Software Profiling to Improve Network on Chip Performance

David Anthony, Charles Lucas, and John Tooker
Department of Computer Science and Engineering
University of Nebraska-Lincoln
Lincoln, NE 68588
{danthony,clucas,jtooker}@cse.unl.edu

ABSTRACT

Network on chip (NoC) processors utilize network connections in place of traditional bus or cross bar interconnects in order to improve performance, power consumption, and scalability in modern multicore processors. However, superimposing a network onto a chip introduces additional factors, such as network latency, routing, and buffer depths all begin to affect the behavior of the system. Communication between processing elements becomes the bottleneck. Current scheduling and orchestration solutions address these issues, but leave room for improvement. In this paper, application profiling is proposed to increase the performance of a NoC's scheduler; a custom simulator is created to validate this advanced scheduler.

1. INTRODUCTION

The growing number of processing elements in today's multicore systems is putting an increased strain on the interconnect system used by the elements to communicate with each other. To address the needs of the system, scalable, high-performance, energy efficient methods are needed for inter-core communication. Network on chip (NoC) systems are a modern attempt to fulfill these needs.

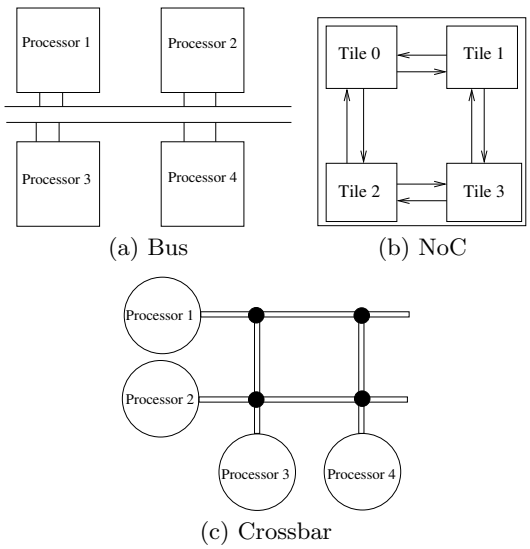
In this paper, the state-of-the-art NoC scheduler is improved upon by using software profiling to predict the future usage of system resources from past behavior, thereby enabling more efficient usage of processor resources without additional software modifications. In order to test this hypothesis, a custom simulator was built to model the system's behavior and generate performance metrics. The novelty of this approach lies in the preemptive reservation of system resources by using application profiling. The spatial locality in a network is exploited by placing related processing elements as closely to each other as possible in order to minimize network traffic and path length.

A plethora of research [3, 5, 13, 17, 20] has been performed on improved network scheduling and planning, but [8] says NoC improvements are not enough. Application specific information must be included in the routing decisions. This project attempts to address this need by using process orchestration to improve the performance of a NoC.

The remainder of the paper is structured as follows. In Section 2 we present our motivating case and related work. Our methods and implementation are discussed in Section 3. Section 4 gives the details of our results. Finally, Section 5 gives our conclusions and future work.

2. MOTIVATION AND BACKGROUND

NoCs are a state-of-the-art attempt to connect many processing elements together on a chip. They offer several advantages over older connection techniques, such as a crossbar switch or bus based interconnect. For illustration purposes, a simple 2x2 NoC is shown in figure 1(b). The network on chip is broken up into smaller computational blocks, or *processing elements*, that each contain a simple network interface. Using these network interfaces, the processing elements are able to communicate with each other. Compared to the bus architecture shown in 1(a), NoCs exhibit a higher degree of concurrency because processing elements on different areas of the chip do not always utilize the same network links, and can thus transmit in parallel with other tiles. NoCs have



(a) Bus (b) NoC

(c) Crossbar

Figure 1: Interconnection Methods

several advantages over the crossbar switch shown in 1(c). They are more efficient from a hardware viewpoint because not as many connections are needed between the different endpoints. Also, the individual links are shorter, so adjacent or nearby processing elements can communicate with each other with extremely low latency. The combination of these benefits results in an extremely fast interconnect[7].

2.1 NoC Issues

Using a network-on-chip is not without its drawbacks. Moving to such a system results in several of the drawbacks seen in more traditional computer networking studies. For instance, the network introduces a non-deterministic latency into inter-node communication, because exchanged data will pass through a varying number of intermediate hops, depending on the route that is taken. This also has the interesting side effect of making the power consumption of the processor vary because the power consumed by transmitting information is proportional to the number of hops a message must pass through[7]. Nodes can also become overloaded by network traffic, which can lead to dropped packets.

One possible approach to alleviating these problems is to group the nodes that need to communicate with each other as closely as possible[3, 5]. This limits the number of hops that a message needs to take between its source and destination. This limits the overall traffic load seen by the entire system. However, deciding how to allocate system resources, while maintaining a high level of system utilization can be difficult[20, 14].

2.2 Profiling

There are many research groups focusing on profiling specific applications or specific hardware. Some focus on virtualization, others on simple algorithms and [5] uses two tractable, sub-optimal algorithms to place processes based on their application's profiles.

Virtualization of applications is the focus of [25]. Their focus is on applying a middle, virtual machine (VM) layer between applications and the hardware they run on. While this virtual layer offers some overhead, profiling applications yields an improvement in performance by being able to dynamically allocate and (from the application's point of view) create resources.

The adaptive virtualization framework monitors three resource areas: CPU, network and disk access. The CPU area monitors the user space, kernel space and IO wait time percentages; while RX/TX packets per second and RX/TX bytes per second are monitored for the network classification; and the disk measures R/W requests rates and R/W block rates. Each application (running on its own VM) is rated on the percentage of each category it uses; based on these percentages, an application is classified as computationally intensive, disk intensive or network intensive.

The types of programs [24] focuses on are small, network routing programs. Though they are small, they are all unique and diverse. [24] focuses on automating this mapping process well by collecting more data than can be collected on more general purpose (larger) applications.

A robust linear regression algorithm is used in [25] as is stepwise regression to focus on the most significant and important statistics. Although there exist outliers which cannot be accurately modeled without significant overhead, more improvement overall is gained by focusing on the more common trends than on improving the outlier's performance. Similarly, the model is reviewed for 'bursts of outliers', which would otherwise skew the model's accuracy.

While [25] focuses on profiling a diverse set of applications for a specific implementation, [24] profiles a specific set of programs (algorithms) to run on a diverse set of architectures. It considers an application from a uniprocessor point of view to figure out how to map it to a multiprocessor by using an, "annotated directed acyclic graph (ADAG) to represent the dynamic execution profile of applications." [24]

Once a program is captured, a best mapping must be performed. This is an NP-complete problem, so random (local) search is used:

> The system performance model is a key component for the randomized mapping algorithm. The model considers processing, inter-processor communication, contention on memory interfaces, and pipeline synchronization effects to determine the overall throughput [24].

ADAG is set up so that, "nodes represent processing tasks and links represent control and data dependencies." [24]

Random search is not the only algorithm used; [5] attempts to place processes based on an application's *shape*. The shape of an application is assessed by the application's current profile (a snapshot). This reactive approach (based on events) yields a performance increase (and energy savings) of around 40%. The techniques in [5] ignore the case when not enough resources are available (a request is simply denied).

The state-of-the-art orchestration schemes proposed in [24] and [5] all use profiling (either history based or instance assessing), to address the communication issues that NoCs incur, but as communication is the bottleneck of such a system, more improvement is needed. Since these technique are reactive, it is expected that a predictive algorithm will perform better, though it will have a larger overhead.

2.3 Routing

In order to motivate the need to place elements together different routing protocols are presented. By examining how the routing protocols operate, the need to exploit spatial locality in placing processing elements emerges.

2.3.1 Wormhole Routing

In traditional packet-switched networks a routing method called store and forward is used. In this method an individual packet is buffered at each intermediate hop it takes between its source and destination. This means that at each hop the entire packet must be received before it is transmitted to the next hop. This large buffer size can introduce unnecessary latency and degrade performance in NoC systems[7, 10].

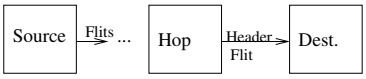


Figure 2: 1-Dimensional Wormhole Routing

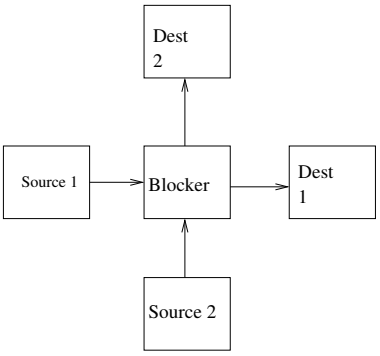


Figure 3: Wormhole Contention

Wormhole routing provides an alternative to traditional store and forward methods that improves message latency. In wormhole routing a message is broken up into small pieces called *flits*. These flits are then transmitted from a source to a destination. At each intermediate hop on the path a flit is sent to the next hop as soon as the preceding flit has cleared the node. This avoids the need to store the entire message at each intermediate hop [11, 10]. In addition to improving latency, wormhole routing also has the advantage of reducing message buffer sizes because the intermediate nodes do not need to store the entire message at one time because they store the smaller flits. Figure 2 demonstrates wormhole routing along a 1-dimensional path.

Wormhole routing does have several drawbacks. First, unlike the closely related *virtual cut-through* routing method, nodes can only hold one flit at a time. This means that the message actually occupies space at every intermediate hop along its route while it is being transmitted[23, 10]. Occupying this space requires the routing algorithm to be deadlock free, otherwise the system may be stuck in a loop where no packets can be received[19]. Another drawback is wormhole routing protocols are typically very simplistic, which means they are prone to congestion, which reduces network efficiency[16]. Using more advanced routing algorithms can lead to better performance, but may introduce non-deterministic latency, use more system resources, and increase the energy consumption of the chip[14, 5, 7]. Figure 3 shows two nodes blocking each other in a wormhole network.

Clearly, shortening paths in wormhole routed networks can reduce contention. This is further explored in [4]. In this paper the effects of different grouping schemes was examined in wormhole routed networks. They showed that spatial locality has a large impact on the performance of wormhole networks, and networks with well located members can significantly outperform networks with poorly located members.

Thus there is a direct benefit for properly placing interdependent processes on physically close processing elements in a close to optimum way.

2.3.2 Adaptive Routing

Adaptive routing may also be performed, as presented in [8]. Here applications are profiled to assess their priorities: latency and throughput requirements. Routing decisions are based on these profiles. Application-level system throughput was increased based on the proposed protocol. This work focused entirely on improving routing protocols, and avoided the question of network topology on network performance.

2.4 System on Chip Models

Accurate simulation tools must be used (or created) to model real-world system performances. Generalizations ease the implementation of models; [Ker10] uses memory latency of the (shared) L2 cache of 10 cycles and main memory latency of 200 cycles. Setup times can be ignored (not included as part of the calculations) when focusing on raw throughput, but not necessarily when considering multiple, dynamic applications.

[15] observes *windows* of process execution, to make the decision of the next window based on the previous one (or previous few). [Bec09] monitors processes for at least 20 seconds, then creates (or updates) their profiles, followed by a custom execution based on that profile. Each window is part of a state, each state holds the amount of resources consumed during that time; amounts of: CPU, Memory, Disk bandwidth, and network bandwidth. Averages and standard deviations are computed. Depending on those calculated values (during a window), that window may be appended to the last state or a new state may be initiated. Multiple *common* states can be discovered for each application.

[24] and our project focus on mapping processes to processors. The system must assign processing task (ADAG nodes) to processing elements. This must be done in the spacial and temporal domain. A schedule must be created and should be started at the top of the ADAG.

[18] proposes a synthetic application simulator, rather than running a specific benchmark suite. Their work focuses specifically on mesh-based applications, and on load balancing by means of repartitioning datasets. A specific benchmark suite is not always the best choice when it comes to simulating a NoC. For these reasons, a custom simulator with synthetic applications will be used to compare the ideas proposed in this paper with the state-of-the-art scheduling schemes.

The goal of [22] is to model performance in a way that promises to be faster than cycle-accurate simulations and more precise than peak-performance metrics. They do this by abstracting a model of an application from observing its execution. This non-precise model is then used to evaluate the performance of several systems. Their simulation runs four orders of magnitude faster than cycle-accurate simulations but they report error in performance estimation of only up to 20%.

2.5 Topology Management and Network Management

Several prior attempts have been made to optimally allocate resources in a NoC. These approaches aim to minimize energy consumption of the system, minimize chip resources, reduce network latency, and improve application performance.

In [13], the authors attempt to reduce the size of the buffers needed for an application by analyzing the network traffic characteristics of an application. These characteristics are then used to profile the buffers needed for message passing, such that the buffer depths are minimized. The ramifications of this work is an assurance that enough buffer space is provided so that message can always be sent. This reduces latency, because nodes are not blocked from sending messages. Second, by minimizing the buffer depths, a minimum number of system resources are needed for them, which reduces the complexity of the system and improves its energy consumption.

[5] take an extremely similar to the approach we are proposing. In this paper, the user behavior is monitored in order to allocate resources in a NoC. Depending on the user behavior, one of two possible resource allocation strategies is employed that attempt to exploit contention behaviors in order to minimize energy. Our approach differs in that it does not depend on the user behavior, only the application behavior. Also, it should incur a smaller overhead because the system does not need to monitor the user behavior.

Both [3] and [12] consider the impact of allocation schemes on fragmentation. Fragmentation can be considered in two contexts. The first is *internal* fragmentation. Internal fragmentation occurs when the resources allocated for a process exceed what is needed for the process. This causes resources to be idle and underutilized. The second type is *external* fragmentation. In external fragmentation, separately allocated areas prevent other processes from claiming a large enough block, and thus prevent them from being run.

3. IMPLEMENTATION

In order to test the hypothesis – that a proactive, reservation based scheduler would yield an improved orchestration protocol – a custom simulator was constructed to simulate several real-world program traces, similar to what was done in [18]. Initially, an off the shelf simulator such as OMNet++, ns-2, or Noxim [2, 9, 21] was going to be used, but this approach did not prove to be feasible because of the steep learning curve involved with these tools and the time constraints of the class[6]. These tools did not provide the needed functionality out of the box, thus a unique simulator was written, rather than attempt to learn a complex codebase and modify it for this project.

3.1 Simuator

A custom, cycle accurate simulator was written in C++. The simulator implemented a 2D mesh topology. The specific routing protocol was not simulated because statistics about the locality of the processing elements was desired, not the actual details on how the messages were routed. A single processing element (including the scheduler) has no global knowledge and receives all of its knowledge via messages.

3.1.1 Network

The simulator consists of a network, which holds processing elements (PEs). The network takes care of routing messages to separate PEs and notifying each PE that another cycle should be simulated. Actual routing is not implemented; a message is either transmitted or not. A PE can only transfer one message per cycle. The topology of the network is a 2D mesh of size S, where S can be specified at compile time. There is also a special PE called the Master PE (MPE) which acts as the scheduler.

3.1.2 Processing Element

Each processing element (PE) can run a process and handle its network traffic. To start a process running on a PE, a message is sent to it with the process's information. When a process needs to spawn threads (children), it sends a message to the MPE. Processes can also talk amongst themselves inside their families (groups of processes sharing information and sharing ancestors) by sending messages directly to each other. Messages sent can either be blocking or non-blocking.

Local memory is not explicitly handled by the PE as it is assumed that the memory latency would be much less that time it would take to remove a process and run another one. IO (disk access), however, is simulated. A certain percentage of cycles generates a IO request, which is a message sent to the MPE requesting this information. If memory access was dependent on a processor's location in the mesh, it would be considered in the profile, but this is not what was simulated.

3.1.3 Master Processing Element

The Master Processing Element (MPE) is similar to an operating system, at leas the scheduling part of it and is at the heart of this proposed scheduling scheme. When processes need to be created, a request is sent (via message) to the MPE, and when a process is done, it tell the MPE (which then sends a message to its parent telling them this). It keep track of application profiles and uses this information to reserve PEs and pick the best place to assign every requested process. When choosing where to place a new process, the MPE places it nearest its application's *center of gravity*. The center of gravity is defined as the shortest Manhattan distance to all other PEs that host other processes in the new processes *family*. A family of processes is defined as processes which belong to the same application.

The MPE holds the scheduling information and can locate information regarding a process's family on request. Initially, the MPE would alert all processes in a family every time one of its members was spawned or died, but this created more messages than could be sent.

3.1.4 Processes

Processes themselves run on processing elements and could be tailored in a number of ways. The most detail was put into when and how often a process would request a child process. This was done by creating a custom function for each application which returned whether or not to create a new process given which cycle a processes was on and how

many processes have been spawned already. A base example would be to randomly create a process ever 30000 cycles until a limit was reached. Processes can also make IO requests with certain probability per cycle and have a fixed amount of instructions to simulate. A process completes

3.2 Profiles

Application profiles are kept by the scheduler (MPE) and are used to make good decisions about where to schedule new processes. The novel approach to this project's scheduling is the concept of reserving process elements in anticipation of future spawn requests.

As requests for spawns come from applications and deaths are reported, an applications profile is updated. A profile can indicate two states in which a running application can be: *valley* or *peak*, where the valley state indicates that an application needs only a few PEs and a peak state indicates a process needs many PEs. The (application specific) number of PEs needed in peaks and valleys is kept. When an application's spawn requests reach a certain threshold, a peak number of PEs will be reserved for the anticipated spawn requests that will come soon. When the death count reaches a threshold, the reserved PEs we be revealed. Only one application may reserve PEs at a time.

While PEs may be reserved, this only places a priority on a certain application's processes. When a spawn request is made, the MPE firsts checks to see if there are reserved PEs on which it should run. If there are no reserved PEs, it checks for empty PEs (those without processes) and assigns it to one of them. If that fails, a process may be assigned to a PE that is reserved for another application. Animation 4 shows this.

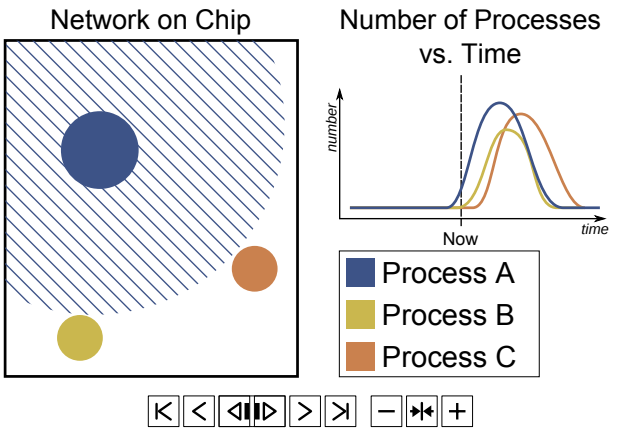


Figure 4: (Animation) Three applications dynamically create processes, reserved space for an application is indicated by crosshatches. Processes of a non-reserved application will be placed outside of the reserved space first.

3.3 Application Models

The lack of publicly available program traces was a major hindrance to this project. The closest resource to what we needed was [1], but this lacked needed trace types. In

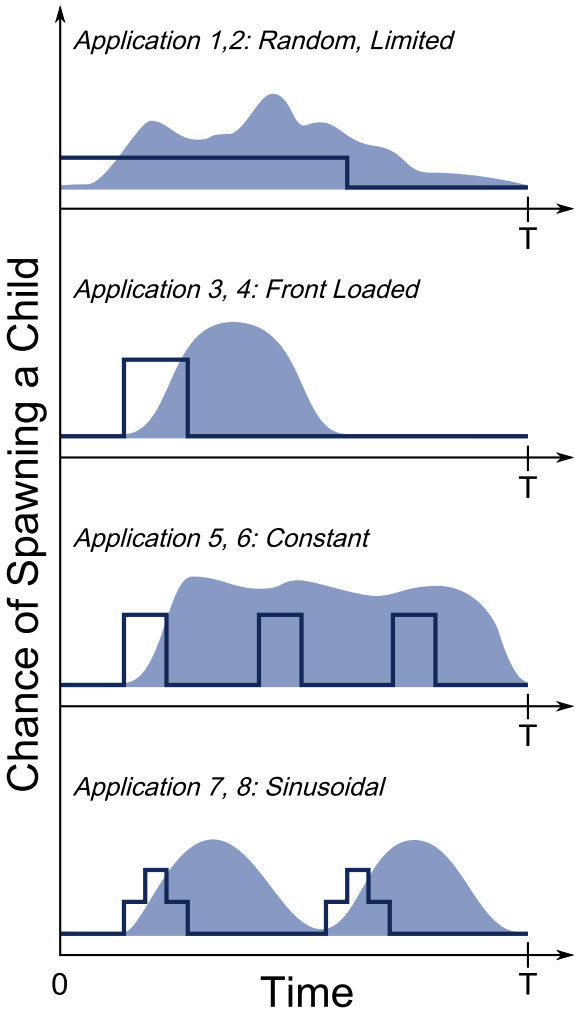


Figure 5: Spawn models used in the eight applications in terms of the application's period. The dark lines indicate the chance of spawning a process at a given time while the shading shows the total amount of processes per application

order to evaluate the effectiveness of the reservation scheduler, synthetic spawn models were generated that exhibited a cyclical processor use pattern, as demonstrated in 5. These models could have been produced from an algorithm such as the ocean simulator where the code is not perfectly parallelized during the entire execution time. This leads to intervals where all or most of the processors (processing elements) would be in use to run the application, and then periods where the code is either not as parallelized, or the processes are synchronizing, in which case a large portion of the processors may be idle.

While the spawn models of each application is different, many other factors were kept constant. Each PE would have the same access delay time to memory and as memory was not the focus of this project, each process would access memory with the same frequency; this amount was factored into

the process's specified lifetime, which was also the same for each application (in terms of cycles). IO request were simulated, and the rates of these requests were also constant across all applications. The processes belonging to an application would communicate amongst themselves and not with other applications. The rate of network messages per process was constant, but the rate of network messages of an application thus depended on how many children we in existence at any one time.

4. RESULTS

To test the performance of orchestrating processes based on our profiling technique, experiments were comparing this paper's approach (using minimal distance and reservation) to a two other approaches (base case and using the minimal distance to schedule). The baseline places processes on the first available node that is free. The state-of-the-art approach places processes greedily in order to minimize the resulting Manhattan distance. Unlike this approach, though, it does not perform any profiling or prediction.

Eight varied application profiles, seen in Figure 5. An attempt normalize each of these applications such that they will each average a maximum of eleven total processes spawned. All eight of these applications were run in every simulation to create a diverse and contentious environment. These simulation were ran around ten times under each of the process placement styles described above. Table 1 shows the results of these trials and Figure 6 summarizes these results.

4.1 Analysis

It can be see that the performance of each method, based on the average Manhattan distance, is as anticipated. The placement of processes in the next available PE performs the worst of the three methods. The technique proposed in this project, based on reservation and profiling, performs better than the state-of-the-art, which uses only the greedy placement algorithm. In most cases, the best performance of the worst scheduling scheme *next available* is still worse than the *reservation* protocol.

Applications one and two randomly spawned processes up to a certain limit. It was predicted that these applications would respond to profiling worse, as there is not much pattern to their behavior, but in fact there is a pattern. The maximum number of process may not be predictable, but the application does stop producing processes after an overall limit is hit, so some PEs can be reserved giving the *reservation* scheme the upper hand. It should be noted that the duration of spawning period for these applications may be longer that others, yielding different patterns of behavior that the others.

While applications one and two are largely random, applications three and four are not as random. During the start of their cycle, they will produce child processes with a high probability, but with an expected value of eleven total processes (though this number may be greater or less than eleven). Because of the short spawning time, PEs may not be reserved in time for the spawns to have a place to go, leading to an average distance performance comparable to that of the minimal distance only scheduling, as seen in Figure 6's applications 3 and 4 columns.

Applications 1 & 2

	Next Available	Minimal Distance	Reservation
Average	0.605 1.004	0.583 0.803	0.459 0.634
Median	0.600 1.015	0.599 0.813	0.463 0.640
Min	0.544 0.870	0.506 0.693	0.388 0.533
Max	0.657 1.147	0.640 0.892	0.547 0.744
Std Dev	0.033 0.095	0.043 0.076	0.037 0.067

Applications 3 & 4

	Next Available	Minimal Distance	Reservation
Average	0.651 0.964	0.515 0.691	0.481 0.627
Median	0.676 0.964	0.519 0.685	0.496 0.627
Min	0.569 0.853	0.382 0.611	0.358 0.537
Max	0.711 1.103	0.641 0.789	0.655 0.703
Std Dev	0.047 0.082	0.086 0.056	0.080 0.046

Applications 5 & 6

	Next Available	Minimal Distance	Reservation
Average	0.960 0.828	0.737 0.733	0.576 0.522
Median	0.970 0.870	0.755 0.740	0.574 0.512
Min	0.874 0.613	0.661 0.459	0.478 0.334
Max	1.042 0.978	0.790 0.968	0.686 0.760
Std Dev	0.056 0.118	0.049 0.165	0.058 0.151

Applications 7 & 8

	Next Available	Minimal Distance	Reservation
Average	0.977 0.818	0.668 0.730	0.510 0.579
Median	0.948 0.811	0.673 0.760	0.499 0.547
Min	0.891 0.672	0.464 0.535	0.449 0.478
Max	1.059 0.956	0.784 0.888	0.704 0.776
Std Dev	0.071 0.094	0.093 0.123	0.068 0.098

Table 1: Statistical values for each simulation scheme's average Manhattan distance between PEs by application. All values are distances.

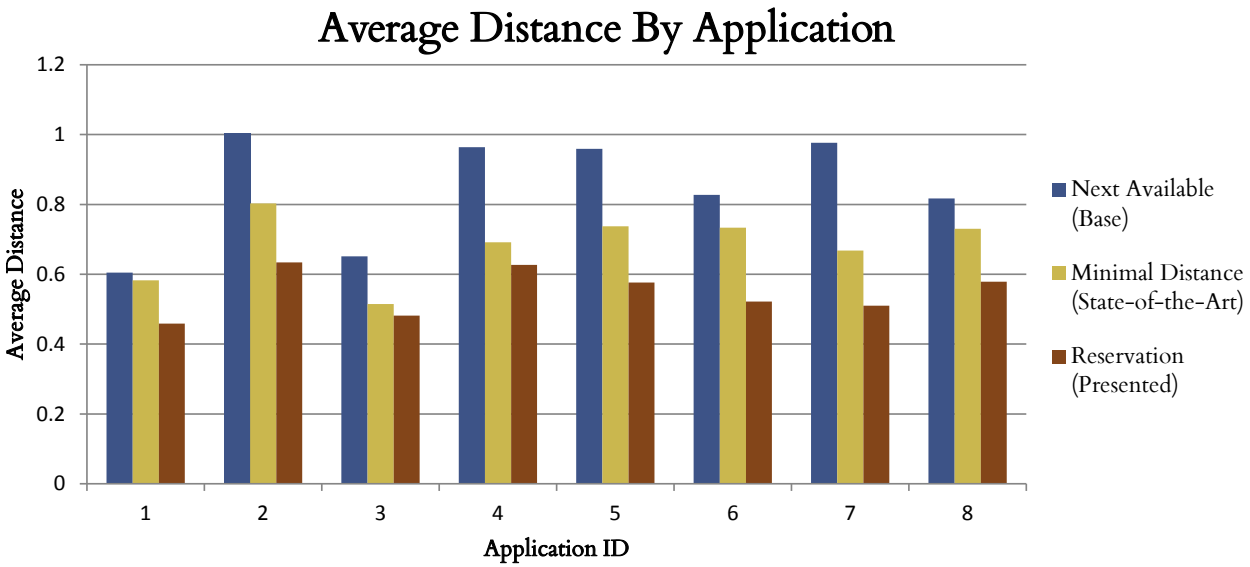


Figure 6: The average Manhattan distance between processes of an application, by application.

Applications five and six attempted to maintain a maximum amount of processes (again, around eleven). This predictable and delay tolerant behavior allows for PEs to be allocated as needed. After one process is done, it will be likely that another process can utilize the vacant, reserved PE unlike applications three and four, which generally will not spawn processes after children have finished running.

Lastly, applications seven and eight produce processes in a cyclical manner, at a higher frequency than the other applications. This regular behavior fits the process profiling model well, yielding average distances lower than without profiling and much lower than the *Next Available* scheduling scheme.

Across all applications, there was a **25% improvement** from the state-of-the-art scheduler based solely on location based scheduling to the proposed scheduler, which combined the location based scheduling with a predictive scheduler.

4.2 Future Work

The work in this paper indicates promising future improvement in Network-on-Chip scheduling. To verify further the results, an accredited network-on-chip simulator should be used to run popular benchmarks. While the custom simulator addresses all the pertinent network on chip constraints, it does make many assumptions (such as routing occurring in one cycle without the possibility of deadlock). Larger systems and larger applications should also be explored.

It has been seen that profiling can improve the scheduling of processes and this paper by no means exhausts the limits of application profiling, as seen with the network communication pattern profilers, [8]. Combining the existing work in this area with the process profiling proposed in this paper should yield a close to optimal scheduling algorithm, one that is still practical for implementation. A greedy scheme was used to reserve and place process, but using a local search technique, as in [24], may yield benefits outweighing the overhead of a more advanced algorithm.

current results justify additional work in this area

5. CONCLUSIONS

As mesh topology multi-processor systems (NoCs) grow in size, the importance of scheduling interdependent processes increases. The state-of-the-art profiling of network-on-chip communication and scheduling schemes (which take distance into account) each yield passable solutions, but they leave room for improvement as communication is the bottleneck of a NoC system. This paper presented a scheduling scheme which takes distance into account while also profiling the processes of each application to anticipate their near-future needs. While the improvements are not orders of magnitude greater, they are constant across a diverse set of applications while using a light-weight, profiling scheduler.

6. REFERENCES

[1] http://tds.cs.byu.edu/tds/.
[2] *OMNet++*. http://www.omnetpp.org/.
[3] S. Bani-Mohammad, M. Ould-Khaoua, and I. Ababneh. An efficient non-contiguous processor allocation strategy for 2d mesh connected multicomputers. *Inf. Sci.*, 177(14):2867–2883, 2007.
[4] T.-S. Chen, C.-Y. Chang, and J.-P. Sheu. Efficient path-based multicast in wormhole-routed mesh networks. *J. Syst. Archit.*, 46:919–930, August 2000.
[5] C.-L. Chou and R. Marculescu. User-aware dynamic task allocation in networks-on-chip. *Design, Automation and Test in Europe*, 2008:1232–1237, 2008.
[6] U. M. Colesanti, C. Crociani, and A. Vitaletti. On the accuracy of omnet++ in the wireless sensornetworks domain: simulation vs. testbed. In *Proceedings of the*

4th ACM workshop on Performance evaluation of wireless ad hoc, sensor,and ubiquitous networks, PE-WASUN '07, pages 25–31, New York, NY, USA, 2007. ACM.

[7] W. J. Dally and B. Towles. Route packets, not wires: on-chip interconnection networks. In *DAC '01: Proceedings of the 38th annual Design Automation Conference*, pages 684–689, New York, NY, USA, 2001. ACM.

[8] R. Das, O. Mutlu, T. Moscibroda, and C. Das. Application-aware prioritization mechanisms for on-chip networks. In *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, pages 280 –291, dec. 2009.

[9] K. Fall and K. Varadhan. *The ns Manual*, May 2010.

[10] S. Felperin, P. Raghavan, and E. Upfal. A theory of wormhole routing in parallel computers. *IEEE Trans. Comput.*, 45(6):704–713, 1996.

[11] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 3 edition, 2003.

[12] K. Ho and K.-H. Cheng. A two-dimensional fibonacci buddy system for dynamic resource management in a partitionable mesh. In *Aerospace and Electronics Conference, 1997. NAECON 1997., Proceedings of the IEEE 1997 National*, volume 1, pages 195 –201 vol.1, jul. 1997.

[13] J. Hu and R. Marculescu. Application-specific buffer space allocation for networks-on-chip router design. In *Proceeding of International Conference on Computer Aided Design*, 2004.

[14] J. Hu and R. Marculescu. Communication and task scheduling of application-specific networks-on-chip. In *IEE Proceedings of Computers and Digital Techniques*, 2005.

[15] G. Keramidas, V. Spiliopoulos, and S. Kaxiras. Interval-based models for run-time dvfs orchestration in superscalar processors. In *CF '10: Proceedings of the 7th ACM international conference on Computing frontiers*, pages 287–296, New York, NY, USA, 2010. ACM.

[16] E. Leonardi, F. Neri, M. Gerla, and P. Palnati. Congestion control in asynchronous, high-speed wormhole routing networks. *Communications Magazine, IEEE*, 34(11):58 –69, nov. 1996.

[17] K. Li and K. H. Cheng. A two dimensional buddy system for dynamic resource allocation in a partitionable mesh connected system. In *CSC '90: Proceedings of the 1990 ACM annual conference on Cooperation*, pages 22–27, New York, NY, USA, 1990. ACM.

[18] Q. Liu, A. Deshmukh, and K. Tomko. Synthetic simulation of mesh-based parallel applications driven by fine-grained profiling. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, page 8 pp., apr. 2005.

[19] Z. Liu and A. Chien. Hierarchical adaptive routing: a framework for fully adaptive and deadlock-free wormhole routing. In *Parallel and Distributed Processing, 1994. Proceedings. Sixth IEEE Symposium on*, pages 688 –695, oct. 1994.

[20] O. Moreira, J. J.-D. Mol, and M. Bekooij. Online resource management in a multiprocessor with a network-on-chip. In *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*, pages 1557–1564, New York, NY, USA, 2007. ACM.

[21] M. Palesi, D. Patti, and F. Fazzino. *Noxim the NoC Simulator*. University of Catania, 2010.

[22] A. Snavely, N. Wolter, and L. Carrington. Modeling application performance by convolving machine signatures with application profiles. In *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*, pages 149 – 156, dec. 2001.

[23] L.-W. Wang. Deadlock-free dynamic routing in wormhole-switched network-on-chip. In *Proceedings of the International Conference on Communications, Circuits, and Systems*, 2010.

[24] N. Weng and T. Wolf. Analytic modeling of network processors for parallel workload mapping. *ACM Trans. Embed. Comput. Syst.*, 8(3):1–29, 2009.

[25] T. Wood, L. Cherkasova, K. Ozonat, and P. Shenoy. Profiling and modeling resource usage of virtualized applications. In *Middleware '08: Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, pages 366–387, New York, NY, USA, 2008. Springer-Verlag New York, Inc.