Frugal but Flexible Multicore Topologies in Support of Resource Variation-Driven Adaptivity

Chengmo Yang
Electrical and Computer Engineering
University of Delaware
140 Evans Hall, Newark, DE 19716
email: chengmo@udel.edu

Alex Orailoglu
Computer Science and Engineering
University of California, San Diego
9500 Gilman Drive, La Jolla, CA 92093
email: alex@cs.ucsd.edu

***Abstract*—Given the projected higher variations in the availability of computational resources, adaptive static schedules have been developed to attain high-speed execution reconfiguration with no reliance on any runtime rescheduling decisions. These schedules are able to deliver predictable execution despite the increased levels of device unreliability in future multicore systems. Yet the associated runtime reconfiguration overhead is largely determined by the underlying system topology. Fully connected architectures, although they can effectively hide the overhead in execution migration, become infeasible as the core count grows to hundreds in the near future. We exploit in this paper the high locality associated with adaptive static schedules, and outline a *scalable* and *locally shareable* system organization for multicore platforms. With the incorporation of a limited set of neighborhood-centered communication links, threads are allowed to be directly migrated among adjacent cores without physical data movement. At the architecture level, a set of 2-dimensional physical topologies with such a local sharing property embedded is furthermore proposed. The inherent regularity allows these topologies to be adopted as a fixed-silicon multicore platform that can be flexibly redefined according to the parallelism characteristics and resilience needs of each application.**

I. Introduction

While Multiprocessor System-on-Chip (MPSoC) architectures [1], [2], [3] are slated to dominate the next computer generation, the limitations of such systems in providing solutions to an ever more diverse set of applications become apparent. In particular, the execution environment is becoming more dynamic and unpredictable, partially because of the expected degradation in device reliability, and partially because of the more diverse and complex application set. During execution, MPSoCs may encounter higher variations in the availability of computational resources that can be induced by various types of device failures [4], [5], thermal stress [6], or resource competitions.

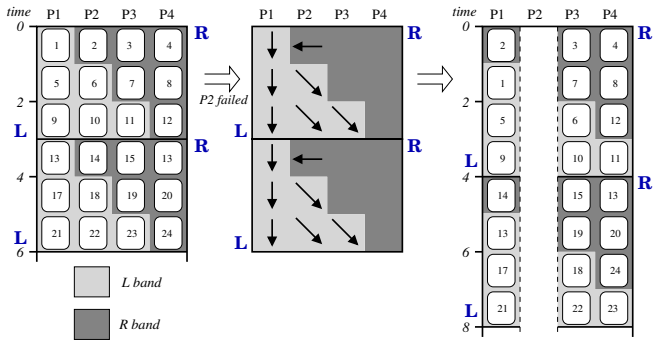
The expected high variations in runtime resource availability dictate the need for an adaptive organization of computations in future MPSoCs. Such systems should be able to *reconfigure* the execution to either withstand a core unavailability, or make use of a previously deallocated core once the cause of unavailability has been cleared. Typically a reconfiguration process can be decomposed into two parts, namely, the generation of runtime re-scheduling decisions, and the migration of the code and data sets of the threads being re-scheduled. The challenge, however, is to minimize decision-making overhead and workload-migration overhead *simultaneously*. Moreover, as the interconnect cost becomes increasingly expensive in terms of both power and performance, execution migration should also be confined within a neighborhood.

978-3-9810801-7-9/DATE11/©2011 EDAA

To minimize the decision making overhead, *adaptive static schedules* [7], [8] have been proposed. By executing sophisticated planning at compile-time, a set of possible execution schedules with diverse resource demands can be compactly captured in advance. In this way, upon run-time resource variations, threads can be migrated in a regular manner with no reliance on dynamic rescheduling. While these schedules are able to deliver predictable reconfiguration processes, to minimize the overall reconfiguration overhead, they still need to be supported by a flexible customization of the underlying MPSoC topology so as to effectively hide the overhead in transferring threads between cores.

While fully connected architectures [9] can effectively hide migration overhead, they become infeasible as the core count grows to hundreds in the near future. Instead, we exploit in this paper the fact that in adaptive static schedules [7], [8] execution migration only involves a limited set of adjacent cores. This high locality indicates that reconfiguration-induced code/data transfer can be eliminated even in a distributed architecture that is envisioned as the dominant architecture for future MPSoCs. In light of this observation, we herein propose a *scalable* and *locally shareable* system organization for MPSoCs that provides neighborhood-centered, dedicated communication links for adjacent cores to access and share a single storage unit in common. This organization enables the simultaneous migration of multiple threads between distinct processing elements (PEs) without any physical movement of the code/data set. More importantly, as such a *locally shareable* system organization is independent of a particular topological structure, we furthermore propose a set of 2-dimensional physical topologies that can be adopted as fixed-silicon but dynamically reprogrammable MPSoC platforms. Such topological structures allow on-chip networks to be customized according to parallelism characteristics and resilience needs of an application, while at the same time delivering the benefit of high-volume amortization.

The rest of the paper is organized as follows. Section II briefly reviews adaptive static schedules and then outlines the technical motivation. Section III presents in detail the proposed locally shareable system organization model, while Section IV presents a set of 2-dimensional physical topologies that can be matched to different levels of reconfiguration and communication needs. The efficacy of the proposed technique is experimentally verified in Section V, while Section VI offers a brief set of conclusions.



(a) Band–level partition (b) Reassignment directions (c) Schedule after reconfiguration

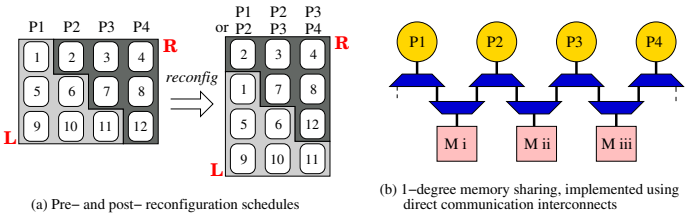
Fig. 1. Canonical case of band-level reconfiguration

II. Technical Motivation

Given the projected higher variations in the availability of computational resources, *adaptive static schedules* [7] have been developed to attain predictable execution despite the increased levels of execution unpredictability in future MPSoCs. These schedules exploit the global optimization capability offered by static scheduling techniques. To attain dynamic adaptivity, these schedules are partitioned into regular yet shiftable *band structures*. A representative schedule example is presented in Fig. 1a, with each rectangle labeled with a number denoting a thread and each column representing one processing element (PE) of the MPSoC. As can be seen, the schedule is horizontally partitioned into two blocks, and each block is furthermore divided into a *Left (L) band* and a *Right (R) band*. The regular shape of the L and the R band delivers highly regular execution reconfiguration, achievable *independent* of the PE being removed. As an example, Fig. 1b uses arrows to indicate the workload migration directions to isolate P_2. It can be seen that in each schedule block the position of the R band remains intact, while the whole L band is shifted one timing step down and one PE to the right. As a result, in the *post*-reconfiguration schedule presented in Fig. 1c, each schedule block can be completed with one fewer PE, albeit with an additional timing step.

The most important benefit of these adaptive schedules is the high temporal and spatial locality associated with the reconfiguration process. Not only do all the threads within a single band share the identical timing offset, but furthermore thread movements are restricted to shiftings between adjacent PEs. Clearly, upon runtime resource variations, reconfiguration can be performed in a regular manner without any rescheduling decisions being made on the fly. The overall reconfiguration overhead is therefore determined solely by the overhead in transferring the threads between PEs, which is in turn highly influenced by the topology of the underlying MPSoC.

In distributed architectures, PEs are connected through on-chip networks [10], thus requiring the code and data sets to be transferred through the network to accomplish thread migration. In contrast, in fully connected architectures [9], PEs communicate typically through a centralized, globally shared memory. Thread migration can therefore be accomplished through remapping the associated address space with no real movement of code or data sets. Unfortunately, fully connected architectures become infeasible as the core count grows to hundreds in the near future.



(a) Pre– and post– reconfiguration schedules (b) 1–degree memory sharing, implemented using direct communication interconnects

Fig. 2. Reconfiguration-induced sharing requirements

The increase in the number of PEs requires both the size and the number of ports of the memory to scale proportionally, necessitating in turn aggressive cache consistency and coherency protocols and hence inducing significant power and performance overhead.

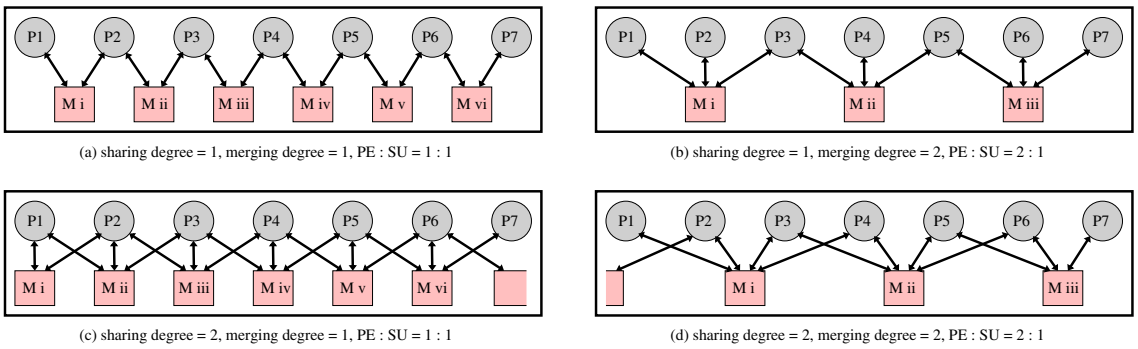
The aforementioned analysis confirms that the attainment of light-weight execution migration requires the PEs to be organized in a *shareable* and *scalable* manner. While these two goals seem contradictory at first sight, the high spatial locality inherent in the adaptive schedules [7] enables their simultaneous attainment. This property can be illustrated more clearly through examining the adaptive schedule shown in Fig. 2a. Each column of the post-reconfiguration schedule can be executed either on core P_i or P_{i+1}, depending on the position of the deallocated PE. A detailed comparison of the pre- and post-reconfiguration schedules indicates that a thread initially executed on PE P_i may need to be migrated *right* to PE P_{i+1}, if the thread is in the *L* band. Similarly, if the thread is in the *R* band, it may need to be migrated *left* to PE P_{i-1}. As an example, if P_3 fails, among the threads that were initially scheduled on it, Thread 7 needs to be migrated to P_2 , while Thread 11 needs to be migrated to P_4. In either case, execution migration is only performed between **two** adjacent PEs, implying that the sharing of a single storage unit between two PEs suffices for eliminating the reconfiguration-induced thread movements. This observation motivates us to propose a **locally shareable** system organization model, wherein localized, dedicated communication links are employed to attain the twin requirements of *scalability* and *shareability*.

III. Locally Shareable Storage Organization

A. Modeling of memory and interconnect organizations

In the proposed locally shareable organization, storage units (SUs) are organized in a distributed form across the entire platform, while sharing is achieved through enabling each PE to directly access multiple SUs. These SUs, which can be either L2 caches or on-chip memory units, are connected through a conventional **on-chip network** for the support of long-distance communications, while the PEs and SUs are directly connected through **point-to-point communication links**. These communication links enable multiple threads to be *simultaneously* migrated between distinct PEs without inducing any interferences or network congestion. As these links are highly localized, they impose negligible hardware cost and routing overhead.

The examination of the adaptive schedule shown in Fig. 2 indicates that the sharing of a single storage unit between two PEs suffices for eliminating the reconfiguration-induced code/data movements. More generally, schedules capable of tolerating a variation of m PEs [8] require a single storage unit to be shared between every group of $m+1$ adjacent PEs. This sharing



(a) sharing degree = 1, merging degree = 1, PE : SU = 1 : 1

(b) sharing degree = 1, merging degree = 2, PE : SU = 2 : 1

(c) sharing degree = 2, merging degree = 1, PE : SU = 1 : 1

(d) sharing degree = 2, merging degree = 2, PE : SU = 2 : 1

Fig. 3. Bipartite graph representation of various topologies with distinct values of sharing degree and merging degree

requirement can be formally specified through a parameter of sharing degree, defined as follows:

> ***Sharing degree****: the extra number of cores with which a core P_i shares a single storage unit.*

Under this definition, the *minimum* amount of sharing degree required in an adaptive schedule of m reconfiguration steps is m. In comparison, the traditional distributed and centralized memory organizations can be viewed as two extreme cases with sharing degrees of 0 and $N-1$, with N denoting the total number of PEs in the platform. Clearly, the former falls short of fulfilling the reconfiguration-induced sharing requirements, while the latter offers overmuch sharing capability and hence suffers from the crucial limitation of the lack of *scalability*.

In this locally shareable organization, the number of communication links is directly determined by the amount of *sharing degree*. By definition, a sharing degree of s directly implies that each single SU is accessible to $s+1$ PEs. Given an MPSoC platform with a total number of N PEs and N SUs, the interconnect network thus exhibits the following characteristics:

$$\text{Average interconnects per PE} = s+1 \quad (1a)$$

$$\text{Average interconnects per SU} = s+1 \quad (1b)$$

$$\text{Total interconnects} = (s+1)\cdot N \quad (1c)$$

Equations (1a) and (1b) indicate that each PE and each SU are connected to multiple direct communication links. Yet this many-to-many interconnect network can still be attained without increasing the number of read/write ports of each PE. As a PE does not access distinct SUs simultaneously, a single read/write port, together with a set of decoders and multiplexers, suffices for the accesses from a single PE to multiple SUs. This type of organization is concretely illustrated in Fig. 2b that shows 4 PEs with a sharing degree of 1.

As each direct communication link connects a PE with an SU, the entire interconnect network can be modeled as a *bipartite graph*[1] between two disjoint sets of nodes, the PEs and the SUs. The bipartite graph representations of MPSoC platforms with sharing degree values of 1 and 2 are respectively shown in Fig. 3a and 3c. Any two consecutive PEs (P_i and P_{i+1}) in Fig. 3a share a single SU, while any three consecutive PEs (P_i, P_{i+1}, and P_{i+2}) in Fig. 3c share a single SU. Yet this increased sharing capacity is attained at a cost of increased interconnects, as Fig. 3c displays more communication links than 3a. This property is also confirmed by Equation (1c), which indicates that the total number of point-to-point communication links in this locally shareable organization is linearly proportional to the sharing degree s.

The larger the sharing capability is, the more the communication links needed in the system and the higher the consequent cost would be. To reduce such cost without sacrificing the sharing capacity, we exploit the flexibility of merging a number of adjacent SUs together, which in turn enables a combination of the communication links emanating from a single PE to these SUs. Here, we formally define the parameter ***merging degree*** as follows:

> ***Merging degree****: the number of SUs that have been merged to form a single SU, which is equal to the ratio of the number of PEs over the number of SUs post-merging.*

Fig. 3b and 3d respectively present the bipartite graphs generated through merging every two contiguous SUs in Fig. 3a and 3c. A comparison between these two sets of graphs confirms that link merging reduces the total number of SUs and increases the number of PEs that share an SU in common. More precisely, by merging k adjacent SUs into a single one, the total number of SUs is reduced from N to N/k, while every set of $s+k$ PEs shares an SU in common. The interconnect network thus exhibits the following properties as a result of link merging:

$$\text{Average interconnects per PE} = s/k+1 \quad (2a)$$

$$\text{Average interconnects per SU} = s+k \quad (2b)$$

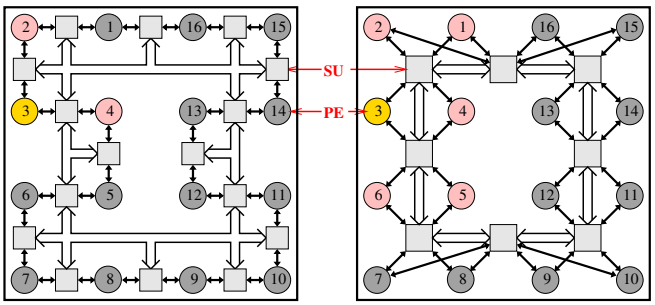
$$\text{Ports per SU} = k \quad (2c)$$

$$\text{Total interconnects} = (s/k+1)\cdot N \quad (2d)$$

A comparison between Equations (1c) and (2d) clearly confirms that link merging can effectively reduce the total number of direct communication links by k times roughly. In an extreme case, if the *sharing degree* equals the *merging degree* ($s=k$), the total number of communication links will always equal $2N$, independent of the s and k values. As a concrete example, the two bipartite graphs in Fig. 3a ($s=k=1$) and 3d ($s=k=2$) contain 12 links.

The sharing of an SU among multiple PEs offers the extra benefit of *adaptive resource allocation*. PEs do not need to be confined to the same amount of storage. Instead, the allocation of storage cells can be determined according to the total amount of storage required by each PE. Clearly, the amount of flexibility in storage

[1]For simplification purposes, the direct communication links in the subsequent parts of this paper are shown in the bipartite graph format instead of the format presented in Fig. 2b.



(a) sharing degree = 1, merging degree = 1 (d) sharing degree = 2, merging degree = 2

Fig. 4. Various 2-dimensional locally shareable MPSoC topologies

allocation is proportional to the *merging degree*. However, it needs to be noted that a SU post-merging needs to simultaneously serve accesses from more PEs, thus necessitating the number of ports per SU to increase linearly as a function of the merging degree. Such an increased complexity therefore imposes upper bounds on the value of the *merging degree*.

IV. Physical Topology and Application Mapping

The *locally shareable* storage model introduced in the last section is independent of a particular topological structure. Distinct 2-dimensional topologies that exhibit a varying amount of sharing and merging degrees therefore can be developed.

A. Topology instances and the associated properties

Fig. 4 presents two representative 2-dimensional physical topologies corresponding to the bipartite graph representations shown in Fig. 3a and 3d. PEs and SUs are connected to each other through the direct inteconnect links, while all the SUs are connected through an underlying on-chip network for the support of long-distance communication and data transfer. It can be observed that these topology instances exhibit the following properties:

- In a topology with a sharing degree of s, every group of $s+1$ cores with consecutive indices (P_i, P_{i+1}, ..., and P_{i+s}) consistently hold in common an SU. As a result, the platform is able to obviate any reconfiguration-induced data transfer among any $s+1$ adjacent PEs, thus effectively minimizing execution migration overhead. In an extreme case, if more than s cores become unavailable, the toleration of the remaining cores requires data to be transferred through the underlying on-chip network that connects all the SUs.
- In both topologies, each PE has at least **two** direct communication links. These topologies therefore tolerate single failures of direct communication links, as single failures can block at most one of the communication paths. A detailed examination shows that this fault tolerace capability is directly determined by the sharing degree s and the merging degree k. More precisely, a merging of k out of s links may reduce the minimum number of interconnects per PE to $s-k$. Accordingly, as long as $s-k \geq 0$, each PE is guaranteed to have at least 2 direct communication links.

The examination above confirms that both MPSoC topologies shown in Fig. 4 deliver the capability of tolerating device failures in the PEs and the communication links. The failure of a set of storage cells would hardly have any impact on the connectivity of the entire multicore platform. Therefore, these topologies can be employed by various application sets to attain fault tolerance and adaptive execution.

B. Topology instance selection

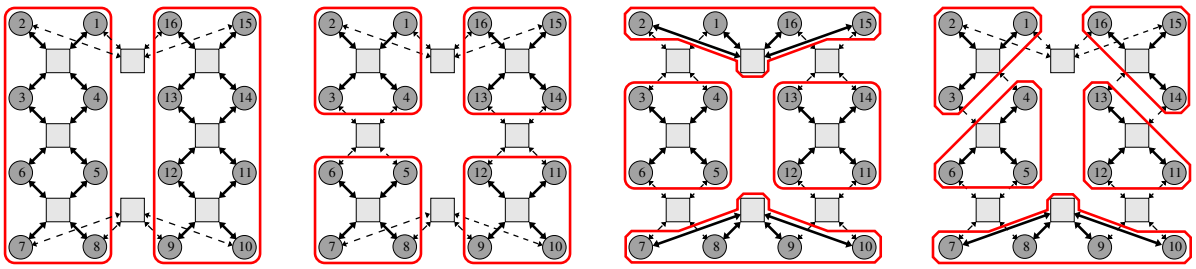
While the topology instances presented in Fig. 4 provide a varying amount of *sharing capability*, they all exhibit a regular structure in that the sharing and merging conditions of each PE are identical. Such regularity enables these topologies to be adopted as fixed-silicon MPSoC platforms, thus providing the benefits of high-volume amortization. Meanwhile, such regularity also delivers flexible redefinitions of the platform to match parallelism characteristics and resilience needs of the application.

The selection of the most suitable topologies for diverse application sets can be based on two considerations. On one hand, the values of sharing and merging degrees should be determined according to the required reconfiguration needs as well as the inter-PE communication patterns. On the other hand, consideration of design complexity imposes upper bounds on the values of sharing and merging degrees.

1) Parallelism consideration: While the number of PEs in either of the two topologies in Fig. 4 is fixed to 16, this parameter can be easily customized to other values without impacting the structures of the bipartite graphs. Topologies with a larger number of PEs can be used to hold applications with large amount of thread-level parallelism. On the other hand, if the application cannot be parallelized into dozens of independent threads, the PEs can be partitioned into a number of *finer-grained* clusters so as to enable simultaneous execution of multiple applications on the fabric. Fig. 5 presents four distinct ways for partitioning the 16 PEs into 2, 4 and 5 clusters. During execution, communications are only performed within each cluster and hence the inter-cluster communication links can be deactivated (the dashed lines in Fig. 5). Yet upon the failure of a communication link, the platform can be *repartitioned* so as to utilize a different set of communication links. This property can be observed through a comparison between Fig. 5b and 5c, wherein the 4-cluster partitions display disjoint sets of active communication links.

2) Reconfiguration and communication consideration: To select an appropriate topology instance for an application, both the reconfiguration needs and the communication patterns of each application should be considered. First of all, to completely eliminate the reconfiguration-induced data transfer, the number of PEs that can directly access a single SU should be no less than the number of reconfiguration steps m embedded within the schedule. Clearly, this topology selection criterion can be directly determined, once the value of m has been determined according to the occurrence frequency of core unavailability within the system.

The topology selection criterion that concerns the communication characteristics is application-specific. Previous studies [11] have shown that most applications display steady communication patterns in that a thread regularly communicates with a small and fixed subset of the rest of the threads. Such communication patterns can be extracted and analyzed at compile time. Clearly, as the interconnect cost becomes increasingly expensive in terms of both power and performance, it is more cost-effective to communicate through the neighborhood-centered, direct links rather



(a) 2–cluster partitioning (b) 4–cluster partitioning (c) 4–cluster partitioning (d) 5–cluster partitioning

Fig. 5. Finer-grained cluster partitions and communication link utilization

than through the underlying on-chip network. In order to do so, the number of *direct neighbors* of a PE should be no less than the maximum number of threads involved in a single communication. Here, the *direct neighbors* of a PE P_i are defined as the PEs that share an SU with P_i.

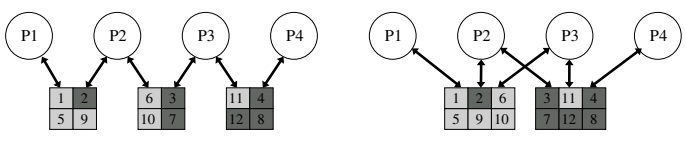
The value of the *direct neighbors* of a PE is determined by the *sharing degree* and the *merging degree*. By definition, a sharing degree of s directly implies that every $s+1$ cores with consecutive indices consistently share a single SU in common. Accordingly, core P_i always has at least $2s$ direct neighbors with contiguous indices, ranging from P_{i-s} to P_{i+s}. This property can be observed in the topologies shown in Fig. 4, wherein the direct neighbors of PE P_3 are shown in pink. In Fig. 4a ($s = 1$), P_3 has P_2 and P_4 as its direct neighbors, while in Fig. 4b ($s = 2$), P_3 has P_1, P_2, $P4$, and $P5$ as its direct neighbors. Meanwhile, as the link merging process increases the number of PEs that share a single SU, in Fig. 4b, P_3 additionally has P_6 as its direct neighbors. In sum, a detailed examination indicates that the average number of direct neighbors of a PE is $2s + k - 1$.

3) Design complexity consideration: While the consideration of reconfiguration and communication needs argues for a topology with larger values of sharing and merging degrees, the consideration of design complexity constrains the number of interconnects. As mentioned before, a larger value of the *sharing degree* increases the total number of interconnects, while a larger value of the *merging degree* requires both the size and the number of ports of the SU to scale proportionally. These considerations therefore impose upper bounds on the values of sharing and merging degrees.

C. Thread Placement Requirements

As each PE can access multiple SUs in the proposed system organization, there exist multiple choices for placing the code and data set of each thread. Upon the selection of the most suitable topology for an application, the associated thread placement decisions can be made, based on the pre-generated adaptive schedule.

As we examined in Section II, a thread initially scheduled on P_i may end up being migrated either to P_{i+1} if it is in the L band, or to P_{i-1} if it is in the R band. Thread placement decisions can therefore be made accordingly. In brief, threads in the L band should be placed in the SU shared between P_i and P_{i+1}, while threads in the R band should be placed in the SU shared between P_i and P_{i-1}. As an example, data placement decisions of the schedule presented in Fig. 2a are shown in Fig. 6a. Taking P_2 as an example, among the three threads originally scheduled on it, Thread 2, lying in the R band, is placed in the SU shared between P_1 and P_2, while Threads 6 and 10, lying in the L band, are placed in the SU shared between P_2 and P_3.



(a) 1–degree memory sharing and data placement (b) 2–degree memory sharing and data placement

Fig. 6. Reconfiguration-induced memory sharing and data placement

If there exist multiple valid selections for the placement of a thread, the decisions are made so as to balance the amount of storage. This occurs if the *sharing degree* of the selected MPSoC topology exceeds the maximum reconfiguration step needed by the application. For instance, the bipartite graph shown in Fig. 6b exhibits a sharing degree of 2. The threads in Fig. 2a that will never be executed by P_1 or P_4, namely, Threads 3, 6, 7, and 10, can be placed in either SU. These threads can then be evenly split into the two SUs in various ways, with one possible solution shown in Fig. 6b. The only constraint is to separate Threads 6 and 7. As the two threads are executed at the same timing step in the pre-reconfiguration schedule, placing them in the same SU would require an increase in the number of access ports.

The data placement decisions above are made under the assumption that the selected MPSoC topology exhibits a merging degree of 1. These decisions display the finest granularity, thus resulting in their applicability to an MPSoC of merging degree $k > 1$, as the data placement can also be merged along with the SUs. If, in an extreme case, all the PEs of an application share an SU in common (due to the lack of parallelism), no data placement decisions need to be made.

V. Experimental Evaluation

To evaluate the efficacy of the proposed locally shareable model, the two topology instances presented in Fig. 4 are compared against the traditional distributed and centralized storage models. The algorithm outlined in [7] has been implemented in C for generating adaptive static schedules. The application set under test is composed of typically parallel algorithms, such as *LU decomposition*, *Laplace equation solver*, and *Gaussian elimination*. DAG representations of these thread graphs can be found in [12].

To evaluate the efficacy of the proposed topologies in eliminating reconfiguration-induced data movement, we report the number of threads that need to be transferred through the on-chip network, if a PE becomes unavailable. The obtained results

TABLE I. IMPACT OF MPSOC TOPOLOGY ON SCHEDULE LENGTH AND THREAD MAPPING

		Reconfiguration cost				*Schedule length ratio*				*Direct communication ratio*			
Benchmark	Thread #	TP0-0	TP1-1	TP2-2	TPn-n	TP0-0	TP1-1	TP2-2	TPn-n	TP0-0	TP1-1	TP2-2	TPn-n
FFT	95	15	0	0	0	1	0.82	0.71	0.53	0	0.41	0.62	1
Fork-join	51	2	0	0	0	1	0.90	0.81	0.52	0	0.05	0.18	1
Gaussian	81	8	0	0	0	1	0.52	0.52	0.52	0	1.00	1.00	1
Laplace	75	5	0	0	0	1	0.96	0.88	0.80	0	0.41	0.87	1
LU	65	5	0	0	0	1	0.96	0.88	0.80	0	0.38	0.87	1
Tree	63	4	0	0	0	1	0.82	0.73	0.64	0	0.86	0.95	1
Average	72	6	0	0	0	1	0.83	0.76	0.64	0	0.52	0.75	1

are presented in Table I. We use "TPs-k" to denote a topology with sharing degree s and merging degree k. TP0-0 and TPn-n therefore correspond to distributed and centralized storage models, respectively. As can be seen, both proposed topologies are able to completely eliminate reconfiguration-induced data movement, thus minimizing reconfiguration overhead.

To illustrate the impact of MPSoC topology on thread scheduling, the selected benchmarks are scheduled onto each of the 4 topologies. The obtained results of schedule lengths (normalized to the TP0-0 case) are also presented in Table I. On average, the TP1-1 and TP2-2 topologies reduce the schedule lengths by 17% and 24%, respectively. Except for *Fork-join*, the attainable schedule length results of the TP2-2 topology are comparable to the results of the conventional centralized system model, indicating that the remaining five benchmarks can be effectively accelerated by the proposed locally shareable system organization. The fundamental reason for this improvement is that these applications display a large amount of parallelism and a limited number of out-going communications per thread, implying that most of the inter-thread communications can be performed through the direct communication links. To illustrate this property, we additionally report, in Table I, the percentage of inter-PE communications that can be performed through the *direct communication links* in the topology. The results show that except for *Fork-join*, the remaining five benchmarks can perform more than 60% inter-PE communications through the direct communication links in the TP1-1 topology, and more than 85% communications through the links in the TP2-2 topology. In comparison, in *Fork-join*, a thread may fork a large number of dependent threads that cannot simultaneously be placed on direct neighbors of the corresponding PE. The schedule length is thus constrained by the longest remote communication.

The values reported in Table I confirm that the proposed locally shareable topologies offer multiple design points for the designer to trade off between the schedule length and the number of direct communication links. A reduction in the schedule length can generally be attained through increasing the number of direct communication links. On the other hand, it would not be cost-effective if a negligible reduction in the scheduling length would end up requiring a significant number of communication links. According to this criterion, TPn-n can be considered as the most appropriate topology for *FFT* and *Fork-join*, while TP2-2 deemed the most appropriate topology for the remaining 4 applications.

VI. CONCLUSIONS

As computational resources increasingly become unavailable at runtime, a fast and predictable *execution reconfiguration* step is necessitated upon a resource variation, which in turn requires the development of advanced MPSoC topologies that can effectively hide thread migration overhead. To attain this goal, we have proposed a locally shareable storage organization for adaptive multicore platforms in this paper. Through making each storage unit directly accessible to a set of adjacent PEs, threads can be directly migrated among these PEs without data movement. At the architecture level, a set of 2-dimensional physical topologies with such a local sharing property embedded is furthermore proposed. Such topological structures can be adopted as fixed-silicon but dynamically reprogrammable MPSoC platforms, wherein decisions regarding topology selection and thread placement can be made according to the parallelism characteristics of the application and reconfiguration requirements of the system. The experimental results confirm that the proposed MPSoC topologies can even halve the execution time of parallel applications, while the reconfiguration-induced data movements between adjacent PEs can be completely eliminated. The utilization of these topologies, in conjunction with adaptive static schedules, brings high-efficiency and predictability into execution despite the increasing amount of uncertainty in future MPSoCs.

REFERENCES

[1] P. Kongetira, K. Aingaran, and K. Olukotun, "Niagara: a 32-way multi-threaded Sparc processor," *IEEE Micro*, vol. 25, no. 2, pp. 21–29, 2005.

[2] R. Kota and R. Oehler, "Horus: large-scale symmetric multiprocessing for Opteron systems," *IEEE Micro*, vol. 25, no. 2, pp. 30–40, 2005.

[3] M. Kistler, M. Perrone, and F. Petrini, "Cell multiprocessor communication network: Built for speed," *IEEE Micro*, vol. 26, no. 3, pp. 10–23, 2006.

[4] R. Blish and N. Durrant, "Semiconductor device reliability failure models," International SEMATECH, Tech. Rep., May 2000.

[5] C. Constantinescu, "Trends and challenges in VLSI circuit reliability," *IEEE Micro*, vol. 23, no. 4, pp. 14–19, July 2003.

[6] C. J. Lasance, "Thermally driven reliability issues in microelectronic systems: Status-quo and challenges," *Microelectron. Reliab.*, vol. 43, no. 12, pp. 1969–1974, Dec. 2003.

[7] C. Yang and A. Orailoglu, "Predictable execution adaptivity through embedding dynamic reconfigurability into static MPSoC schedules," in *5th CODES-ISSS*, Sept. 2007, pp. 15–20.

[8] C. Yang and A. Orailoglu, "Fully adaptive multicore architectures through statically-directed dynamic execution reconfigurations," in *VLSI-SoC*, Sept. 2010.

[9] D. E. Culler and J. P. Singh, *Parallel Computer Architecture*. Morgan Kaufmann Publishers, 1999.

[10] W. H. Ho and T. M. Pinkston, "A design methodology for efficient application-specific on-chip interconnects," *IEEE Trans. Parallel Distrib. Syst.*, vol. 17, no. 2, pp. 174–190, 2006.

[11] J. S. Vetter and F. Mueller, "Communication characteristics of large-scale scientific applications for contemporary cluster architecture," in *16th IPDPS*, Apr. 2002, pp. 27–36.

[12] Y.-K. Kwok and I. Ahmad, "Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors," *IEEE Trans. Parallel Distrib. Syst.*, vol. 7, no. 6, pp. 506–521, June 1996.