the Bluetree stage β_1 and the Bluetree root stage β_0. Finally, the maximum blocking number along the request path is $N_{RQ}^{WC}(\omega) = N_{RQ}^{WC}(\beta_0) = 30$. The worst-case memory access latency can be calculated using (13) as $t^{WC}(\omega) = (N_{RQ}^{WC}(\beta_0)+1) \times t_D + N_B + N_R + N_D = (30+1) \times 20 + 3 + 2 + 4 = 629$, where the root memory latency is assumed as constant $t_D = 20$ and the local static priority-based arbitration is employed in the router along the response path.

Our method presented in this section defines the general analytical flow to bound the worst case of the *locally arbitrated* platform. It can be extended to other architectural configurations than the Meshed Bluetree, which may require modification to the analysis of the local arbitration scheme. It is to be noted that the worst-case analysis may produce pessimistic bounds as the results, which potentially leads to conservative system design and resource dimensioning, as the memory access latency is the main part forming the overall program execution time. If the exact memory access profiles can be provided, the accurate memory access latency with no pessimism can be determined as discussed at the beginning of this section, based on the detailed status of the memory flow and the local arbiter at every pipelined stage. Without such exact memory access profiles, which is often the case in reality, the worst-case analysis reported in this section must

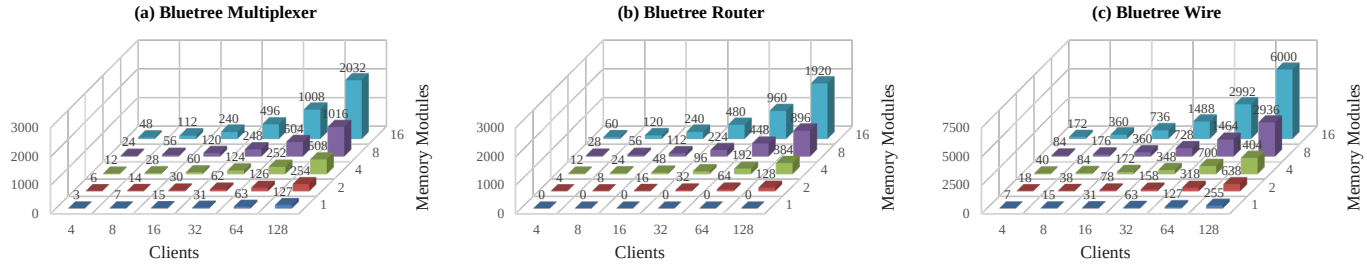

Fig. 6. Hardware consumption.

be deployed for real-time applications. The bound provided can also be tightened in future work, e.g., by restricting the demand from processors. Evaluation on the tightness is also important, and requires sufficiently representative memory workload patterns to be fair.

Compared to the conventional Bluetree-based architecture, a single memory access across the Meshed Bluetree experiences higher delay with the longer pipelined data path. However, simultaneous memory accesses can be processed by the paralleled memory modules concurrently, which reduces the contention and increases bandwidth, hence improving the overall system performance. To summarise, our Meshed Bluetree provides good average-case performance and guarantees the worst-case memory latency, the latter being particularly important for the real-time applications.

VI. Experimental Results

In this section, we evaluate our proposed Meshed Bluetree architecture on FPGA with both synthetic memory workloads and real-world benchmarks under various system configurations. Our experiment is conducted on Virtex-7 FPGA VC709 [20] with 100MHz of clock frequency. Two kinds of single-port memory modules are employed. An FPGA BRAM module [21] is reconfigurable from 4KB to 256MB, and gives immediate response to a memory request. A VC709 DDR3 DRAM module [22] is of 4GB in size and responds with approximately 30 clock cycles (when the FPGA system is with 100MHz of clock frequency and the DDR3 DRAM module is with 400MHz). Below, we first report the hardware consumption of the interconnect.

A. Hardware Consumption

The numbers of components required to construct the Meshed Bluetree interconnect, including multiplexers, routers, and wires, are reported in Figure 6, with the system cardinality increasing from 4×1 to 128×16. According to the analysis in Section IV, the results are calculated using (1), (2) and (3), which covers the entire interconnect. As shown in the graph, the number of components is proportional to the number of clients and memory modules, respectively.

The hardware consumption of the Bluetree multiplexer and the Bluetree router at the register-transfer level (RTL) is reported in Table I, in terms of look-up tables (LUTs), registers, and BRAMs, which are the basic logic units on FPGA. Gate-level consumption, which depends on the fabrication technology, may be evaluated in the future, where more detailed information such as the width and length of wires, as well as the exact amount of area, is available. Our current design employs the round-robin arbitration within the Bluetree multiplexers (i.e., the local blocking factor $\alpha = 1$) and the static priority-based arbitration within the Bluetree routers. The entire Meshed Bluetree architecture is implemented with Bluespec System Verilog [23][24] and synthesised with Xilinx Vivado [25][26].

TABLE I
Hardware consumption at the RTL level.

Component	LUT	Register	BRAM
Bluetree Multiplexer	105	269	0
Bluetree Router	88	251	0

As shown in Table I, one single Bluetree router consumes slightly fewer resources than a Bluetree multiplexer, and their difference is mainly on the internal arbiter design. The BRAM consumption is 0 with the selected arbitration schemes. It is to be noted that this resource consumption is obtained from the Vivado synthesis report, and will be much lower after optimisation. Based on Figure 6 and Table I, the hardware consumption of the Meshed Bluetree interconnect increases linearly over the system cardinality.

B. Synthetic Memory Workloads

This section evaluates memory access latencies across the 8-client Meshed Bluetree architecture with various configurations. We deploy traffic generators as clients, which simulate memory requests without processing any data. The memory workload parameters include the path outstanding request number and the request interval. The traffic generator successively issues memory requests with randomised varying request intervals in between, until the path outstanding request number is reached, and then stalls. After a memory response returns, this traffic generator starts to issue memory requests again. Such synthetic memory workloads provide traffic patterns close to practical applications and facilitate behaviour observation.

In the experiments, we limit the outstanding request number for each client to be 2. The varying memory request interval is randomly produced from $[1, 64]$. This refers to the practical applications with memory requests distributed

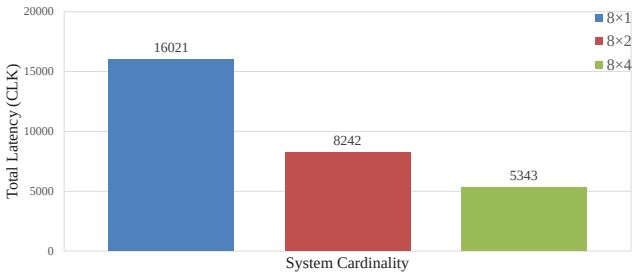

Fig. 7. Total latency with multiple homogeneous memory modules.

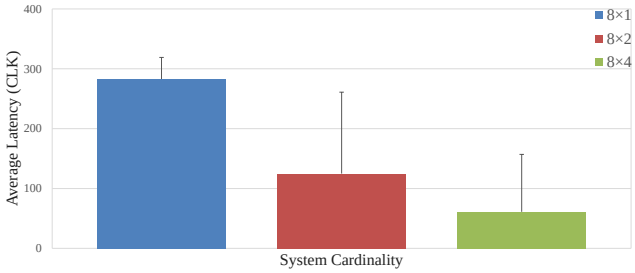

Fig. 8. Average latency with multiple homogeneous memory modules.

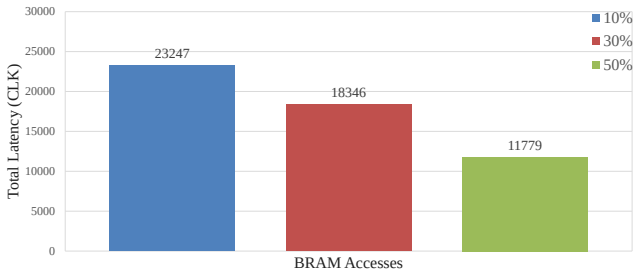

Fig. 9. Total latency with mixed memory modules.

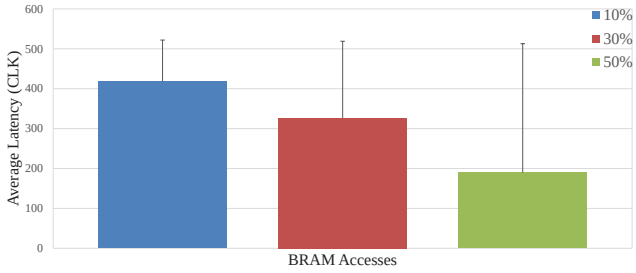

Fig. 10. Average latency with mixed memory modules.

over time. Two sets of experiments on multiple homogeneous memory modules and mixed memory modules, respectively, are performed and analysed below, each with a total of 100 memory requests. As write buffers are commonly employed to expedite the memory writes, we focus on single-mode memory reads with randomly generated addresses, which take considerable latencies.

1) Multiple Homogeneous Memory Modules: This experiment evaluates latencies across the Meshed Bluetree architecture with multiple homogeneous memory modules, under the system cardinalities 8×1, 8×2, and 8×4. It is to be noted that the 8×1 Meshed Bluetree architecture is the same as the conventional 8-client Bluetree-based architecture. The memory module is implemented based on FPGA BRAM with an additional delay of 20 clock cycles (as there are only 2 DDR DRAM modules on VC709). The accesses are partitioned among these paralleled memory modules following the uniform distribution.

Figure 7 shows that the total latency, reflecting the overall system performance, is roughly reduced by half as the number of memory modules doubles. The reduction is not exactly by half (slightly less than), as according to our previous analysis, memory accesses experience longer data path delays across the meshed interconnect. Although some path delays can be masked by the waiting for the root memory module, the latency of a single memory access increases. In addition, following a randomised process, the memory accesses are not evenly partitioned to the paralleled architecture, neither the target memory modules nor the issuing time instants. The contention over the heavier shared memory module increases the latency.

Figure 8 examines every single memory access, showing the average latency and the highest observed latency, the latter being the cap line. Although the blocking due to the shared resources still occurs, the simultaneous memory requests are partitioned into multiple memory modules in parallel through the Meshed Bluetree interconnect, which effectively alleviates the contention to a single memory module, and thus reduces the average memory access latency as well as the highest observed latency.

2) Mixed Memory Modules: This experiment evaluates latencies across the Meshed Bluetree architecture with mixed memory modules, under the system cardinality of 8×2, using an FPGA BRAM module and a VC709 DDR3 DRAM module. In the experiment, the percentage of memory accesses to the BRAM module varies from 10%, 30%, to 50%, as the faster memory module tends to be of smaller size and memory address range.

Figure 9 shows that the total latency is reduced with the increasing BRAM access percentage. When this percentage changes from 10% to 30%, the total latency is reduced by about 21% due to the much faster response from BRAM. When this percentage further increases from 30% to 50%, the total latency drops even faster by 36%. Therefore, if the architecture scales with faster memory modules in parallel, the system could have more noticeable performance improvement.

Figure 10 examines every single memory access, showing that the average latency gets reasonably reduced with more accesses to the faster memory module. However, the highest observed latency remains unchanged. As the memory accesses are randomly partitioned between BRAM and DRAM, the traffic generator can quickly issue the next memory requests to the DRAM module after receiving the very fast response from the BRAM module. In this case, the contention to the shared DRAM module is not alleviated, which thus does not improve the highest observed latency.

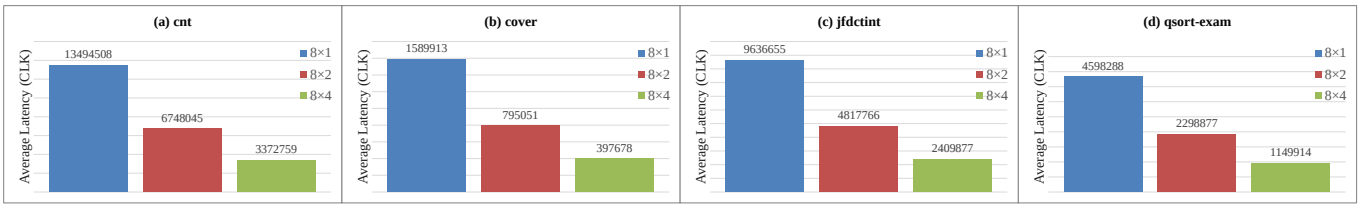

Fig. 11. Performance with multiple homogeneous memory modules.

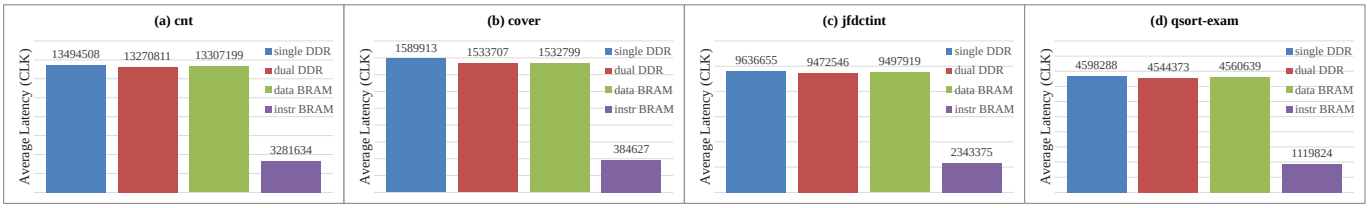

Fig. 12. Performance with separate instruction and data memory modules.

C. Benchmarks

This section evaluates the average-case performance of the Meshed Bluetree using Mälardalen benchmarks [27]. The experiments are based on the 8-Microblaze [28] system running 8 calculation-intensive benchmarks of different functionality. Each core executes one benchmark. It is to be noted that there is no local memory deployed, which makes the root memory modules under heavy pressure. The memory accesses are evenly partitioned to the shared memory modules. The results are averaged over 1000 repeated runs. Due to the space limit, four benchmarks, *cnt*, *cover*, *jfdctint*, and *qsort-exam* are shown in two sets of experiments.

1) Multiple Homogeneous Memory Modules: This experiment evaluates the performance of the Meshed Bluetree architecture with multiple homogeneous memory modules, under the system cardinalities 8×1, 8×2, and 8×4, on the benchmarks. The memory module is implemented based on FPGA BRAM with an additional delay of 20 clock cycles (as there are only 2 DDR DRAM modules on VC709). Figure 11 shows that the average latency is reduced roughly by half as the number of memory modules in parallel doubles, for all the benchmarks.

2) Separate Instruction and Data Memory Modules: This experiment evaluates the performance of the Meshed Bluetree architecture with separate instruction and data memory modules, using FPGA BRAM module and VC709 DDR3 DRAM. The system is configured as 8×1 with a single DRAM, 8×2 with dual DRAM, and 8×2 with DRAM and BRAM. In the last case, the memory accesses are partitioned as instruction DRAM and data BRAM, or instruction BRAM and data DRAM.

Figure 12 shows the experimental results. Compared with a single DRAM configuration (denoted as single DDR in the graph), the average latency with the separate instruction DRAM and data DRAM configuration (denoted as dual DDR in the graph) only slightly drops. Similar observations are made with the instruction DRAM and data BRAM configuration (denoted as data BRAM in the graph), where the average latency in Figure 12 (c) *jfdctint* even increases compared with the dual DRAM configuration. The reason is that, as the benchmarks are instruction-intensive, faster data BRAM accesses lead to more frequent instruction requests to the slower DRAM and aggravate the congestion, which increases the average latency. When the system is configured with instruction BRAM and data DRAM (denoted as instr BRAM in the graph), the average latency drops by approximately 75%. In all these experiments of this section, no local memory, such as cache, gets deployed, which is a factor to reduce the memory access latency.

VII. Concluding Remarks

This paper introduces the first time-predicable distributed memory interconnect — Meshed Bluetree — that supports multi-core architectures with multiple parallel memory modules. We first present the design of the Meshed Bluetree architecture, which is constructed by coupling a distributed router network with multiple conventional Bluetree-based architectures in parallel. This allows simultaneous memory accesses to be processed by the parallel memory modules concurrently, which increases the bandwidth and alleviates the contention over one shared memory module. We also report the predictable timing analysis with the static calculations to bound the worst case across the Meshed Bluetree architecture. The evaluation with synthetic memory workloads and real-world benchmarks demonstrates the effectiveness of our design. That is, as the number of memory modules increases, the latency is reduced with the same scale.

One promising direction for future work is to investigate hardware/software co-design strategies that map (or divide) tasks to the memory modules in the Meshed Bluetree or similar architectures, aiming to improve the performance, including the execution time, power consumption, and reliability. In addition, more benchmarks, e.g., with more intensive demands or of mixed types, can be taken to evaluate the proposed interconnect, and finer analysis can also be performed on more specific memory workload patterns.

REFERENCES

[1] ARM, *AMBA 5 AHB Protocol Specification*, 2015.

[2] Xilinx, *AXI Interconnect*, 2017.

[3] W. J. Dally and B. Towles, "Route packets, not wires: On-chip inteconnection networks," in *Proceedings of the 38th Annual Design Automation Conference*, ser. DAC '01. New York, NY, USA: ACM, 2001, pp. 684–689.

[4] L. Benini and G. De Micheli, "Networks on chips: A new soc paradigm," *Computer -IEEE Computer Society-*, vol. 35, no. 1, pp. 70–78, Jan 2002.

[5] J. H. Rutgers, M. J. G. Bekooij, and G. J. M. Smit, "Evaluation of a connectionless noc for a real-time distributed shared memory many-core system," in *2012 15th Euromicro Conference on Digital System Design*, Sep. 2012, pp. 727–730.

[6] ARM, *AMBA AXI and ACE Protocol Specification*, 2011.

[7] G. Plumbridge, J. Whitham, and N. Audsley, "Blueshell: A platform for rapid prototyping of multiprocessor nocs and accelerators," *SIGARCH Comput. Archit. News*, vol. 41, no. 5, pp. 107–117, Jun. 2014.

[8] M. Schoeberl, S. Abbaspour, B. Akesson, N. Audsley, R. Capasso, J. Garside, K. Goossens, S. Goossens, S. Hansen, R. Heckmann, S. Hepp, B. Huber, A. Jordan, E. Kasapaki, J. Knoop, Y. Li, D. Prokesch, W. Puffitsch, P. Puschner, and A. Tocchi, "T-crest: Time-predictable multi-core architecture for embedded systems," *Journal of Systems Architecture*, vol. 61, 04 2015.

[9] H. Wang, N. C. Audsley, and W. Chang, "Addressing resource contention and timing predictability for multi-core architectures with shared memory interconnects," in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2020, pp. 70–81.

[10] M. Schoeberl, D. V. Chong, W. Puffitsch, and J. Sparsø, "A Time-Predictable Memory Network-on-Chip," in *14th International Workshop on Worst-Case Execution Time Analysis*, ser. OpenAccess Series in Informatics (OASIcs), vol. 39. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2014, pp. 53–62.

[11] M. Dev Gomony, J. Garside, B. Akesson, N. Audsley, and K. Goossens, "A generic, scalable and globally arbitrated memory tree for shared dram access in real-time systems," in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2015, pp. 193–198.

[12] M. D. Gomony, J. Garside, B. Akesson, N. Audsley, and K. Goossens, "A globally arbitrated memory tree for mixed-time-criticality systems," *IEEE Transactions on Computers*, vol. 66, no. 2, pp. 212–225, Feb 2017.

[13] G. F. Pfister and V. A. Norton, "Hot spot contention and combining in multistage interconnection networks," *IEEE Transactions on Computers*, vol. C-34, no. 10, pp. 943–948, Oct 1985.

[14] N. Kavaldjiev, G. J. M. Smit, and P. G. Jansen, "A virtual channel router for on-chip networks," in *IEEE International SOC Conference, 2004. Proceedings.*, Sep. 2004, pp. 289–293.

[15] A. Mello, L. Tedesco, N. Calazans, and F. Moraes, "Virtual channels in networks on chip: Implementation and evaluation on hermes noc," in *2005 18th Symposium on Integrated Circuits and Systems Design*, Sep. 2005, pp. 178–183.

[16] N. Audsley, "Memory architecturesfor noc-based real-time mixed criticality systems," *Proc. WMC, RTSS*, pp. 37–42, 2013.

[17] A. DeHon and R. Rubin, "Design of fpga interconnect for multilevel metallization," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 10, pp. 1038–1050, 2004.

[18] A. O. Balkan, G. Qu, and U. Vishkin, "A mesh-of-trees interconnection network for single-chip parallel processing," in *IEEE 17th International Conference on Application-specific Systems, Architectures and Processors (ASAP'06)*, Sept 2006, pp. 73–80.

[19] A. Rahimi, I. Loi, M. R. Kakoee, and L. Benini, "A fully-synthesizable single-cycle interconnection network for shared-l1 processor clusters," in *2011 Design, Automation Test in Europe*, March 2011, pp. 1–6.

[20] VC709, https://www.xilinx.com/products/boards-and-kits/dk-v7-vc709-g.html, [accessed 31 October 2019].

[21] *Xilinx 7 Series FPGAs Memory Resources*, 2017.

[22] *Xilinx 7 Series FPGAs Memory Interface Solutions*, 2017.

[23] Bluespec, https://bluespec.com/, [accessed 31 October 2019].

[24] *Bluespec System Verilog Reference Guide*, 2014.

[25] Xilinx, https://www.xilinx.com, [accessed 31 October 2019].

[26] Vivado, https://www.xilinx.com/products/design-tools/vivado.html, [accessed 31 October 2019].

[27] J. Gustafsson, A. Betts, A. Ermedahl, and B. Lisper, "The Mälardalen WCET benchmarks – past, present and future," in *WCET2010*, B. Lisper, Ed. Brussels, Belgium: OCG, Jul. 2010, pp. 137–147.

[28] Xilinx, *MicroBlaze Processor Reference Guide*, 2019.