Dynamically Reconfigurable On-Chip Communication Architectures for Multi Use-Case Chip Multiprocessor Applications

Sudeep Pasricha†, Nikil Dutt, Fadi J. Kurdahi,

University of California, Irvine, CA
{spasrich, dutt, kurdahi}@uci.edu

Abstract – **The phenomenon of digital convergence and increasing application complexity today is motivating the design of chip multiprocessor (CMP) applications with multiple use cases. Most traditional on-chip communication architecture design techniques perform synthesis and optimization only for a single use-case, which may lead to sub-optimal design decisions for multi-use case applications. In this paper we present a framework to generate a dynamically reconfigurable crossbar-based on-chip communication architecture that can support multiple use-case bandwidth and latency constraints. Our framework generates on-chip communication architectures with a low cost, low power dissipation, and with minimal reconfiguration overhead. Results of applying our framework on several networking CMP applications show that our approach is able to generate a crossbar solution with significantly lower cost (2.4× to 3.8×), and lower power dissipation (1.5× to 3.1×), compared to the best previously proposed approach.**

I. Introduction

In the near future, emerging applications particularly in the multimedia and networking domains will be implemented as chip multiprocessors (CMPs) that consist of several on-chip components such as programmable processors, memories, peripherals and network interfaces [1]. These CMP implementations will have to support several application operating modes (called use-cases) because of the trend towards digital convergence [2]. For instance, portable handheld global positioning systems (GPS) today operate not only as navigational devices, but can also play MP3 songs, support 3D games, act as personal organizers, support wireless internet browsing and display streaming video. Each of these functionalities has a distinct performance requirement. 3D gaming, for instance, requires high performance to render graphics and maintain a high frame display rate, otherwise user experience is compromised. In contrast, playing MP3/WAV songs requires simpler audio codec decoding and has a much lower performance requirement. Some functionalities, such as H.264 video playback, may even have multiple performance requirements (i.e., multiple use cases), depending on the chosen decoding parameters and display modes. With the number of use-cases increasing rapidly in applications today, designers are finding it extremely challenging to create CMP implementations that can support multiple heterogeneous use-cases.

On-chip communication architectures play a critical role in supporting diverse communication requirements across application use-cases [3]-[5]. Whether bandwidth and/or latency constraints in CMP applications can be satisfied depends to a significant extent on the design of the underlying communication architecture fabric that facilitates the entire on-chip inter-component data communication. Several research efforts have proposed techniques for designing hierarchical shared bus, crossbar (or matrix), and networks-on-chip (NoC) communication architectures for single use-case applications. However, very few works have looked at designing communication architectures for applications with multiple use-cases. In particular, no existing work has looked at designing crossbar based on-chip communication architectures customized for multi use-case applications. Since crossbar based communication architectures are beginning to be widely used in CMP designs [6], there is a need to explore techniques to design low cost and low power dissipation crossbar communication architectures that meet multi use-case application performance requirements. This paper addresses such a need.

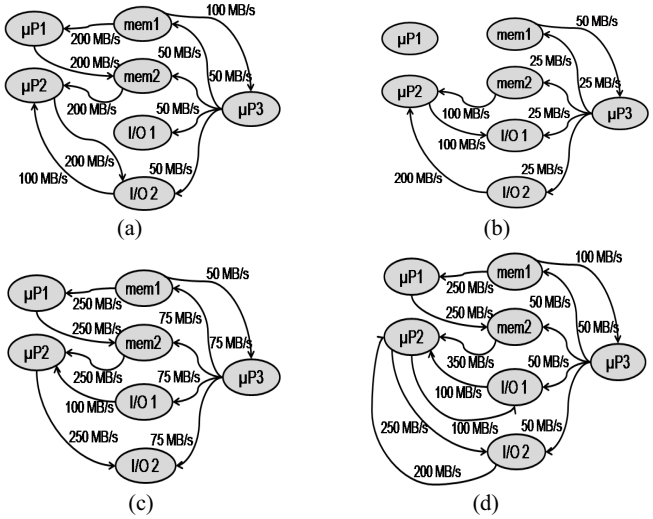


Fig. 1. Use cases for networking application (a) use-case1, (b) use-case2, (c) use-case3, (d) compound use-case (use-case2+use-case3)

Fig. 1 shows an example of a networking router subsystem with multiple use-cases. The application is implemented as a CMP with three processors, on-chip memories and network interfaces (I/O). Fig. 1 (a)-(c) show three use cases of the application that have vastly different bandwidth requirements between components. It is also possible for the application to execute multiple use cases simultaneously, as shown in Fig. 1 (d), where use cases 2 and 3 execute at the same time. Typically, switching between (individual or compound) use cases takes place during application execution when users interact with the application or if there is a change in the environment (e.g., change in wireless signal strength, or battery level) [7]. Such a switch in use cases results in the temporal switch of application task and communication graphs. It is very likely that a communication architecture customized for a single use case may not meet the performance requirements for another use case. Thus there is a need to enhance traditional on-chip communication architecture synthesis techniques to handle multi use-case applications.

In this paper we propose a framework for designing crossbar-

† author currently with Colorado State University, Fort Collins, CO

978-1-4244-2749-9/09/$25.00 ©2009 IEEE

based on-chip communication architectures to meet performance requirements for multi-use case applications. Our framework synthesizes a low cost and low power dissipation crossbar architecture that supports dynamic (runtime) reconfiguration to match dynamically changing use-case requirements. Every time a use case switch occurs during application execution, attributes of the crossbar are modified dynamically so that the performance requirements of the new use case can be supported. Results from experiments on several networking CMP designs show how our approach generates crossbar architectures with up to 3.8× lower cost and up to 3.1× lower power dissipation compared to the best previously proposed technique. To the best of our knowledge, this is the first piece of work that focuses on synthesizing and optimizing crossbar on-chip communication architectures for multi use-case applications.

II. Related Work

There has been a lot of work done in the area of on-chip communication architecture synthesis. Research efforts have looked at synthesizing hierarchical shared bus architectures [8]-[10], crossbar or bus matrix architectures [11]-[12] and networks-on-chip (NoC) [13]-[15]. The synthesis techniques used in these works optimize for different design goals such as power, performance, cost and area. However, these techniques optimize the on-chip communication architecture assuming only a single operating mode for the application (i.e., a single use case). Since today's digital convergence devices must perform multiple functions, applications being designed today are beginning to incorporate multiple use cases. It is extremely important to consider these multiple use-cases during on-chip communication architectures synthesis to avoid sub-optimal designs [23].

A few approaches have proposed using dynamic reconfiguration of on-chip communication architectures, to cope with changing traffic patterns during application execution. Sekar et al. [16] proposed the FLEXBUS bus architecture that dynamically changes the topology from a single shared bus to a dual shared bus to cope with changing performance requirements of the application. Lahiri et al. [17] and Richardson et al. [18] proposed shared bus architectures with dynamically varying arbitration priorities and TDMA (Time Division Multiple Access) slot allocations to cope with changing traffic demands. A few works such as Pandey et al. [19] propose dynamic voltage scaling (DVS) for shared buses at runtime to reduce power dissipation in the communication architecture. Our multi use-case crossbar synthesis framework makes use of a combination of dynamic arbitration scheme reconfiguration and dynamic voltage and frequency scaling (DVFS) to adapt to the changing requirements across multiple use-cases.

Recent work [20]-[23] has begun to focus on designing and optimizing on-chip communication architectures to meet performance constraints of multiple use-cases. Murali et al. [20] proposed creating a single synthetic worst case from multiple use cases and designing a mesh NoC architecture for it. The authors also incorporated dynamic voltage and frequency scaling (DVFS) to reduce NoC power dissipation. This approach was enhanced by the same authors in [21] with a greedy mapping heuristic for better dataflow path selection. Hansson et al. [22]-[23] described an optimization technique to reduce switching time between use cases on a mesh NoC fabric. Our proposed approach in this paper is different from these existing works in the following ways: *(i)* we focus on synthesizing and optimizing a crossbar communication architecture instead of a mesh NoC for multi use-case applications, *(ii)* we incorporate an early physical layout tool to obtain more accurate performance and power estimates during synthesis, and *(iii)* our proposed synthesis technique creates a crossbar architecture with low dynamic reconfiguration overhead, instead of the high overhead of updating several distributed slot/routing tables with new paths/schedules from memory for a mesh NoC in [20]-[23] every time a use case switch occurs. Results from experiments on several CMP designs (Section VI) show how our synthesis framework has low reconfiguration overhead and also outperforms these approaches.

(a) full crossbar (b) partial crossbar

Fig. 2. CMP with crossbar on-chip communication architecture

III. Crossbar based CMP Architectures

CMP architectures consist of multiple processing cores, each with one or more ports that can behave as either a master or a slave. Masters initiate data transfer transactions by issuing requests and slaves receive the requests and process the transactions (either manipulating and storing data, or returning data or an acknowledgement to the initiating master). Crossbar communication architectures (e.g., AXI PL300 [24] or STBus [25]) are often used to connect masters and slaves in a CMP architecture, and support data transfers. Fig. 2 (a) shows an example of a CMP architecture with a full crossbar communication architecture. The crossbar consists of a source router block (SRouter) at every master interface that buffers data received from the master, decodes the destination address and then sends the data to the appropriate destination router via a bus. At the destination router (DRouter), the data is buffered and arbitration is performed to select one out of the possibly several waiting data items (from different source routers) to send to the appropriate slave. Since a full crossbar as shown in Fig. 2 (a) may be too costly for large CMP systems because of an excessive number of buses, and buffers/routers, designers typically make use of partial crossbar architectures as shown in Fig. 2 (b). Partial crossbars have fewer buses and routers/buffers, but still satisfy application constraints, resulting in lower cost and lower power dissipation [11]-[12].

IV. Problem Description

For a given multi use-case application, we assume that hardware/software partitioning has already taken place and the computation tasks have been mapped to appropriate processing cores in the CMP architecture. Let $\theta = \bigcup_{x=1}^{n} UC_x$ be a superset of all n application use-cases $(UC_1, UC_2, \ldots, UC_n)$. For each use case $UC_n \in \theta$ we are given a communication constraint graph $UCG_n(V, E)$ where $v_i \in V$ represents a processing element (master or slave core) and the directed edge $e_{ij} = \{v_i, v_j\} \in E$ denotes the data communication from core v_i to v_j. For every e_{ij} in graph $UCG_n(V, E)$, $\varphi_n(e_{ij})$ denotes the bandwidth constraint in bits per second, and $\gamma_n(e_{ij})$ denotes the latency constraint for data transfer from core v_i to v_j. For every $v_i \in V$, the height and width of the core is also known, and denoted by $\mathcal{H}_i$ and $\mathcal{W}_i$, respectively.

Let $\mathcal{B}_m$ denote the set of buses connecting masters (or source routers) to slaves (or destination routers), $\mathcal{B}_s$ represent the set of buses connecting slaves to the destination routers, and $\mathcal{A}$ represents the set of arbiters at the destination routers. Let $\mathcal{F}$ denote the clock frequency of all the buses in the matrix, and $\mathcal{S}$ denote the arbitration scheme used in an arbiter. The objective of the multi-use case crossbar design problem is to determine a topology $\mathcal{T}(V, B_m, B_s, \mathcal{A})$ with appropriate core to bus mapping, protocol parameters $\mathcal{P}(\mathcal{F}, \mathcal{S})$, and a chip layout, such that: *(i)* for every use case $UC_n \in \Theta$ that has a corresponding communication constraint graph $UCG_n(V, E)$, $\forall e_{ij} = \{v_i, v_j\} \in E$ there exists a path in $\mathcal{T}$ that satisfies $\varphi_n(e_{ij})$ and $\gamma_n(e_{ij})$,*(ii)* crossbar cost (number of buses) is minimized, and *(iii)* overall system-level power dissipation for inter-core communication is minimized.

V. MUCSYN: Multi Use-Case Crossbar Synthesis

A. Synthesis Strategies

In this subsection we discuss different approaches for designing crossbar communication architectures that can support applications with multiple use-cases.

***Traditional Approach*:** Out of all the approaches, the simplest is the traditional approach [11]-[12] which constructs the crossbar for the most communication intensive use-case. The partial crossbar is optimized to support the selected use-case performance constraints. However, such an approach does not guarantee that performance constraints of other use-cases will be satisfied. As an example, a crossbar architecture designed for the use-case in Fig. 1 (a) with the traditional approach was found to be unable to support performance requirements of the use-cases in Fig. 1 (c)-(d).

***WC Approach*:** Another approach is to create a synthetic worst-case use case and optimize a partial crossbar for it, as done in [20]. The synthetic worst-case use case consists of the most stringent bandwidth and latency constraints from all the application use cases. Consequently, a crossbar architecture optimized for such a use case satisfies the constraints of all the use cases of an application. However such an approach is typically very conservative and overdesigns the system, especially for applications in which traffic characteristics of use cases are very different, or if the number of use cases is large [21]. For instance, for a 35 core, 13 use-case CMP design that we experimented with (Section VI), the crossbar architecture designed with this approach was around 3× larger compared with our proposed approach.

***WC Approach with DVFS*:** The previous approach can be enhanced by adding dynamic voltage and frequency scaling (DVFS) [20]-[21]. In this approach, the voltage and frequency for buses are scaled down when a use case with less stringent performance constraints is executing. Such an approach may reduce power dissipation compared to the *WC Approach*, but does not reduce crossbar cost. It may even increase area overhead due to additional circuitry required for DVFS.

***Proposed Approach (MUCSYN)*:** In contrast to the previous approaches, our proposed approach does not limit itself by creating a synthetic WC, and instead considers crossbar topology generation, arbitration policy selection and layout generation in an integrated manner to create a low cost and low power dissipation crossbar architecture. Unlike previous approaches [20]-[21], the integrated layout allows an accurate estimation of power and performance numbers during synthesis. Additionally, the synthesized crossbar supports low overhead dynamic arbitration policy reconfiguration and DVFS to reduce crossbar cost and power dissipation while satisfying use case constraints. The framework for our proposed approach is described in the next subsection.

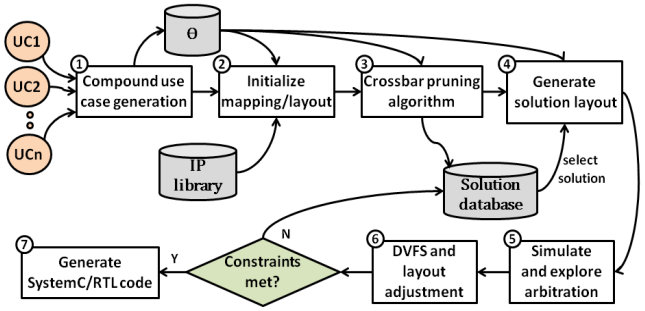


Fig. 3. Multi Use Case Synthesis Framework

B. Framework Overview

Fig. 3 shows a high level overview of our multi use-case crossbar synthesis framework. In the first step we populate the application use-case superset θ with individual use-cases as well as any compound use-cases (combinations of use-cases that can run simultaneously) from the application. Next we perform the initial mapping of IP cores onto a full crossbar template and create an initial layout of the design. Subsequently, we make use of our crossbar pruning algorithm to create a set of partial crossbar architectures that can theoretically satisfy constraints for all the use-cases of the application. The best solution is selected and a layout is generated for the partial crossbar architecture. Information obtained from the layout (e.g., wire length, number of repeaters and wire pipelining depth) is used in a cycle accurate SystemC [26] system-level simulation to generate accurate power and performance results for the solution. In this step, dynamic arbitration reconfiguration is also explored, to ensure that all the use case constraints are satisfied. Next, DVFS is explored for one or more use-cases with this solution, and adjustments with the additional DVFS circuitry made to the layout. If the solution violates one or more constraints, it is discarded. The next best solution from the solution database is selected and the process repeated till a solution is found that satisfies all bandwidth and latency constraints for the use cases. Finally, RTL and cycle accurate SystemC code for the optimized crossbar is generated.

We now describe the synthesis framework in more detail. In Step 1, compound use-cases are automatically generated from individual use-cases, based on input from the designer that specifies which use-cases can execute in parallel. Each edge of the use case graph $UCG_n(V, E)$ can have a bandwidth and/or latency constraint. In the compound use case, the bandwidth constraint of the combined edge is the summation of the individual use case bandwidths, while the latency constraint is the minimum latency of the combined edges. Such an automatic compound use case generation is very useful and saves considerable designer effort, especially for applications with a large number of use cases, several of which may execute in parallel. The individual and newly generated parallel use cases are used to populate the use-case superset θ.

In the next step (Step 2), the IP cores of the CMP are mapped and connected to a full crossbar template of a standard communication architecture (e.g., AXI or STBus). In this

initialization step, use case constraints are mapped to the buses in the crossbar. A bandwidth use case vector $\Omega_{BW} = (\varphi_1(e_{ij}),\varphi_2(e_{ij}),\ldots,\varphi_n(e_{ij}))$ is created for each bus B in the crossbar that connects any two cores v_i and v_j. This vector aggregates the bandwidth constraints for all *n* use cases of the application pertaining to the data flow between the cores connected by the bus. Similarly, a latency use case vector $\Omega_{LAT} = (\gamma_1(e_{ij}),\gamma_2(e_{ij}),\ldots,\gamma_n(e_{ij}))$ is also created for each bus. Next an initial layout of the chip is generated using a high-level simulated annealing floorplanner based on sequence pair representation (PARQUET [27]). More details on the layout generation are presented in subsection V.C.

Algorithm 1 Crossbar Pruning

1. Generate search tree *ST* of all possible agglomerative clusterings
2. Perform breadth first search ∀ node *v* ∈ *ST*
3. For each previously unvisited node *v*
  a. mark *v* as visited
  b. calculate cost *c(v)*
  c. generate TDMA arbitration slot schedules for solution
  d. check if *v* satisfies bandwidth constraints
    i. select a previously unselected bus B
    ii. *violation* = *width(B)*×*maxfreq(B)* < $\boldsymbol{\varphi_n(e_{ij})}$, ∀$\boldsymbol{\varphi_n(e_{ij})}$ ∈ $\boldsymbol{\Omega_{BW}}$
    iii. repeat **i-ii** till *violation* == *true* or no more buses left
4. If node *v* violates constraint, remove *v* and all its children from *ST*
5. If node *v* satisfies constraints, *S* = *S* ∪ *v*
6. Continue traversal till no more unvisited nodes are left

In Step 3, we use a crossbar pruning algorithm to reduce the number of buses and logic components in the crossbar, while ensuring that bandwidth and latency constraints for all use-cases are satisfied. The algorithm is an extension of our previous work [12] that generated a low cost, non-reconfigurable crossbar for a single use case. The pseudo code for the enhanced algorithm is presented in Algorithm 1. The algorithm starts by creating a search tree with each level of the tree representing a successive agglomerative clustering of two IP cores (or groups of IP cores). The root node represents the full crossbar solution, while other nodes represent solutions with cores clustered together. The clustering of cores in a node may result in buses being clustered together, making the node a potential partial crossbar solution. Whenever two buses are clustered together, their corresponding bandwidth vectors are summed and the minimum latency values from the latency vectors are selected to create vectors for the new bus. Once the tree is created, a breadth first search is performed on the tree. For each traversed node the solution cost (number of buses) is calculated, and TDMA slot schedules are derived on a per use case basis using weighted bandwidth allocation [18]. The solution represented by the node is then checked to ensure that it satisfies all bandwidth constraints. The check consists of comparing the maximum bandwidth supported by each bus (which is the product of the bus width and the maximum clock frequency that the bus can support) against bandwidth constraints $\varphi_n(e_{ij})$ for all the *n* use cases in the bandwidth use case vector Ω_{BW}. There is a violation if the bandwidth constraint for any use case exceeds the bandwidth that can be supported by the bus. If such a violation occurs, the node is bounded, i.e., the node and all its children are discarded from the search tree. This is done because the node does not represent a feasible solution, and since all the children of this node must contain the clustering that violated the constraints, they also do not represent feasible solutions. If on the other hand the node does not violate any bandwidth constraints, then the node is added to the solution database *S*. This process is repeated till no more unvisited nodes remain in the search tree.

In Step 4, the best solution from the database *S* is selected and a layout is created for this partial crossbar solution (subsection V.C). In Step 5, we customize the crossbar model and IP core connections in our SystemC [26] system-level simulation environment and simulate the application. Information obtained from the layout (e.g., wire length, number of repeaters and wire pipelining depth) and power models proposed in our earlier work [28] are used to generate statistics for crossbar power estimation and to check if application performance constraints are satisfied. If there are any latency violations, we explore using dynamic arbitration reconfiguration across use-cases. This is done because for some use-cases with stringent latency constraints, another scheme such as the static priority (SP) scheme might be more effective than a TDMA scheme. We therefore explore additional arbitration schemes in the destination routers, such as static priority (SP), first-in first-out (FIFO), and two level TDMA/RR (round robin) schemes [5] for use-cases with constraint violations. If any of these new schemes satisfy previously violated constraints, we use that scheme instead of TDMA. In Step 6, dynamic voltage and frequency scaling (DVFS) is performed on buses to reduce power dissipation. Our DVFS technique is applied on buses with available slack (usually with low bandwidth and high latency constraints) on a per use-case basis. We make use of the model from [29] where it is assumed that the square of the voltage scales linearly with frequency. We also perform layout adjustment to account for the additional area of the level up/down shifter circuitry [30] for DVFS, as well as the area overhead of multiple arbitration schemes wherever dynamic reconfigurable arbitration is used. If the solution is found to violate one or more constraints, then it is discarded. The next best solution is then selected from the database and Steps 4-6 repeated till a solution is found that satisfies all constraints. Once such a solution is found, our framework generates the cycle accurate SystemC and RTL code for the partial crossbar communication architecture.

C. Layout Generation

As mentioned in the previous subsection, our multi use case synthesis framework makes use of a high level floorplanner [27] to generate early layouts of our crossbar based CMP architecture, to obtain accurate performance and power statistics. We view bandwidth and latency constraints imposed by the application as mutually independent. For instance, many packet forwarding or non-critical multimedia traffic streams have high bandwidth requirements, but no stringent latency constraints, except those imposed by the application period [5]. In contrast, control-oriented signaling events (e.g., interrupts) and cache misses have a tight latency constraint, but no bandwidth requirements. The floorplan of an application must ensure that cores that have high bandwidth communication between them are placed close to each other, to reduce power dissipation. Additionally, cores that have tight latency constraints must also be placed closer together, to satisfy performance constraints. This implies a trade-off since it is not always possible to put cores closer together to obtain minimal power and also satisfy latency constraints. To appropriately bias our simulated annealing (SA) floorplanner towards achieving these goals, we use a hybrid bandwidth/latency metric as described in [31]. In this metric, latency constraints are considered more critical than bandwidth constraints. This is

because while bandwidth constraints can be met even if cores are somewhat further apart, tight latency constraints are highly dependent on core distance. If e_{ij} is a communication edge in $UCG_n(V,E)$ with the largest bandwidth requirement, and e_{mn} is a communication edge with the most stringent (lowest) latency constraint, then an integer k is determined, such that

$$\frac{\varphi(e_{ij})}{\gamma(e_{ij})^k} \leq \frac{\varphi(e_{mn})}{\gamma(e_{mn})^k}$$

Once k is determined, an edge weight is assigned to each edge, such that

$$\forall e_{ij} \in E,\ w(e_{ij}) = \frac{\varphi(e_{ij})}{\gamma(e_{ij})^k}$$

This weight is used to create 'bias nets' between cores that bias the SA algorithm to place the cores based on weight values – the higher the weight of an edge between two cores, the more likely they are to be placed adjacent to each other in the 2D floorplan. If two edges have the same weight, the one with the tighter latency is given a higher priority.

D. Dynamic Reconfiguration Infrastructure

We assume that the dynamic arbitration and DVFS reconfiguration is handled by a reconfiguration core (*DRCore*), and initiated by events such as a user command to start or stop an application, an embedded resource manager [32], or mode switches in the incoming stream [7]. *DRCore* can be any programmable processor, such as an ARM CPU, which may also be responsible for initiating and handling other control events in the system. It is connected to all the routers and is a key enabler for any needed reconfiguration during use-case switching. To ensure consistency, *DRCore* performs busy waiting, till all reconfiguration requests have been sent and all the acknowledgements received. Once acknowledgements from all the connected routers have been received, *DRCore* sends configuration words to reprogram arbitration schemes (e.g., TDMA slot schedules or a static priority list) and enable/disable DVFS if needed. Signals required for indicating a use-case switch to the routers, and for sending acknowledgement back to *DRCore* are part of the control signals of the interface protocol chosen (e.g., AXI [24] has user configurable signals that can be used for this purpose). Since the dataflow paths do not need to be reconfigured, the use case switching overhead is very low (~microseconds) and there is no need for a 'smooth switching' between use cases as in [20]-[23], where the applicability of DVFS and dynamic arbitration reconfiguration for critical use cases is limited. To reduce reconfiguration overhead for systems with large number of cores, multiple *DRCores* can also be used in a distributed manner. In this work however, we only consider a single *DRCore* for simplicity.

VI. Experiments

To determine the efficacy of our proposed multi use-case crossbar synthesis framework, we performed experiments on several CMP applications. We selected five proprietary applications from the networking domain, each with multiple use cases. The applications are responsible for various frame and data packet processing operations including protocol conversion, encryption/ decryption, forwarding, and data packing and unpacking. These applications are parallelized and implemented on a CMP architecture. Table I shows the characteristics of the CMP implementation of these applications, such as the total number of cores (memories, peripherals, processors), the number of programmable processors (ARM CPUs) and the number of use cases. The applications in Table I were modeled in SystemC [26] [33] at the transaction level based bus cycle accurate abstraction [34] to quickly and accurately estimate performance and power consumption of the applications. The various cores were interconnected using the AMBA AXI [24] protocol based crossbar communication architecture. The layout die size for these CMP applications was assumed to be 2cm×2cm.

TABLE I
CMP applications and their characteristics

CMP Application	# of cores	# of proc.	# of use cases
DHub1	18	5	8
PEncD	28	7	6
Fproc	35	10	13
PckFtr	40	14	21
DHub2	47	19	18

We synthesized crossbar communication architectures for each of these applications using our proposed multi use-case synthesis framework (MUCSYN) as well as with the worst case (WC), and WC with DVFS approaches [20]-[21] (subsection V.A) for comparison purposes. Fig. 4 shows the cost of the crossbar (number of buses) of the best solution generated by these approaches. It can be seen that the solution generated by MUCSYN has significantly fewer buses (2.4× to 3.8×) than the WC approaches. The WC with DVFS approach has the same cost as the WC approach because DVFS only affects power dissipation, and does not reduce crossbar cost. MUCSYN creates a solution with a lower cost because of two main reasons: *(i)* it optimizes the use cases separately during topology selection unlike the WC approaches which combine all the use-cases into a single synthetic use case with much more conservative constraints, and *(ii)* it makes use of dynamic arbitration reconfiguration to better match use-case characteristics and lower the cost of the crossbar compared to WC approaches.

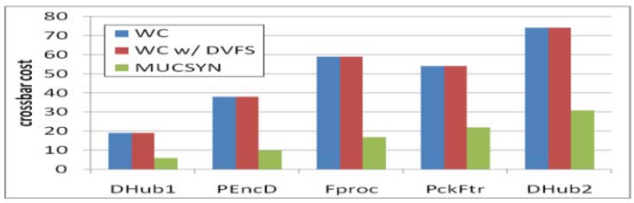


Fig. 4. Cost comparison for synthesized crossbars

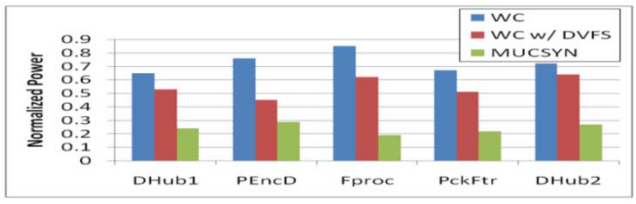
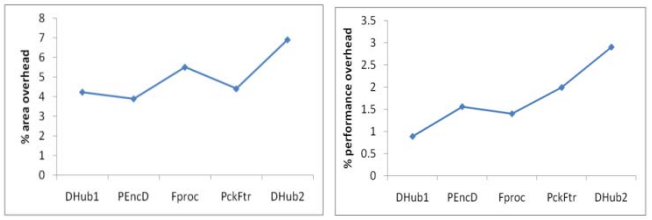


Fig. 5. Power dissipation for synthesized crossbars

Fig. 5 shows a comparison of the power dissipation of the crossbar solutions generated by the approaches, normalized to the power dissipation of a full crossbar solution. It can be seen that MUCSYN has a much lower power dissipation compared to the WC approaches. The WC approach has a lower power dissipation compared to the full crossbar because of the fewer buses and router components in its partial crossbar solution. The

WC with DVFS further reduces power dissipation further by opportunistically scaling voltage/frequency during application execution. MUCSYN outperforms these WC approaches because it has much fewer buses and router components, which coupled with the DVFS technique dissipates significantly lower power (1.5× to 3.1×).



(a) area overhead (b) performance overhead

Fig. 6. Dynamic reconfiguration overhead

Fig. 6 (a) shows the area overhead of the dynamic reconfiguration logic required for the solution obtained using our proposed framework. The area overhead is due to the DVFS circuitry and logic for multiple arbitration scheme selection and operation. The area overhead is less than 7%, and largely due to the DVFS circuitry. Fig. 6 (b) shows the performance overhead of using DVFS and dynamic arbitration reconfiguration to reduce cost and power dissipation in the crossbar solution generated by our framework. It can be seen that the performance overhead of runtime reconfiguration is minimal (less than 3%). Such a low overhead framework for generating dynamically reconfigurable crossbar architectures is extremely valuable for designers of multi use-case CMP applications.

VII. Conclusion

In this paper, we proposed a novel framework that synthesizes crossbar-based communication architectures for CMP applications with multiple use cases. The generated crossbars support dynamic reconfiguration of the arbitration schemes and dynamic voltage and frequency scaling (DVFS) to better adapt to the changing needs of different use cases which can frequently switch at runtime. Experimental results on several networking domain case studies indicate that our proposed framework generates a lower cost crossbar (2.4× to 3.8×) compared to traditional and previously proposed synthetic worst case design approaches. The synthesized crossbar from our framework dissipates much lower power (1.5× to 3.1×) than these approaches. In addition, the dynamic reconfiguration area and performance overhead of the synthesized crossbar solution is minimal. Such a framework for generating dynamically reconfigurable crossbars is invaluable for emerging CMP applications.

Acknowledgements

This research was partially supported by grants from SRC (2005-HJ-1330 and 1617.001) and NSF (CCF-0702797).

References

[1] K. Olukotun et al., “The Case for a Single-Chip Multiprocessor,” In Proc. ASPLOS, 1996, pp. 2-11.
[2] A. Covell, F, Whyte, “Digital Convergence: How the Merging of Computers, Communications and Multimedia is Transforming Our Lives”, Aegis Publishing Group, 1999.
[3] International Technology Roadmap for Semiconductors, 2006, http://www.itrs.net/
[4] R. Ho, W. Mai, and M. A. Horowitz, “The future of wires”, Proc. of IEEE, 89(4):490–504, April 2001.
[5] S. Pasricha, and N. Dutt. “On-Chip Communication Architectures”, Morgan Kauffman, Apr 2008.
[6] P. Kongetira, K. Aingaran, K. Olukotun, "Niagara: a 32-way multithreaded Sparc processor," IEEE Micro, pp. 21-29, Mar-Apr 2005.
[7] M. Rutten et al., “Dynamic reconfiguration of streaming graphs on a heterogeneous multiprocessor architecture,” Electron. Imag., 2005.
[8] M. Gasteier, M. Glesner “Bus-based communication synthesis on system level”, In ACM TODAES, Jan 1999.
[9] N. Thepayasuwan, A. Doboli, “Layout conscious bus architecture synthesis for deep submicron systems on chip”, In Proc. DATE 2004.
[10] S. Pandey, et al., “On-chip communication topology synthesis for shared multi-bus based architecture”, In Proc. FPLA, 2005.
[11] S. Murali, G. De Micheli, “An Application-Specific Design Methodology for STbus Crossbar Generation”, In Proc. DATE 2005.
[12] S. Pasricha, N. Dutt, M. Ben-Romdhane, "Constraint-Driven Bus Matrix Synthesis for MPSoC", In Proc. ASPDAC 2006.
[13] A. Hansson et al., “A unified approach to constrained mapping and routing on network-on-chip architectures””In Proc. ISSS 2005.
[14] J. Hu, R. Marculescu, “Exploiting the Routing Flexibility for Energy/Performance Aware Mapping of Regular NoC Architectures”, In Proc. DATE 2003.
[15] S. Murali, G. De Micheli, “Bandwidth-Constrained Mapping of Cores onto NoC Architectures”, In Proc. DATE 2004.
[16] K. Sekar et al., “FLEXBUS: a high-performance system on-chip communication architecture with a dynamically configurable topology”, In Proc. DAC 2005
[17] K. Lahiri, et al., “Design of high-performance system-on-chips using communication architecture tuners”, IEEE TCAD, May 2004.
[18] T.D. Richardson, et al., “A hybrid SoC interconnect with dynamic TDMA-based transaction-less buses and on-chip networks”, In Proc. VLSI Design (VLSID), 2006. pp. 8-15.
[19] S. Pandey, T. Murgan, M. Glesner, “Energy Conscious Simultaneous Voltage Scaling and On-Chip Communication Bus Synthesis”, In Proc. VLSI-SoC 2006.
[20] S. Murali et al. “Mapping and configuration methods for multi-use-case networks on chips”. In Proc. ASP-DAC, 2006.
[21] S.Murali et al. “A methodology for mapping multiple use-cases on to networks on chip”, In Proc. DATE, 2006.
[22] A. Hansson, M. Coenen, K. Goossens, "Undisrupted Quality-of-Service during Reconfiguration of Multiple Applications in Networks on Chip," In Proc. DATE 2007
[23] A. Hansson, K. Goossens, “Trade-Offs in the Configuration of a Network on Chip for Multiple Use-Cases”, In Proc. of NOCS 2007.
[24] AMBA AXI PL300 www.arm.com/support/PL300_ACI.html
[25] “STBus Communication System: Concepts and Definitions”, Reference Guide, STMicroelectronics, May 2003
[26] T. Grötker, S. Liao, G. Martin, S. Swan. “System Design with SystemC”. Kluwer Academic Publishers, 2002.
[27] S. N. Adya, I. L. Markov, "Fixed-outline Floorplanning: Enabling Hierarchical Design", IEEE Trans TVLSI, pp. 1120-1135, Dec. 2003.
[28] S. Pasricha, Y. Park, F. Kurdahi, N. Dutt, “System-Level Power-Performance Trade-Offs in Bus Matrix Communication Architecture Synthesis”, In Proc. CODES+ISSS 2006.
[29] J. M. Rabaey, “Digital Integrated Circuits”, Prentice Hall, 2002.
[30] Ji-Hoon Lim et al., “A Novel High-Speed and Low-Voltage CMOS Level-Up/Down Shifter Design for Multiple-Power and Multiple-Clock Domain Chips”, IEICE Trans. Electron., Mar 2007.
[31] K. Srinivasan, K. S. Chatha, G. Konjevod, “An automated technique for topology and route generation of application specific on-chip interconnection networks”, In Proc. of ICCAD 2005.
[32] R. J. Bril et al., “Multimedia QoS in consumer terminals,” in Proc. SIPS, 2001.
[33] S. Pasricha, "Transaction Level Modeling of SoC with SystemC 2.0" In Proc. SNUG 2002.
[34] S. Pasricha, N. Dutt, M. Ben-Romdhane, "Extending the Transaction Level Modeling Approach for Fast Communication Architecture Exploration", In Proc. DAC 2004.