A Reconfigurable Architecture for Multicore Systems

Annie Avakian, Jon Nafziger, Amayika Panda, Ranga Vemuri
Department of Electrical and Computer Engineering
University of Cincinnati
Cincinnati, Ohio, USA
{avakiaam,nafzigjw,pandaaa}@mail.uc.edu, ranga.vemuri@uc.edu

***Abstract*—Various studies concluded that bus-based multiprocessor architectures outperform Network-on-Chip (NoC) architectures when the number of processors is relatively small. On the other hand, NoC architectures offer distinct performance advantages when the number of processors is large. This led to recent proposals for hybrid architectures where each node in a mesh-style packet-switched NoC architecture contains a bus-based subsystem with a small number of processors. Experimental results using select benchmarks demonstrated that these hybrid architectures offer superior performance when compared with purely bus based or purely NoC style architectures. Our studies indicate that while a hybrid architecture is preferable, the optimal number of processors on each bus subsystem varies based on the application. This number appears to vary between 1 and 8 depending on the communication requirements of the application. Further, various applications simultaneously executing on the same system require differing numbers of processors on each bus-based subsystem to minimize the overall throughput time. In this paper, we present a new reconfigurable NoC architecture which allows scalable bus-based multiprocessor subsystems on each node in the NoC. Following configuration, the system provides a multi-bus execution environment where each processor is connected to a bus and the bus-based subsystems communicate via routers connected in a mesh-style configuration. The system can be reconfigured to vary the number of bus subsystems and the number of processors on each subsystem. Each processor contains a Level 1 (L1) cache and each bus, connected to a router, has access to a Level 2 (L2) cache. The L2 caches distributed across the network together form a large virtual L2 that can shared by all the processors in the system via the router network. We present the architecture in detail, discuss a configuration algorithm, and discuss experimental results (using the NS2 and SIMICS simulators) on standard and synthetic benchmarks indicating the performance advantages of the proposed architecture.**

***Index Terms*—multicore; reconfigurable; architecture; network on chip;**

I. Introduction

According to David House's extension of Moore's Law [1] demand on computing power doubles approximately every 20 months. As the industry continuously reacts to meet this growing demand, new ideas must be explored. Traditionally this has been met by increasing the clock speed of individual processors. Unfortunately, the diminishing returns in increasing processing speed after passing the 3 GHz plateau have forced exploration of other alternatives. This has lead to a new trend in increasing the total number of processor cores in a single chip [2]. In September of 2006, Intel announced the development of a single processor design containing 80 processor cores [3]. While simply increasing the total number of processor cores on a chip is straight-forward, connecting them together to provide a harmonious system is proving to be among the most critical design tasks. The simplest method used to allow communication between processor cores is the bus design. Each processor core maintains a unique L1 cache while all processor cores share a single L2 cache. However experiments have shown that the communication delay between the L2 cache and the individual processor cores increases significantly as the number of processor cores increases. Therefore this is not a viable method of developing a system with more than a handful of processor cores.

Different approaches have been tried in order to reduce communication latency, and so far Network-on-Chip (NoC) has come out as the winner. In the NoC scheme, processor cores are connected to routers and communication between cores is done using protocols similar to a computer network. While processor cores have a private L1 cache, the L2 cache is distributed and shared among all the cores. Concentrating on the hardware aspects alone cannot yield a good architecture. As we move toward parallel computation, each process is expected to be parallelized and composed of several threads working on the same data set. If the L2 cache is distributed among routers [4], then the latency to obtain the data varies depending on the network hops required in the NoC architecture. Physical proximity plays an important role in this scheme. The closer the L2 cache, the fewer hops is needed to obtain the requested data.

To reduce communication latency in an NoC architecture, several proposals have been introduced. One such proposal is a global sub mesh interconnect [5]. The authors propose using hierarchical rings for global routing and local rings for local routing. Using this method, Bourduas and Zilic showed they can achieve significant gains in number of hops for large mesh sizes. Another approach to reduce interconnect latency was proposed by Bolotin et al. [6]. The authors proposed reducing cache access time by introducing priority support at the hardware level. They also proposed using virtual invalidation ring, or an efficient way to store and forward short messages.Knauerhase et al [7] offer a different approach. They show the operating system can take advantage of the underlying architecture better if it learns from the information

978-1-4244-6534-7/10/$26.00 ©2010 IEEE

garnered from run time task behavior. While these methods reduce cache access time, they do not take advantage of the characteristics of parallel applications. On the other hand, Das et al. [8] suggest having eight processor cores on a bus connected to a router as a hybrid bus-mesh architecture. While this approach does take advantage of the inherent properties of multithreaded applications, the architecture is not flexible. Fixing the number of processors to eight per bus may not be optimal to all applications. Section II of this paper presents some experimental data concerning this.

We propose a reconfigurable hybrid bus-network architecture, where the number of processor cores attached to each bus is reconfigurable and is dependent on the needs of the active processes and applications. By configuring switches, multiple buses are created each with varying number of processor cores. The buses are then connected to the routers. The configuration of the switches remains the same during the lifetime of the running process.

The paper is organized as follows: Section II discusses the motivation behind the proposed architecture. Section III describes the proposed architecture. Section IV gives a detailed analysis of the benefits of the proposed hybrid architecture. Section V concludes.

II. Motivation

In this section, we will present experimental analysis of several simulated benchmark data sets to understand the benefits of bus-based vs. NoC architectures. We set four goals for the proposed architecture.

A. Exploiting Data Access Characteristics

The first step in developing a multiprocessor architecture is to understand the data access patterns of typical processes and applications executing on the architecture. Exploiting the temporal and spatial localities is essential. Therefore recognizing that threads of the same process will use the same data, grouping those threads becomes the first priority. Having a hybrid model of bus-network setup, where threads communicate locally via a bus and globally via a network can greatly reduce communication time between caches and processor cores. Buses are faster when data accesses are not intensive, and furthermore the area is much smaller when compared to assigning one router to each processor core. To show the benefits of a hybrid model, we simulated both the NoC architecture Fig 1 and the hybrid architecture Fig 2. Both simulations were done in C++. In the NoC architecture, each router has a processor core. The cores have private L1 caches, while the L2 cache is shared and is equally distributed among the routers.

In this scenario, a processor sends a request for memory access. The router acknowledges the request, and sends a packet to the router with the corresponding address. The memory request process time is measured as the number of hops the request traveled to reach the destination plus the number of hops for the return path. In our example, routers can process one request at a time, therefore if a router has

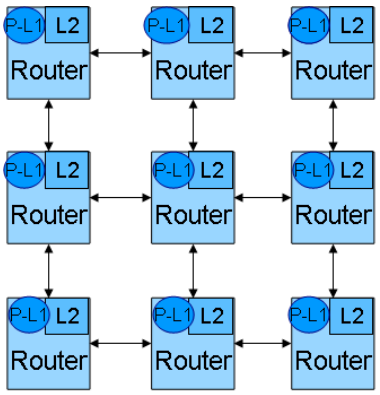

Fig. 1: NoC Architecture

pending requests from other processor cores, then the request is queued in the order of arrival. This delay is also added to the overall request process time. All links are considered uniform and have a unit time delay. The overall result of the simulation is the average processing time of all the memory requests generated by the processor cores. In the hybrid bus-NoC model, p number of processor cores are connected to a bus. The bus in turn is connected to the router. The L2 cache is still distributed shared L2. However, the size of the L2 cache in each router is p times larger than the first case, since p processor cores are connected to the bus. Whenever a core generates a memory request, the processor in this case needs to wait for a bus grant in order to send the request. Once the router receives the request, it processes it and sends it back on the bus. If the request needs to be sent to a different router, then it is packetized and sent on the network. The memory request process time is measured as the time it takes to get the bus grant, plus the time it takes the router to process the request.

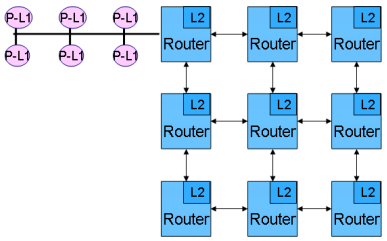

Fig. 2: Hybrid Architecture

Several benchmarks have been simulated on both architectures. The sets of data generated are produced by synthetic benchmarks. The numbers 5, 10, 20, 30, 40, 50 in Figures 3 and 4 represent the percentage of the time cores generate memory requests. We have a random process that generates a number between 0 and 1. If the number is less than the percent value, then the processor core generates a memory requests. The destination of the request, meaning the router to whom the request will reach, is also generated randomly. The delay then is calculated from the time the request is generated,

to the time the data reaches the source again. It includes wait times in the router queues, number of hops, and process time. We considered a unit time for all delays. In the tables and corresponding figures, it is evident that the hybrid model performs better than the NoC model. But as seen from Figure 5, we cannot add as many processor cores as we please to the bus since the bus saturates faster than the network on chip. But for small number of processor cores, having a hybrid model reduces the overall delay. It is also interesting to see how the average wait time increases significantly when the memory requests exceed 20%. The reason for that sudden jump is the saturation of the system.

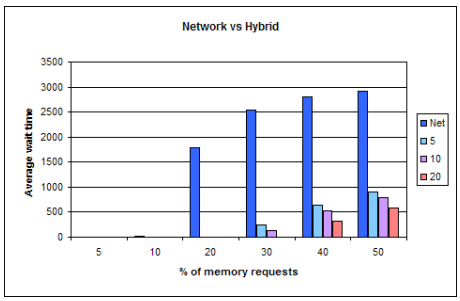

Fig. 3: {Number of Routers, Number of cores on each bus} {4x4,4}

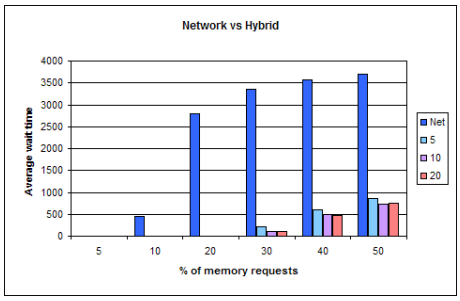

Fig. 4: {Number of Routers, Number of cores on each bus} {5x5,4}

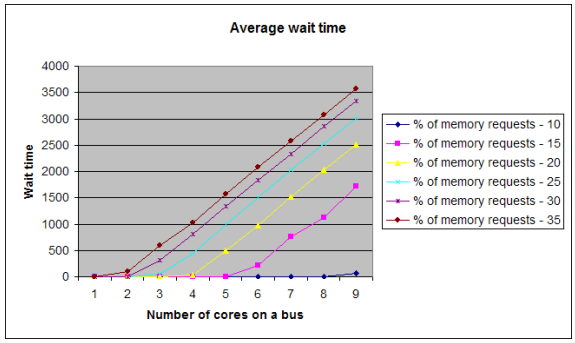

Fig. 5: Average wait time based on number of cores per bus and memory intensity

B. Determining Number of Cores per Bus

Now that we established the need for a hybrid bus-network model, the second step is determining how many cores need to be assigned to each bus. But then, why should we make the number static? If the number of processor cores on a bus fixed, it simplifies the architecture.But making it variable makes the architecture more flexible. As seen in Figure 5 different memory access rates have different needs. Therefore we propose having a reconfigurable interconnect, where the number of cores per bus is variable and assigned by the operating system at run time. The proposed architecture will be introduced in the following section.

C. Adaptability of Architecture to the Next Generation

A third step is studying how well the architecture can adapt to new designs. If design engineers decide to put functional units alongside processor cores on a single chip or have asymmetric processor cores rather then symmetric ones [9], then the architecture should be able to use the functional units without major redesign. In a static architecture, it would be difficult to incorporate other functional units alongside processor cores without redesigning the entire architecture. But in the reconfigurable model, functional units can be connected to processor cores without much overhead. Therefore the need for reconfigurable interconnect becomes a necessity

D. Reliability

An important final step is determining the reliability of the architecture. Having a large number of processor cores with reconfigurable hardware also means core failures can be tolerated gracefully. The architecture should be able to identify non-functional processor cores, and adapt to the existing architecture. Reconfigurability guarantees the continuation of operation without sacrificing performance.
Based on the four points mentioned in this section, we propose Reconfigurable Architecture for Multicore Systems (RAMS), the details of which are explained in the following section.

III. Proposed Architecture - RAMS

A. Architecture Features

From the Motivation section, the need for a reconfigurable hybrid architecture is clear. Based on the aforementioned observations, we propose the architecture shown in Figure 6. We will explain how this architecture covers the needs mentioned in the motivation section.

In the figure, processors are represented by circles, routers are rectangular, and the diamonds are bus switches that have 2 states, on and off. Each processor core has a local L1 cache, and the L2 cache is shared and distributed equally among the routers. As seen, the routers are connected as a mesh. Communication between routers is packet switched. The processor cores are also connected as a mesh. But each segment of the bus is controlled by a switch. Based on the configuration of the switches, buses can be created dynamically. Once the buses are formed, they need to be connected to the routers. The

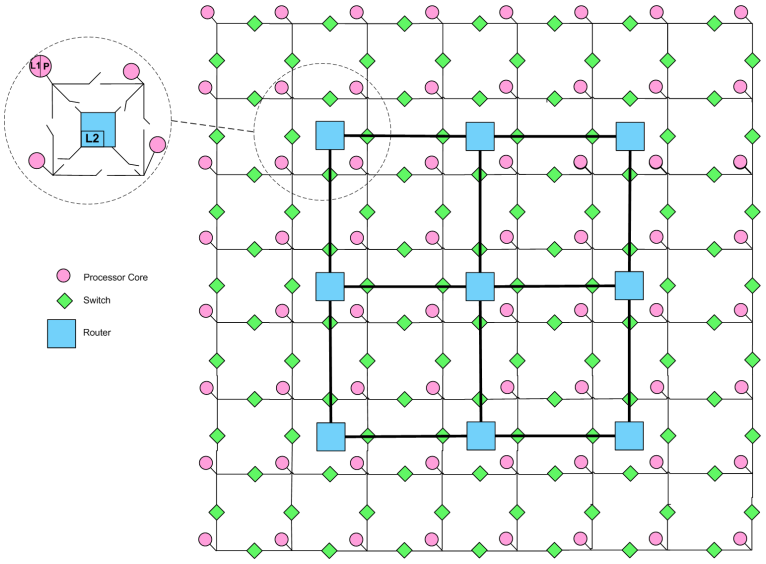

Fig. 6: Hybrid Reconfigurable Architecture

connection point between routers and the buses are shown in the bubble. Each router has 4 access points to the bus. The alternative was to dedicate one processor core to each router as a starting point, but having four allows the bus to expand in any direction. The number of switches needed to implement this architecture is 2p(p-1) + 4R where p is the number of processors on each row and R is the number of Routers in the system. Therefore the architecture provides hybrid bus-network model. It also gives the flexibility of connecting as many processor cores on a bus as needed. If there are different functional units in the design, they can be connected to different processes on demand via reconfiguration. Finally, adapting to core failure is simple. The OS can disregard the dead processor core and remove it from the available cores list.

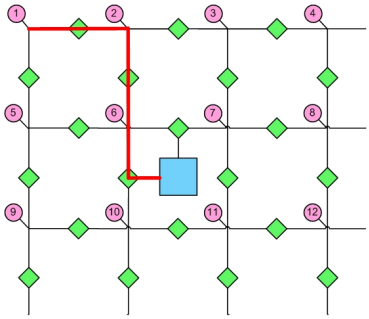

Fig. 7: Cores that cannot be Assigned

B. Configuration

The switches need to be configured so as to form a bus and connect the bus to a router to form a hybrid bus-NoC instance. In order to map cores to routers, several constraints need to be taken into consideration while configuring the system. Listed below are some of those constraints:

- No processor should be connected to more than one router at the same time.
- No core should be stuck isolated and unusable.
- All cores need to be shared equally between routers on a need basis. No router can use all the available cores, starving other cores in the process.
- Any mapping of cores to routers should be possible.

A core can become unusable if the cores around it are assigned to routers and it does not have a free path to a different router. Since the cores that are on the edge of the mesh have a higher probability of becoming stuck, the algorithm should try to assign those cores first. But routers can only get to the boundary cores starting from their local neighbors. An example of this scenario is shown in Figure 8. If cores 2 and 5 are assigned, core 1 cannot be assigned to any other bus until either core 2 or core 5 is released.

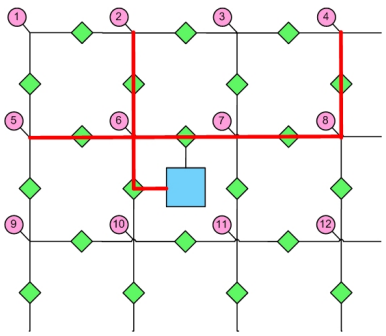

Fig. 8: Cores that cannot be Assigned

In order to meet this constraint, the proposed algorithm assigns a priority to each core. The priority can be based on several factors. In the first case, the priority can be based on the distance of the core relative to each router. The ones close

to the routers would have high priorities, and the ones that are further away would have lower priorities. This insures that the buses are short and the cores assigned to each router are condensed. The disadvantage of this approach is that the cores that are on the extremities will have less likelihood of being assigned therefore becoming unusable. Another approach is to have R priorities for every core, each corresponding to the distance of each router and its position in the Mesh. Meaning, if a core is on the lowest left corner, then its priority relative to the top right router will be low, but relative to the cores in the center of the mesh, the priority would be high. However, since this algorithm will be used at run time, speed of execution and the amount of data stored is very important. Therefore to simplify this proposed method, we used one priority for every core based on its location in the Mesh. The other priority schemes will be explored in the future. The priority values start at p which is equal to the number of cores per row. The cores on the boundary have the highest probability of getting stuck, therefore the priority of those corner cores are set to the highest. Following that, the priority decreases while going inward. The priorities for the cores at the center of the mesh are set to zero as they do not have a high probability of being blocked. Additionally simulations showed that setting these values as all zeroes instead of a linear decrease including the values of three, two and one provided for greater flexibility and granted the inner groups the ability to grow in a balanced way. An example of the priority values for an 8x8 core is shown in Table I. Because of assigned priorities, the routers at the corners of the mesh will have a tendency to connect to the cores at the corners and edges of the mesh, thereby minimizing the risk of cores being unusable. The algorithm is shown in Algorithm 1.

As seen, the algorithm first assigns priorities to each processor core. When a process asks for NP cores, the algorithm lists the routers have at least NP cores available. It then chooses the router that has the least amount of available cores that is greater than NP. The greedy approach is used in this aspect. Once the router is chosen, the algorithm then starts forming the bus. It enqueues the available neighboring cores in terms of priority. The processor core connected to the router is the first core in the queue. It then starts queuing the neighbors of the cores in the priority queue. It continues this process until NP cores are used to form a bus.

In Figure 9, we can see an instance of the mapping algorithm. If we need to map 5 cores to a new bus, then we see that R1 has 6 cores available and R2 has 12 cores available, then the algorithm will choose R1. The smaller one is chosen in case in the next cycle another process needs more than 6 cores. Once R1 is chosen, the bus is formed starting with the nearest cores of highest priority. The result of the mapping algorithm is shown in Figure 10
The numbers at the top of each core represent the router a particular core is assigned to. -1 means that particular core is not assigned.

Input: List of Routers **R**, List of Processors **P**, Number of processors needed **NP**
Output: Router with NP processors available
for $i = 1$ to p **do**
  $Assign\_Priority(i)$
end for
for $i = 1$ to r **do**
  if i has NP cores Available **then**
    Add i to $List\_of\_Routers$
  end if
end for
$Chosen\_Router = Minimum(List\_of\_Routers)$
$List\_of\_Available\_Cores =$
$Neighbors\_of\_Chosen\_Router$
$Sort\_By\_Priority(List\_of\_Available\_Cores)$
for $i = 1$ to $List\_of\_Available\_Cores$ **do**
  $NP = NP - 1$
  if $NP == 0$ **then**
    break
  end if
  for all $neighbor$ of i **do**
    if $neighbor$ is Available **then**
      $Append(List\_of\_Available\_Cores)$
    end if
    $Sort\_By\_Priority(List\_of\_Available\_Cores)$
  end for
end for

Algorithm 1: Mapping Algorithm

TABLE I: Priority of Cores

8	7	6	5	5	6	7	8
7	6	5	4	4	5	6	7
6	5	0	0	0	0	5	6
5	4	0	0	0	0	4	5
5	4	0	0	0	0	4	5
6	5	0	0	0	0	5	6
7	6	5	4	4	5	6	7
8	7	6	5	5	6	7	8

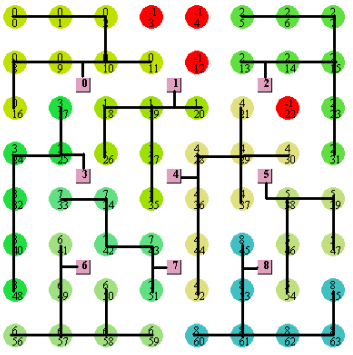

Fig. 10: Example of Mapped Cores

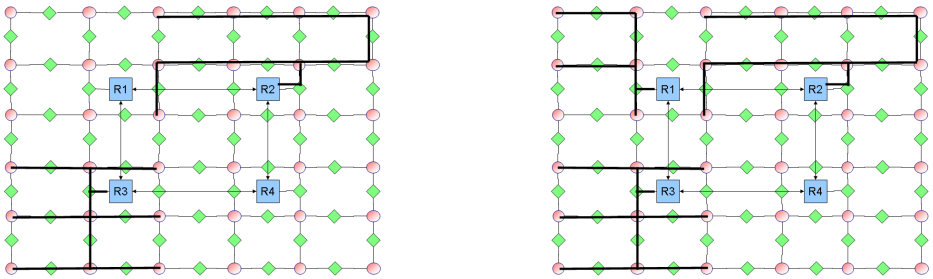

Fig. 9: Before and After New Assignment

Of course there is always room for improvement in order to ensure better mapping. However, this algorithm guarantees an assignment if a path exists since it checks the number of available cores for every router. Furthermore, if a hardware failure should occur, then the algorithm can assume the failed processor core is always assigned, and continues operation without major disruption. The algorithm was implemented in Java. The run time is less than 1 second for architectures with over a hundred cores.

IV. Experimental Results

In the experimental analysis, we show two sets of data. The first one is to show the benefits of the hybrid model using the Network Simulator (NS2) tool [10] . The second set shows the run time improvement of the reconfigurable architecture using SIMICS tool [11] [12].

A. Benefits of the Hybrid Model

In RAMS, since processor cores that communicate regularly are on the same bus, the requests sent on the network are fewer when compared to the NoC implementation since in the NoC implementation, all communication between cores is done using the network.
To compare the two models, we simulated two architectures. The first is a 9x9 NoC implementation and the second is a 3x3 hybrid bus-network implementation with 8 cores per bus. In the NoC implementation requests are generated every 1 microsecond, while in the hybrid implantation, 2 scenarios are shown, one when requests are generated by the nodes every 4us and the other every 8us. The result computed is the average wait time it take to process a request and is dependent on the number of hops of the mesh. The bandwidth of the links is 10MB with a propagation delay of 10us.

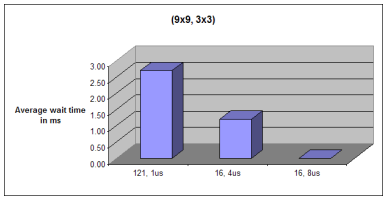

Fig. 11: Comparison of 3x3 and 9x9 NoCs

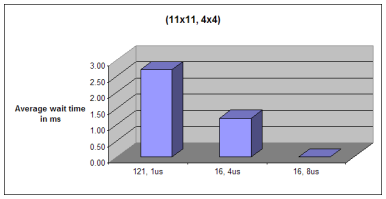

Fig. 12: Comparison of 4x4 and 11x11 NoCs

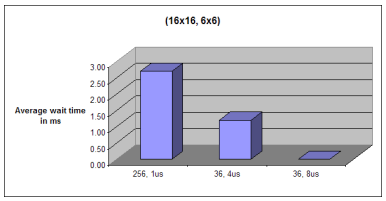

Fig. 13: Comparison of 6x6 and 16x16 NoCs

Both cases of the hybrid implementation, process requests were received faster when compared to pure NoC. The delay was computed as the number of hops needed to get from source to destination. Queuing delay is included in the results. This result was expected based on the C++ simulation results shown in the Motivation section. Since the average distance of a 2D Mesh is

$$\frac{2}{3} * N^{\frac{1}{2}} \tag{1}$$

reducing N reduces the average distance significantly, hence the major improvement in the hybrid model.

B. Benefits of Reconfigurability

To verify the merits of reconfigurability, we simulated the architecture in Simics. The simulation variables used for the experimental results are shown in Table II. We used the benchmark suite PARSEC since it was developed solely for multicore architectures [13] [14].

The number of hops is evaluated based upon the time it takes to access a physical cache addresses tied to an individual router. In the non-reconfigurable model, processor cores were attached to routers in an even allotment. Meanwhile in the RAMS model, cores were attached to routers using the reconfigurable approach. The results generated from the x264, streamcluster and fluidanimate PARSEC benchmarks with a 3 x 3 L2 Mesh and 64 cores are in Figures 17, 18, and

TABLE II: Experimental setup

Parameter	value used
Number of cores	16,32,64,128
L2 Size in KB	3072*number of processors
L2 Line Size	128B
L2 # of Cache Lines	L2 Size * 1024 L2 Line Size
L2 Associativity	8-way
L2 Replacement Policy	LRU
L1 Size in KB	64
L1 Line Size	128B
L1 # of Cache Lines	L1 Size * 1024 L1 Line Size
L1 Associativity	2-way
L1 Replacement Policy	LRU
Benchmark	PARSEC

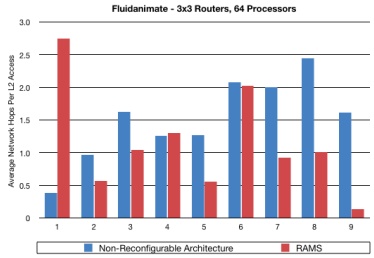

Fig. 14: Average Number of Hops to Access Data on each L2 Slice

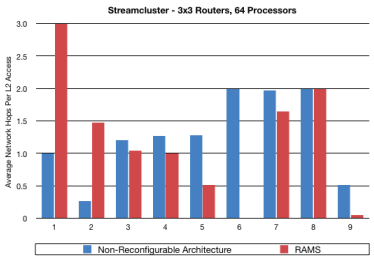

Fig. 15: Average Number of Hops to Access Data on each L2 Slice

19. Figures 20 and 24 show the average number of network hops per L2 access for all the L2 slices on 64 processors and 128 processor architectures respectively. These plots show the average one-way networks hops consumed in accessing each of the 9 L2 slices. After comparing the Non-Reconfigurable and RAMS approaches, there is a clear trend showing that overall network activity on a per router basis is greatly reduced in RAMS. This is further evidenced in Figure 20 where each benchmark showed a minimum average improvement of at least 45% hops per cache access. Figures 21, 22 and 23 shows a nearly identical experimental setup, however this simulation involved 128 total cores. Again the Average Network Hops per L2 access are greatly reduced when using the RAMS approach. On every benchmark and for nearly every L2 slice, the overall network activity per access is significantly reduced. All three benchmarks saw the Average Network Hops per L2 Access reduced by a minimum of 38% hops per access This significant increase is evidence of the usefulness and need for

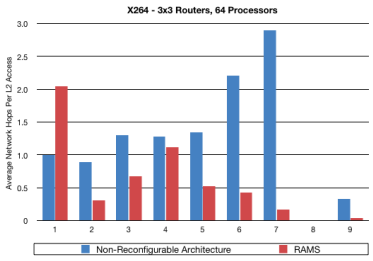

Fig. 16: Average Number of Hops to Access Data on each L2 Slice

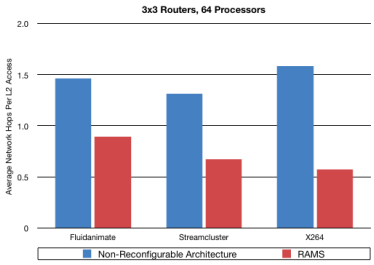

Fig. 17: Average Number of Network Hops for 64 Cores

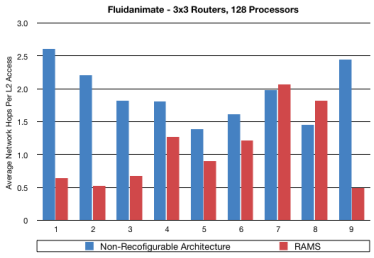

Fig. 18: Average Number of Hops to Access Data on each L2 Slice

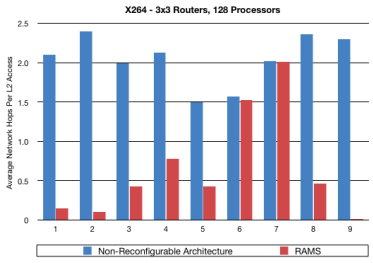

Fig. 19: Average Number of Hops to Access Data on each L2 Slice

a reconfigurable approach such as RAMS.

V. Conclusion and Future Work

Hybrid bus-based and NoC architectures are essential to gain the performance benefits of both. In such hybrids, the superiority of bus-based architectures for small number of processors can be combined with the scalability of NoC architectures for large number of processors. Further, configurability introduces additional flexibility in allocating optimal

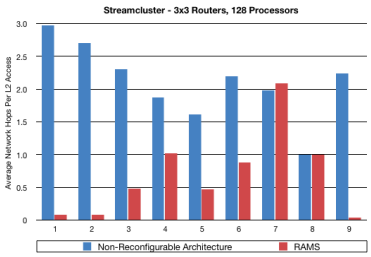

Fig. 20: Average Number of Hops to Access Data on each L2 Slice

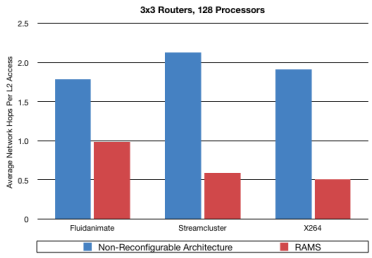

Fig. 21: Average Number of Network Hops for 128 Cores

bus sizes for each individual application, maximizing processor utilization and enhancing reliability. In this paper, we have presented a new hybrid, configurable architecture named RAMS. Preliminary studies show that the proposed RAMS architecture is superior to a fixed bus or a fixed topology NoC architecture. Further study is needed to fully understand the cost of reconfigurability. Thermal dissipation is another issue that has yet to be addressed. This is critical, as dense areas of active processor cores will result in tightly clustered areas of higher temperatures [15] [16]. Preventing heat buildup can be done by considering thermal dissipation as a variable in the smart scheduling and mapping done for processes. If one part of the chip has a temperature beyond a specified threshold, subsequent tasks can be assigned to distant cores allowing for the heat to disperse. Inclusion of this feature would improve the overall device consumption and additionally extend the lifetime of the chip. Thread migration is another potential improvement that has not been considered [17]. Thread migration is the act of moving threads from one processor core to another. This feature would allow process mapping to be dynamic and more threads to run simultaneously on the chip. Several experiments have been performed in cache migration and each achieved varying degrees of success [18] [19]. Finally, various cache optimizations can be used to find further enhancements to the RAMS architecture. Operating system level procedures to further target placement in local caches proved useful in reducing both network traffic and cache contention in [20] and [21]. All of these experiments could be implemented in variations on the RAMS architecture to provide additional enhancements.

REFERENCES

[1] "Moore's law." [Online]. Available: http://www.intel.com/technology/mooreslaw/

[2] H. J., B. J., and K. S., "From a few cores to many: A tera-scale computing research overview," 2006. [Online]. Available: http://download.intel.com/research/platform/terascale/terascale_overview_paper.pdf

[3] "Intel research advances 'era of tera'," 2007. [Online]. Available: www.intel.com/pressroom/archive/releases/20070204comp.htm

[4] D. Wingard, "Addressing multicore communication challenges using noc technology," in *Computer Design, 2006. ICCD 2006. International Conference on*, Oct. 2006, pp. 188–188.

[5] S. Bourduas and Z. Zilic, "A hybrid ring/mesh interconnect for network-on-chip using hierarchical rings for global routing," in *Networks-on-Chip, 2007. NOCS 2007. First International Symposium on*, May 2007, pp. 195–204.

[6] E. Bolotin, Z. Guz, I. Cidon, R. Ginosar, and A. Kolodny, "The power of priority: Noc based distributed cache coherency," May 2007, pp. 117–126.

[7] R. Knauerhase, P. Brett, B. Hohlt, T. Li, and S. Hahn, "Using os observations to improve performance in multicore systems," *IEEE Micro*, vol. 28, no. 3, pp. 54–66, 2008.

[8] R. Das, S. Eachempati, A. Mishra, V. Narayanan, and C. Das, "Design and evaluation of a hierarchical on-chip interconnect for next-generation cmps," Feb. 2009, pp. 175–186.

[9] M. D. Hill and M. R. Marty, "Amdahl's law in the multicore era," *Computer*, vol. 41, no. 7, pp. 33–38, 2008.

[10] "http://nsnam.isi.edu/nsnam/index.php/main_page."

[11] "http://www.virtutech.com."

[12] P. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," *Computer*, vol. 35, no. 2, pp. 50–58, Feb 2002.

[13] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: characterization and architectural implications," in *PACT '08: Proceedings of the 17th international conference on Parallel architectures and compilation techniques*. New York, NY, USA: ACM, 2008, pp. 72–81.

[14] C. Bienia, S. Kumar, and K. Li, "Parsec vs. splash-2: A quantitative comparison of two multithreaded benchmark suites on chip-multiprocessors," in *Workload Characterization, 2008. IISWC 2008. IEEE International Symposium on*, 2008, pp. 47–56. [Online]. Available: http://dx.doi.org/10.1109/IISWC.2008.4636090

[15] M. Monchiero, R. Canal, and A. Gonzalez, "Power/performance/thermal design-space exploration for multicore architectures," *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 5, pp. 666–681, 2008.

[16] E. Kursun and C.-Y. Cher, "Temperature variation characterization and thermal management of multicore architectures," *Micro, IEEE*, vol. 29, no. 1, pp. 116–126, Jan.-Feb. 2009.

[17] K. Shaw and W. Dally, "Migration in single chip multiprocessors," *Computer Architecture Letters*, vol. 1, no. 1, pp. 12–12, January-December 2002.

[18] M. Chaudhuri, "Pagenuca: Selected policies for page-grain locality management in large shared chip-multiprocessor caches," in *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on*, Feb. 2009, pp. 227–238.

[19] N. Eisley, L.-S. Peh, and L. Shang, "Leveraging on-chip networks for data cache migration in chip multiprocessors," in *PACT '08: Proceedings of the 17th international conference on Parallel architectures and compilation techniques*. New York, NY, USA: ACM, 2008, pp. 197–207.

[20] S. Cho and L. Jin, "Managing distributed, shared l2 caches through os-level page allocation," in *MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 455–468.

[21] M. Awasthi, K. Sudan, R. Balasubramonian, and J. Carter, "Dynamic hardware-assisted software-controlled page placement to manage capacity allocation and sharing within large caches," in *IEEE INTERNATIONAL SYMPOSIUM ON HIGH PERFORMANCE COMPUTER ARCHITECTURE*, 2009, pp. 250–261.