Low Contention Mapping of Real-Time Tasks onto a TilePro 64 Core Processor

Christopher Zimmer and Frank Mueller
North Carolina State University, Raleigh, NC 27695-8206, mueller@cs.ncsu.edu

Abstract

Predictability of task execution is paramount for real-time systems so that upper bounds of execution times can be determined via static timing analysis. Static timing analysis on network-on-chip (NoC) processors may result in unsafe underestimations when the underlying communication paths are not considered. This stems from contention on the underlying network when data from multiple sources share parts of a routing path in the NoC. Contention analysis must be performed to provide safe and reliable bounds. In addition, the overhead incurred by contention due to inter-process communication (IPC) can be reduced by mapping tasks to cores in such a way that contention is minimized.

This paper makes several contributions to increase predictability of real-time tasks on NoC architectures. First, we contribute a constraint solver that exhaustively maps real-time tasks onto cores to minimize contention and improve predictability. Second, we develop a novel TDMA-like approach to map communication traces into time frames to ensure separation of analysis for temporally disjoint communication. Third, we contribute a novel multi-heuristic approximation, HSolver, for rapid discovery of low contention solutions. HSolver reduces contention by up to 70% when compared with naïve and constrained exhaustive solutions. We evaluate our experiments using a micro-benchmark of task system IPC on the TilePro64, a real, physical NoC processor with 64 cores. To the best of our knowledge, this is the first work to consider IPC for worst-case time frames to simplify analysis and to measure the impact on actual hardware for NoC-based real-time multicore systems.

1. Introduction

Distributed software models on network-on-chip (NoC) processor architectures provide significant advancements but also challenges for real-time systems. These advancements come from simplifications in processor cores that result in increased accuracy of static timing analysis, simplified scheduling algorithms due to an abundance of cores, and synchronization free data resource models implemented through explicit inter-process communication (IPC) in the form of messages. Due to these advancements, this processor architecture is seeing increased use in hard real-time systems such as in [24] where the authors explore real-time hazard detection in satellites using the Opera Maestro processor [10], a radiation hardened TilePro with 49 cores developed by Boeing. A drawback of these processors is posed by NoC contention of multiple tasks. Such contention exists for shared-memory accesses, for off-chip memory references and for message passing when utilizing distributed software models instead of shared memory. Our work focuses on message passing over the NoC assuming separate NoC interconnects for memory, coherence, I/O and messaging [3]. Other work on increasing predictability and coping with non-uniform memory latencies is orthogonal [4].

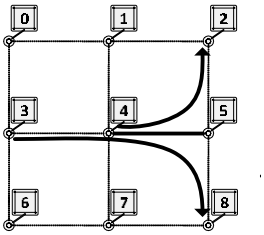

Figure 1. NoC Contention (Config 1)

Message-based communication over the NoC has been shown to increase scalability compared to shared-memory programming [7]. We conjecture that it can also assist in increasing predictability by decreasing contention as it is easier to analyze messages statically than shared memory references [21]. Even under message passing, poor task-to-core mappings can result in a loss of predictability due to latencies incurred through NoC contention. Consider a mesh NoC with full-duplex links, i.e., two messages traveling in opposite directions over a link do not result in contention, that utilizes static dimension-ordered wormhole routing favoring horizontal routing before vertical [3]. Consider the example "Config 1" in Figure 1 of nine cores with a mesh NoC. Two messages are sent, one from core $4 \rightarrow 2$ and the other from $3 \rightarrow 8$, as depicted by the lines with arrows. When sent at the same time, contention on the link $4 \rightarrow 5$ (depicted as a thick link in the NoC mesh) results in a delay for one of these messages due to arbitration within the NoC hardware routers. (Packets are not interleaved as an open virtual channel monopolize links between endpoints.) As a result, sending tasks experience highly variable latencies. Such variability can be reduced or even eliminated when tasks are layed out intelligently to lower or even completely avoid contention, respectively. The effect shown in this example is amplified as the size of NoC meshes increases resulting in larger paths through networks and communication that is more frequent.

This work was supported in part by NSF grants CNS-0720496 and CNS-0905181

We propose an abstraction model for message-based NoC contention that, when applied to statically scheduled hard real-time tasks, allocates messages into temporal windows, so called “frames”. These frames provide the foundation for static analysis on communication paths to evaluate task-to-core mappings. We formulate a constraint problem and implement an “exhaustive solver” that provides optimal mappings. Unfortunately, exhaustive approaches do not scale beyond small NoC mesh sizes as they can take days to solve mapping layouts. Hence, we further develop a multi-heuristic solver, called “HSolver”. We identify a set of effective mapping patterns that yield near-optimal results while providing sustained scalability in finding such solutions even for large NoC meshes. Finally, we contribute a micro-benchmark that empirically tests our designed model and evaluate our approach on a Tilera TilePro processor with 64 cores [3]. Using our solvers, we are able to reduce contention by over 70% compared to a naïve and the constrained exhaustive solutions for NoC sizes too large to be solved optimally within hours.

To the best of our knowledge, this is the first work to address predictability of NoC communication via framing messages into temporal windows for real-time tasks. Previous work [11] viewed communication as temporally stateless. This limited the amount of communication that could feasibly be solved. It also resulted in solutions that were overly conservative in that *any potential* for common message routes were considered contention. By using temporal windows, we are able to solve the problem by *separating temporally disjoint messages* when analyzing link contention scenarios and thus increasing communication predictability.

2. Background

Prior work on conservation cores shows the potential of heterogeneous multicores where cores are specialized according to application demands and parallelization constraints [23], [15]. In such multicore scenarios with mesh topologies, task-to-core mapping particularly affect applications with real-time constraints due to time perturbation resulting from NoC contention. Our work considers a NoC model in which a single task is mapped to a single core, i.e., where an abundance of cores is available (but only some of them are active at any time) or where core specialization requires such a mapping (as for conservation cores). Such abstractions, sometimes referred to as “dark silicon” [13], [12], assume that cores are abundant, yet not all cores can necessarily be active simultaneously due to thermal/power constraints or lack of parallelism. For example, Intel’s SCC faces such a performance/power tradeoff that requires core deactivation and/or dynamic voltage and frequency scaling [1]. This model reduces real-time scheduling to core activation/deactivation, which simplifies static timing analysis as it eliminates the penalties and complexities due to the caching effects of context switching. We gain further accuracy in our distributed model by eliminating the need to analyze shared memory and resource sharing. Instead, we focus on explicit message passing.

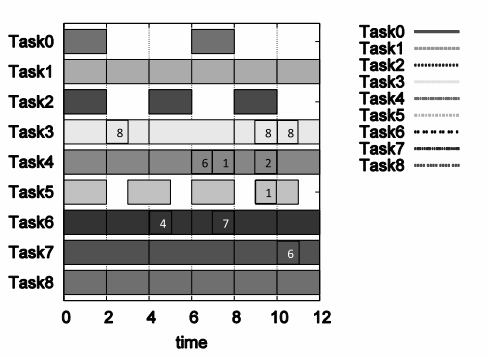

Figure 2. Temporal Framing Example

3. Software-Based Temporal Framing

Temporal framing is a technique similar to time-division multiple-access (TDMA) that imposes frames to bound communication into time windows. It differs from TDMA in that it does not limit the amount of IPC in a single frame; instead, it is used solely for analysis purposes to try and statically map tasks to processors and to reduce communication interference. Programmatically, this is facilitated using self-referential frame checking within a task for identifying when a specific message can be sent/received. To guarantee predictability within a real-time environment, we assume that senders and receivers are at least predetermined within the hyper-period of a periodic task set and that communicating pairs are guaranteed to be active during any frame in which they send or receive data. This assumption is easily supported under the dark silicon model because high utilization and delays due to resource sharing do not exist. Because of this assumption, we can then use temporal frames as a means of synchronizing senders and receivers to reduce latency incurred through non-synchronized IPC.

As an example consider the nine real-time tasks shown in Figure 2 where tasks are represented by shaded blocks. In this figure, the execution of the system is broken into twelve temporal frames. Communication in a frame is represented by indicating within a sender’s frame the receiver address. Communication too large to fit within a single frame is considered across multiple frames. For example, Task 3 send a message to Task 8 during frames nine and ten (see Figure 2). Using this model, we formulate a mathematical model to map tasks to cores while maintaining temporarily disjoint communication.

In this model, we abstract out directional links from a core and its switch (input queues and an output port, see Figure 1) without affecting correctness: Output contention cannot occur between core and switch as only a single task, i.e., only a single sender exists per core. Input contention could result when multiple tasks send to a single receiver along disjoint paths to a destination switch but only one

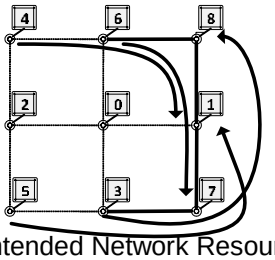

Figure 3. Contended Network Resource (Config 2)

message can be delivered at a time via the switch-to-core link. Such contention is permissible under our model, and we show how blocking under input contention can be bounded by the number of senders. As an example, if task 6 sends a message to task 8 in frame 2 of Figure 2, sends from tasks 3 and 6 would result in input contention at task 8, the receiver of both. Whichever send, that of task 3 or 6, comes later, it would block until the earlier message has been received due to input contention. In the following, we will explicitly refer to "input contention" and otherwise use the term contention to refer to "link contention" within the mesh.

4. Motivation

Let us provide a motivational example to assess the impact of contention-based latency on real-time tasks. We use a 3x3 NoC with nine tasks broken into 12 temporal frames as described in the previous section, our "running example" used throughout the paper. The randomly generated task set has high utilization that takes advantage of the NoC architecture using message-based IPC. There are 10 messages that are sent within the hyper-period. These messages are shown in the temporal framing example in Figure 2. We evaluated three layouts of the tasks on the NoC, each with different amounts of network contention, to show the impact of contention on jitter.

We first evaluate a contention scenario based on a naïve layout, "Config 1", as shown in Figure 1. In this layout, the tasks are mapped to the core corresponding to the task id. The naïve layout results in contention along edge $4 \rightarrow 5$. This contention is a result of two simultaneous messages, as previously described. There is actually an additional message during this frame 9 from Tasks $5 \rightarrow 1$ (see Figure 2) that does not result in contention due to link duplexing.

The second contention scenario, "Config 2" in Figure 3, contains a shows the effect of contention across multiple temporal frames and its effect on jitter. Temporal frames 7 and 9 result in contended links between two sending/receiving pairs in each frame.

We also evaluate a third layout without contention, "Optimal" in Figure 4. No routes are shown in this figure due to the absence of contention since links are full duplex, i.e., edge traversals initiated on opposite ends of a link do not result in contention.

The graph in Figure 5 shows a comparison of the jitter across the worst case transfer times (WCTT), i.e., the maximum measured transfer latency for each transfer in the

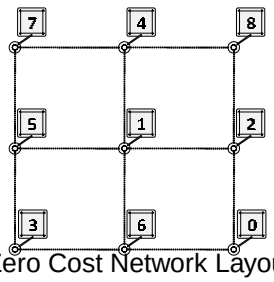

Figure 4. Zero Cost Network Layout (Optimal)

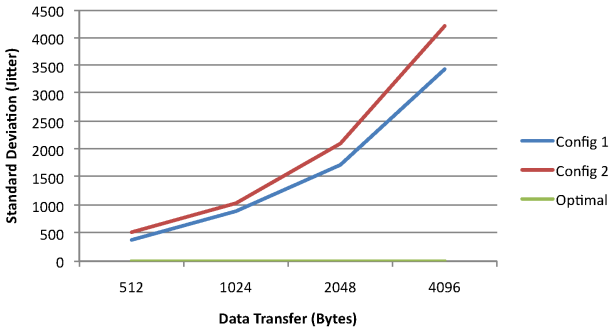

Figure 5. Contention Related Jitter

presence of contention, measured over the course of 100 task executions for each of the layouts. The x axis indicates message size, and the y axis shows the amount of jitter in CPU cycles on a 700 MHz TilePro64. As message sizes increase, jitter and WCTT along edges with link contention also increase for both Config 1 and Config 2. Additional contention across temporal frames leads to greater jitter within the system as seen by high jitter for Config 2 than Config 1. Results for Optimal, a zero-cost layout, show constant and low jitter as data transfer sizes increase. It is important to note that small jitter is incurred even in optimal layouts as NoC grid sizes increase due to added latency from additional hops to traverse the network. These results emphasize the necessity to consider and minimize task layout and network contention on NoC architectures for real-time systems.

5. Exhaustive Solver Model

We utilize the temporal framing model described previously as a basis for a constraint programming formulation. This reduces contention, ease analysis, and maintain communication flexibility. The constraint framework allows us to systematically determine optimal task-to-core mappings for NoC architectures. In the formalization, the set of tasks is considered as a temporal frame graph shown in Figure 6. The figure depicts the graph representation of our running example representing any inter-task communication, and edge weights indicate the time frame during which the communication occurs. We construct temporal frame graphs based on the specification of communication within the real-time task sets (as in Figure 2) and map them onto a core graph, a representation of the NoC topology. We then formulate a constraint-programming model that is solved

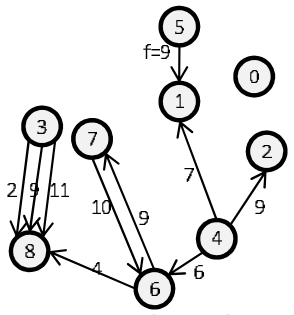

Figure 6. Temporal Framing Graph

in a branch-and-bound traversal to enumerate and evaluate all possible mappings so that the amount of IPC based contention is minimized. The following definitions specify the constraint model for determining an optimal solution (there may be more than one) that minimizes contention.

Definition 1: A Temporal Frame Graph $TFG = (T, C)$ is a weighted and directed graph, where $t_i \in T$ represents tasks in a real-time system and a directed edge $c_{i,j,f} \in C$ represents communication between two vertices (t_i, t_j) where t_i is the sender and t_j is the receiver within the temporal frame f indicated by c_f.

Definition 2: A Directed Mesh Graph $G = (V, E)$ is a representation of cores over a NoC where $v_i \in V$ represents cores on a NoC and $e_i \in E$ is an edge between two cores v_i, v_j identified in directional order by (v_i, v_j).

Definition 3: A function $Map(t)$ maps vertex $t \in T$ onto a vertex $v \in V$.

Definition 4: An ordered set $Path(v_i, v_j)$ denotes the XY dimension ordered edges on the Manhattan path [19] (edge traversal) between $v_i, v_j \in G$.

Definition 5: A function $Cross(v_i, v_j, v_m, v_n)$ is defined as

$$Cross(v_i, v_j, v_m, v_n) = \begin{cases} |Path(v_i, v_j) \cap Path(v_m, v_n)| & if\ v_i \neq v_j \wedge v_m \neq v_n \wedge |Path(v_i, v_j) \cap Path(v_m, v_n)| > 0 \\ 0 & Otherwise \end{cases}$$

.

Definition 6: Function $Cost(c^1, c^2)$ with $c^1, c^2 \in C$ is defined as

$$Cost(c^1, c^2) = \begin{cases} Cross(Map(t_i^1), Map(t_j^1), Map(t_i^2), Map(t_j^2)) & if\ c_f^1 \neq c_f^2 \\ 0 & Otherwise \end{cases}$$

.

Definition 7: The objective function (to be minimized) is $Min(TFG) = \sum_{c^1 \in C, c^2 \in C} Cost(c^1, c^2)$.

Definition 8: A set of constraints on the minimization function are defined as $\forall t_1 \in T, \forall t_2 \in T : t_1 \neq t_2 \implies Map(t_1) \neq Map(t_2)$.

The constraint framework defined above specifies an optimization problem whose cost is to be minimized, i.e., the cost associated with mapping the TFG onto a Mesh Graph G. To understand the cost function, let us revisit the definitions. Between any two vertices in the core graph there is an ordered set $Path(v_i, v_j)$ that represents the set of edges traversed over an XY dimension-ordered route between v_i and v_j (see Def. 4). The set contains edge tuples in which $< v_x, v_y >$ defines a single edge between v_x and v_y. This tuple is strictly (directionally) ordered such that $< v_x, v_y >$ and $< v_y, v_x >$ refer to separate edges to support the notion of full duplex edges that exist in many NoC architectures. To determine the conflicts that occur between two paths, we define the function $Cross(v_i, v_j, v_m, v_n)$ that specifies the cardinality of the intersection of the two paths defined by $(v_i v_j)$ and (v_m, v_n).

The $Cross$ function is used to define a scalar function $Cost(c^1, c^2)$ parametrized by two edges obtained the TFG graph (see Def. 6). It then applies the map function on source and destinations in c^1, c^2 and determines the number of contended links that exist between the two paths. This determines the total number of contended links that exist on the paths defined by c^1 and c^2 assuming that c^1 and c^2 occur during the same temporal frame. Otherwise, the result is zero since communication does not exist during the same temporal frame and thus no messages can interfere with one another. The optimization function of this constraint framework is minimizing the sum of the cost functions across all edges in the TFG. The constraints to bound this function enforce a unique mapping, *i*.e. each task is mapped to one core and no two tasks share a core (see Def. 8).

6. Heuristic Model

Branch and bound, exhaustive optimization solvers scale exponentially as the number of variables grow. This holds true in our exhaustive contention solver detailed in the previous section. Solutions for NoCs 5x5 and larger can take hours to obtain an optimal solution. This is particularly true when the full depth of the search tree has to be traversed, even when optimized in C and parallelized over multiple nodes (using MPI) as in our implementation. We developed HSolver, a heuristic solver to create a low contention layout for NoCs too large to be solved exhaustively. HSolver is a multi-heuristic solver designed based on patterns identified from optimal solutions generated from the exhaustive solver.

HSolver composes multiple heuristics and generates fast and low contention mappings of tasks to cores based on communication traces. It determines the lowest cost solution over a set the heuristics during the mapping process. HSolver applies two classes of heuristics at each stage of the mapping process: (1) task selection heuristics and (2) core selection heuristics. The base algorithm operates by choosing a task selection heuristic and then mapping it to each unmapped core identified by each of the core selection heuristics, ultimately mapping it to the core that results in the lowest contention cost (local minimization). The mapping process for each task terminates after evaluating every core or when a single mapping results in an unchanged system cost. In this section we use the term degree to denote the edge degree of the corresponding temporal frame graph.

6.1. Task Mapping Heuristics

HSolver applies three different heuristic selection techniques with each of the core heuristic strategies to determine the lowest cost solution.

(1) **Maximum Degree First**: This selection strategy is based on the premise that high degree solutions will result in high contention on a given NoC with few scheduling choices to avoid contention. Here, we seek to schedule the highest degree tasks first so that we have a higher degree of flexibility when scheduling subsequent tasks adjacent in the TFG. This heuristic performs best in situations where only a few tasks communicate frequently. In such a situation, the few communicating tasks are scheduled to cores early offering the most scheduling alternatives to reduce contention cost. We expect that this heuristic will be used frequently.

(2) **Minimum Degree First**: The lowest degree selection strategy chooses tasks starting with the least frequently communicating one first. Using either sending or receiving activity increases the communication degree. This strategy is the inverse of the previous strategy. We do not expect frequent use but include it for symmetry.

(3) **Maximum Cross Chat First**: This strategy operates on the principle of scheduling tasks together that frequently communicate. The selection heuristic starts by scheduling the task with the highest degree onto the empty core map. We then select subsequent tasks that most frequently communicate with the currently scheduled group if the number of message exchanges with this group is greater than a predetermined minimum threshold. When no tasks to be scheduled remain within that group, determined through graph connectivity within the TFG, we schedule the remaining highest degree task and begin group scheduling again. This heuristic is based on a common pattern seen within the optimal solutions of frequently communicating partitions. We expect this to be a frequently used heuristic for high-frequency communicating task sets.

(4) **Minimum Cross Chat First**: This strategy operates on a trivial change to the Maximum Cross Chat First strategy: It schedules the lowest degree nodes first. The objective here is to schedule the smallest partitions first. We do not expect to see this pattern but included it for symmetry.

6.2. Core Mapping Heuristics:

The following are strategies used to select the order of the cores to evaluate an also selected single task at a time.

(1) **Maximum Degree First**: NoC tiles on 2D meshes have a varying number of communication edges dependent on their location. Innermost tiles contain four communication edges, non-edge corner tiles contain three communication edges, and corner tiles contain two communication edges. This technique attempts to schedule tasks to high degree cores first to try to reduce opportunities for contention later. This example is based on optimal solutions that map frequently communicating tasks to highest degree cores. This gives high degree tasks more flexibility in establishing communication channels over the NoC without contention.

(2) **Maximum Cross Chat First**: Similar to the task mapping strategy, this core mapping strategy attempts to place tasks with high degrees of common communication physically close. When a task is selected, the core mapping heuristic analyzes the current mapping layout to determine the task on the map with the largest common communication degree. If the highest degree of cross chat is greater than a predetermined threshold, it will then determine an empty core within the fewest number of hops and attempt to schedule the task in that location. If the task selection heuristic selects a task below the threshold, the core selection heuristic places the task as far away as possible from the previously mapped groups.

(3) **Spiral Out-To-In**: This mapping strategy orders core selection onto a spiral traversal of the NoC starting at the first core in the Cartesian space and the assigning cores as a spiral around a matrix. This solution is most effective when scheduling lowest degree tasks first. This places low degree tasks along the low edge count tiles on the outside edges of the NoC, with the highest degree tasks toward the center of the NoC. We expect Maximum Degree First selection to work equally well in most scenarios and thus do not expect this core selection heuristic to be used frequently.

(4) **Spiral In-To-Out**: This mapping strategy reverses the allocation order of the Out-To-In spiral and allocates starting with the internal high degree nodes. This allocation strategy while originally included for symmetry is expected to work quite well when paired with task selection strategies that schedule high degree tasks early, giving more flexibility in location to high degree tasks. Later contention then would occur mainly from placement of lower edge tasks and may result in more frequent use of this selection technique.

7. Micro-Benchmark

To evaluate the effect of task mappings onto NoCs of a a real processor, we have designed and implemented a micro-benchmark that emulates the layouts and message traces on actual hardware. We found it necessary to implement our own benchmark due to a lack of NoC-level message passing benchmarks for massive multicores (e.g., Parsec [9] and Splash-2 [26] support shared memory only while NAS [6] relies on heavy MPI semantics with collectives). Our micro-benchmark provides cycle-accurate measurements on this hardware platform and allows the distinction between hardware or the software overheads. Our benchmark implements the temporal frames abstraction described previously.

The micro-benchmark requires as input the real-time schedule, message traces, and a mapping layout. The complete framework and inputs are shown in Figure 7. These inputs are divided into various phases based on the message traces and real-time schedule. These phases are defined

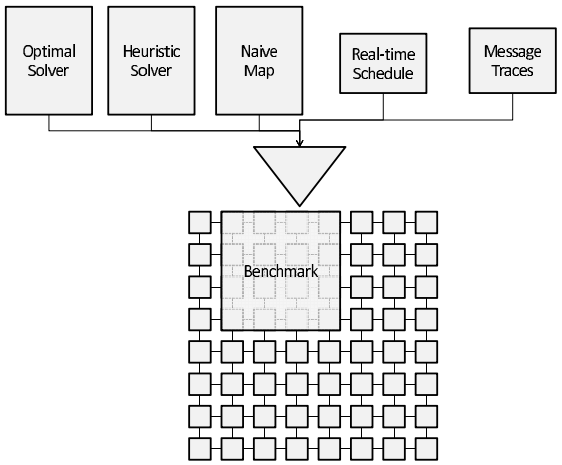

Figure 7. Tilera Evaluation Framework

as computation, sending, receiving, and idle phases. The benchmark uses the real-time task configuration to determine the emulated NoC size and lays out the task set on a contiguous grid of the configured NoC according to the layout designated.

The benchmark is deployed on a Tilera TilePro64 processor running at 700 MHz that contains 64 cores in an 8x8 NoC mesh and six independent mesh networks for memory, coherence, I/O, etc. Among these meshes is a low latency message-passing network called the user-dynamic network (UDN) that can be used to transfer messages of sizes up to 1KB at a time. The network is 32 bits wide and supports bi-directional communication. Messages are transferred via wormhole routing that locks the XY ordered Manhattan path between two cores during the transfer of a message. Further exploration of the Tilera message passing network can be found elsewhere [25]. The benchmark is designed to take advantage of the dataplane options supported by the Tilera architecture that include a low overhead operating system variant of Linux called Zero-Overhead-Linux. Using this platform, our benchmark is able to allocate a grid of cores on the TilePro64 to emulate smaller grids that do not service operating system interrupts. This results in very predictable execution. Our micro-benchmark allows us to execute randomly generated task-sets derived from their message traces. We measure the impact of layout, network contention, and temporal abstraction on predictability.

8. Results

We implemented two solvers, an exhaustive solver and HSolver, to find effective mappings that minimize/reduce contention costs. We then compared their outcomes across 100 randomly generated task sets of multiple sizes. Each randomly generated benchmark consisted of a number of tasks equal to the number of cores in the experiment. The period for each task was randomly generated but limited by varying the maximum resultant hyper-period. Varying the maximum hyper-period and the number of messages within each task set allowed us to control the communication density to improve the likelihood of link contention. The exhaustive solver was implemented using C++ with MPI and evaluated using 64 cores over 4 nodes with two sockets of AMD Opteron 6128 processors (8 cores per socket). HSolver was implemented in C++ using only a single processing thread within the same hardware configuration. To evaluate this work systematically, we randomly generated multiple real-time task sets and message traces for all evaluated NoC dimensions. Thresholds used in HSolver were dynamically evaluated from two to $1/2$ the number of cores in the NoC and compared to determine which threshold value resulted in the lowest cost solution.

In our first experiment, we compare the minimum solutions for each of the solvers as the complexity of the systems increase. We refer to two different complexity metrics: (a) the size of the grid and (b) the number of messages sent during a task set's hyper-period. Increasing the number of messages increases the probability that during any single time frame multiple transfer pairs contend for the same link, thus decreasing the chances for a zero cost solution. Also, increasing the size of the grid exponentially increases the time to convergence. This requires time limits on the exhaustive solver to avoid indefinitely waiting for solutions. We refer to this in the remainder of the paper as the *constrained exhaustive model*, which only provides an optimal solution if it terminates within the given time bound — otherwise, the minimum within the traversed subspace is returned. We constrain the exhaustive approach to runtimes ranging from three minutes at 4x4 up to one hour for 8x8 NoC sizes.

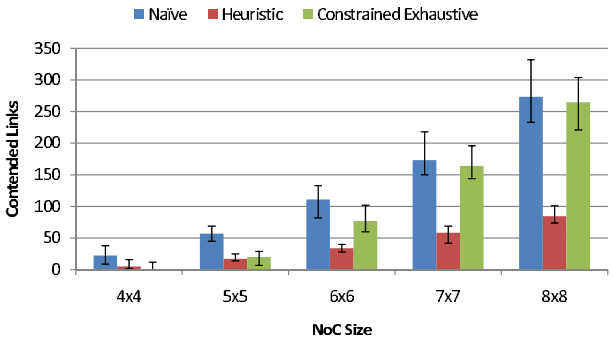

Figure 8. Average Contention for Mapping Strategies

We evaluated the minimum aggregate cost across 100 randomly generated task sets in naïve , heuristic, and constrained exhaustive mappings as the NoC size increases along with a linear increase in the number of messages. The results in Figure 8 show the aggregate contention for each NoC size. For each NoC size plotted over the x axis, three bars report the CPU cycle delays due to links contention (y axis) for the naïve , the HSolver and constrained exhaustive solver, respectively. The x-axis scales quadratically with the size of the NoC. Table 1 shows the averaged amount of

Table 1. Average Solving Times [hh:mm:ss.ms]

NoC	Heuristic	Constrained Exhaustive
4x4	00:00:00.60	00:08:00.00
5x5	00:00:03.00	02:57:00.00
6x6	00:00:14.00	06:46:00.00
7x7	00:00:50.00	06:58:00.00
8x8	00:03:00.00	11:06:00.00

processor time taken by the two solver types for each of the NoC sizes. In the constrained exhaustive approach, after the time cut-off, we still allowed execution to occur but no new branches to be formed. This is why the times vary for the constrained exhaustive solutions as the remaining subspace will still be considered after disabling further branch-outs.

The results show that as the grid and message complexities increase, the solving time for the constrained exhaustive case increases exponentially while HSolver's time increases linearly with mesh sizes. Furthermore, the aggregate contention cost for constrained exhaustive solutions exceeds that of HSolver as mesh sizes increase. The constrained exhaustive solver generates optimal results consistently for NoC sizes of 4x4. At 5x5, it was still able to generate optimal solutions for a large subset of the test cases — but not all of them. At 6x6 and beyond, the constrained exhaustive solver no longer provided optimal solutions or even mappings anywhere close to those of HSolver when its traversal was cut short by timing constraints due to the exponential explosion of the search space. Error bars show the disparity between the minimum and maximum results within each local benchmark grouping. The error bars indicates that HSolver results in solutions that are more consistent across each data set. We also observe a near constant growth rate in overhead for the heuristic approach but a faster (at last second degree polynomial if not higher) growth rate for both the naïve and the constrained exhaustive solvers as solution sizes increase. This is important for assessing scalability for large NoC processors and with high IPC utilization. The growth rate shows that as these metrics increase, naïve and constrained exhaustive approaches will result in real-time systems with quadratically increasing levels of contention. Hence, heuristic schemes seem very promising as the first 100-core processors are said to be released in 2011 with the TileGX [2].

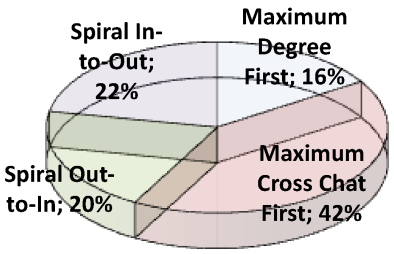

Figure 9. Percent Use of Core Selection Strategies

We also evaluated the HSolver approach to determine the rate at which heuristics were used to generate the low-cost solution. These results allow us to identify which are the important heuristics and to provide a direction for

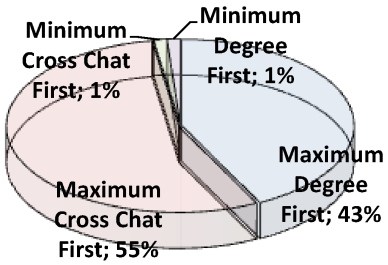

Figure 10. Percent Use of Task Selection Strategies

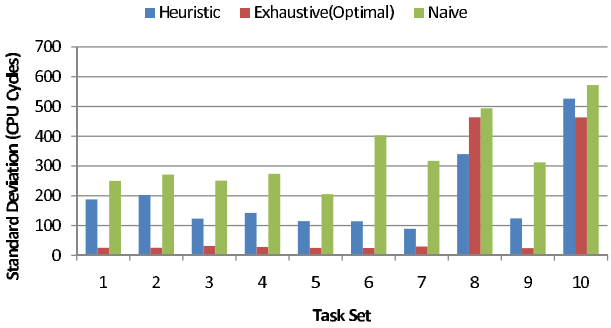

Figure 11. Contention Jitter on a 4x4 NoC

analysis in determining why certain heuristics are more effective. Results in Figures 9 and 10 are taken across all benchmarks generated. Figure 9 shows the core selection strategies and the percent of use of each during heuristic solving. These results show a significant variation in the effectiveness of core strategies. Overall, minimizing the distances between frequently communicating cores is the most beneficial heuristic. This correlates well with the results from Figure 10, where two selection strategies account for 98% of the low-cost solutions. The most effective solution is generally obtained by selecting tasks by maximum cross-chat relative to the currently mapped tasks.

We further conducted experiments to assess the impact of link contention on communication jitter. We evaluated a 4x4 mesh with 10 randomly generated task sets, each containing 200 messages within their hyper-period. Using the exhaustive solver without time constrains to yield optimal results, we determined that only tests 8 and 10 contained schedules where mappings with zero contended links were found. For all cases, time-frame windows of 10μs were imposed. We then calculated the standard deviations over all 2KB transfers that were issued. Figure 11 depicts the standard deviation in clock cycles for different tasks sets for the three mapping approaches. The figure shows that any single contended link can have a significant impact on the standard deviation of transfer latencies. The exhaustive results for all test cases (except for 8 and 10) show that the standard deviation of a system is only affected by cache latencies. Cache warm up is included in these costs and acts as an additive constant on the worst-case transfer times due to additional latencies for data and instruction references. In test cases 8 and 10 the heuristic algorithm shows better performance than the opti-

Table 2. Solving Times per Task Set for a 4x4 Mesh [hh:mm:ss.ms]

Test Case	Heuristic	Exhaustive(Optimal)
1	00:00:00.60	00:00:26.66
2	00:00:00.60	00:04:53.33
3	00:00:00.60	00:02:93.33
4	00:00:00.60	00:02:93.33
5	00:00:00.60	00:01:06.00
6	00:00:00.60	00:00:26.66
7	00:00:00.60	00:00:08.00
8	00:00:00.60	01:10:13.33
9	00:00:00.60	00:00:08.00
10	00:00:00.60	01:00:02.00

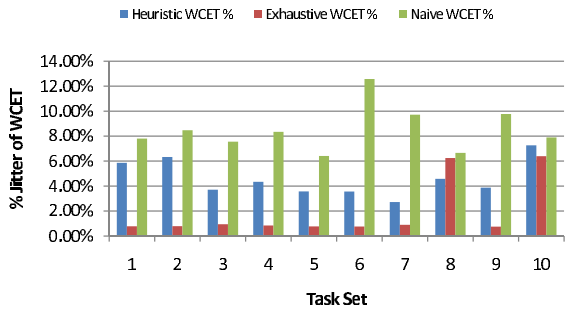

Figure 12. Jitter as a Percentage of Worst Case Transfer Time on a 4x4 NoC

mal contention solver. The heuristic solver and the optimal solver found solutions with the same amount on contention but the mappings were different. Further analysis into these discrepancies indicated varying performance depending on the path lengths of communication resulting in contention.

HSolver was unable to generate any zero-cost solutions for the benchmarks in our 4x4 configuration but was able to show a reduction in jitter of almost 40% when compared with the naïve mapping. To understand this impact, it is necessary to discuss the amount of time required to converge on a solution within the exhaustive and heuristic approaches. Table 2 shows the timing results for each configuration evaluated in this experiment. All results determined by the heuristic approach converged within fractions of a second. Using the exhaustive solver, convergence can take up to 70 of minutes for solutions with contention. As grid sizes grow, convergence grows exponentially for the exhaustive solver while HSolver's convergence times grow linearly. To help correlate the impact of the results, Figure 12 shows the contention jitter as a percentage of the measured worst-case transfer time for the same task sets and solver approaches. These results indicate that naïve mappings can result in jitter of almost 13% of the total worst case transfer time. Jitter of this scale could result in missed deadlines in hard real-time systems due to poor task placement. In comparison, heuristic scheduling reduces that impact by up to 50% while optimal task placement without contention shows less than 1% jitter.

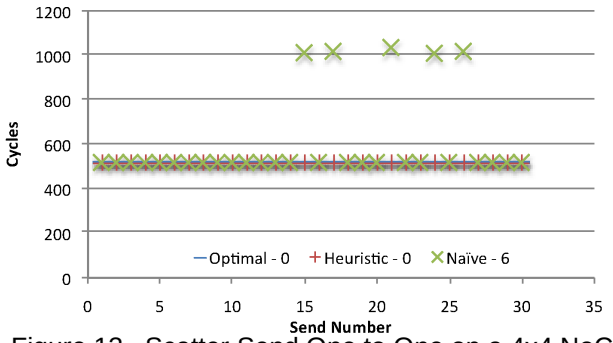

Figure 13. Scatter Send One to One on a 4x4 NoC

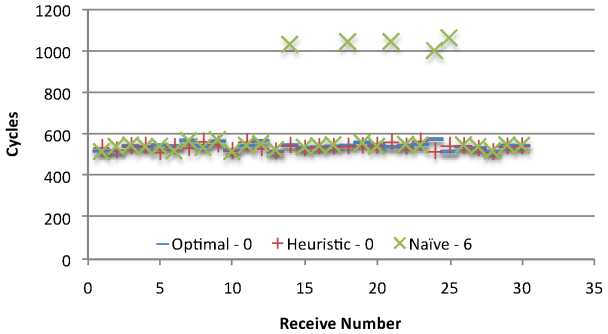

Figure 14. Scatter Receive One to One on a 4x4 NoC

The final experiments illustrate the impact of unavoidable contention on real-time predictability. We previously defined this as input contention (Section 3), i.e., two or more cores send to a single receiver in the same time frame. In this scenario, task placement may result in (unavoidable) contention imposed by the application code. Figures 13- 16 depict the cost for sends and receives for one-to-one and two-to-one pairing of senders/receivers. In taking these results we allowed a cache warm-up period prior to capturing the effects of contention. Senders under one-to-one (Figure 13) experience tight WCET send costs of exactly 515 cycles (irrespective of hop counts), only naïve has higher costs as it results in occasional link contention. Senders under two-to-one (Figure 15) experience overheads in three bands. Contention-free sends require 515 cycles. The other bands result from input contention for a single packet transfer.

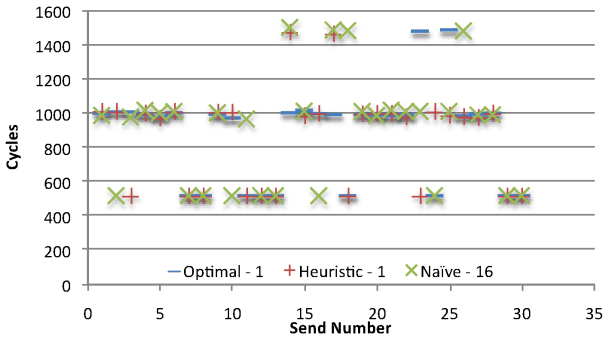

Figure 15. Scatter Send Two to One on a 4x4 NoC

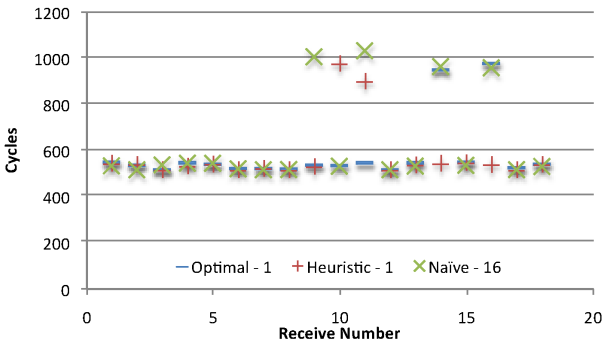

Figure 16. Scatter Receive Two to One on a 4x4 NoC

When a sender is blocked once under input contention, ≈ 1000 cycles are observed. The band around 1500 cycles is due to link contention and input contention. This effect is shown twice for both optimal and heuristic solutions and is a result of a single link contention and input contention on seperate cores. This example shows the worst-case experienced over multiple runs and emphasises the significant impact that contention can have on bounding WCET. In this example the two types of contention discussed lead to an 191% increase in WCET. Figures 14 and 16 depict the receiver overhead for one-to-one and two-to-one transfers, which is quite uniform for heuristic and optimal (link contention free) with occasional higher cost for naïve due to link contention. Occasionally higher receive costs (≈ 1000) for heuristic and optimal (two cases each) appear to correspond to the 1500 send cases. Overall, overheads can be safely bounded for sends (600/1500 cycles) and receives (600/1100 cycles) per packet (without/with input contention) for hard real-time systems. For soft real-time, tighter bounds of 600/600 and 600/1100 for send and receives, respectively, can be provided.

9. Related Work

Bender [8] uses mixed integer linear programming models of heterogeneous multi-cores to solve the task layout problem to guarantee execution time. The solver operates agnostically of the underlying communication mechanism and is unable to model delays due to contention. In contrast, our work operates on a precise model of the underlying network structure for the sole purpose of contention analysis and its effect on execution time. Communication size and time is considered in both their and our work. Murali and De Micheli [20] investigate splitting communication along alternate paths to avoid network based delays. Our work focuses on NoC architectures with static routing, i.e., without alternate path routing, as seen in current multicores. Hu/Marculescu as well as Kuchcinski investigate resource allocation of acyclic graphs to provide timing guarantees [17], [18]. These approaches address heterogeneous multi-core architectures and focus on mapping tasks to the correct processor types while our paper addresses homogeneous architectures and focuses on resource mapping to reduce communication overheads.

More recent work by Chou and Marculescu [11] explores intelligent task mapping to reduce contention in order to increase throughput and reduce the number of hops used per communication to reduce energy consumption. They also utilize a solver model but consider communication without temporal properties. This can lead to overly conservative solutions with less flexibility for mappings compared to our approach. Zhu *et al.* [27] create task to core mappings without considering communication. After the mapping is completed, scheduling is then performed on the communication to reduce contention. Both of these papers operate on a class of programs known as streaming data flow (SDF) programs that are generally related to media with soft real-time constraints. Our focus is on hard real-time systems and considers communication first rather than in a second step. Lee *et al.* [5] investigate reducing power and delays that result from contention on a NoC memory network. In contrast, our focus is on predictability.

Stuijk *et al.* [22] look into resource allocation for multi-processor SDFs to increase throughput. This approach uses TDMA to schedule communication into time slices but keeps communication physically disjoint via hardware. Our temporal frames are TDMA-like and require the programmer to comply with frame access constraints in software. As NoC sizes increase, they offer significant real estate to support several simultaneous messages without contention where TDMA approaches significantly limit available bandwidth. Another area in which our work differs from most of these works is that we do not consider the SDF model but instead focus on a hard real-time model. In SDF models, the impetus is on achieving maximum throughput; our work focuses on reduction of contention to increase predictability.

Goossens *et al.* [14] survey contention free routing via TDM slot reservation for both wires and buffers. TDM in the network is realized at the hardware level. Our work is implemented on top of an architecture that does not provide contention avoidance at the hardware level. Their model is more restrictive and costly in terms of power in that TDM hardware will even be used at times when there is no contention on the network. Our software model allows for variable frame sizing to avoid impeding performance in systems with little contention.

In the context of framing, NoC links and routers could potentially be gated when no messages are crossing them to reduce power consumption [16]. Our work could benefit from such an approach, but we focus on predictability for real-time systems instead of power, and we utilize currently available architectures instead of resorting to simulation.

10. Conclusion

We have designed a temporally aware distributed real-time abstraction for creating contentionless task-to-core

mappings. Using constraint programming techniques we modeled an exhaustive solver to determine optimal mapping for solvable NoCs. Based on heuristics derived from solutions to optimal task layouts, we were able to design a multi-heuristic solver, HSolver, that generates fast and low contention solutions for heavily contended NoCs. Compared with naïve and time-constrained exhaustive solving, HSolver was able to reduce aggregate contention by up to 70% while reducing jitter by up to 40% in our experiments. We additionally contributed a micro-benchmark of task system IPC, implemented and evaluated on a Tilera TilePro64, a state-of-the-art NoC processor with 64 cores. We evaluated 100 randomly generated task sets of increasing NoC size, some containing up to 600 messages in NoCs of 8x8 using 60 temporal frames on the TilePro64. Overall, we contributed a compelling method for a novel approach to contention-based modeling of real-time tasks that communicate via messages over a NoC on a multicore processor and demonstrated how predictability can be significantly improved in such environments.

References

[1] Single-chip cloud computer. blogs.intel.com/research/2009/12/sccloudcomp.php.

[2] Tilera gx processor family. http://www.tilera.com/products/processors/TILE-Gx_Family.

[3] Tilera processor family. http://www.tilera.com/.

[4] Exploration of distributed shared memory architectures for noc-based multiprocessors. *Journal of Systems Architecture*, 53(10):719 – 732, 2007.

[5] Communication-aware task assignment algorithm for mpsoc using shared memory. *Journal of Systems Architecture*, 56(7):233 – 241, 2010.

[6] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, D. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. The NAS Parallel Benchmarks. *The International Journal of Supercomputer Applications*, 5(3):63–73, Fall 1991.

[7] A. Baumann, P. Barham, P.-E. Dagand, T. Harris, R. Isaacs, S. Peter, T. Roscoe, A. Schüpbach, and A. Singhania. The multikernel: a new os architecture for scalable multicore systems. In *Symposium on Operating Systems Principles*, pages 29–44, 2009.

[8] A. Bender. Milp based task mapping for heterogeneous multiprocessor systems. In *Proceedings of the conference on European design automation*, EURO-DAC '96/EURO-VHDL '96, pages 190–197, Los Alamitos, CA, USA, 1996. IEEE Computer Society Press.

[9] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The parsec benchmark suite: Characterization and architectural implications. In *International Conference on Parallel Architectures and Compilation Techniques*, October 2008.

[10] M. Cabanas-Holmen, E. H. Cannon, C. Neathery, R. Brees, B. Buchanan, A. Amort, and A. Kleinosowski. Maestro processor single event error analysis.

[11] C.-L. Chou and R. Marculescu. Contention-aware application mapping for network-on-chip communication architectures. In *International Conference on Computer Design*, pages 164 –169, oct. 2008.

[12] J. Donovan. Arm cto warns of dark silicon.

[13] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. In *International Symposium on Computer Architecture*, pages 365–376, 2011.

[14] K. Goossens, J. Dielissen, and A. Radulescu. Æthereal network on chip: Concepts, architectures, and implementations. *IEEE Des. Test*, 22:414–421, September 2005.

[15] N. Goulding-Hotta, J. Sampson, G. Venkatesh, S. Garcia, J. Auricchio, P.-C. Huang, M. Arora, S. Nath, V. Bhatt, J. Babb, S. Swanson, and M. Taylor. The greendroid mobile application processor: An architecture for silicon's dark future. *IEEE Micro*, 31:86–95, March 2011.

[16] K. C. Hale, B. Grot, and S. W. Keckler. Segment gating for static energy reduction in networks-on-chip. In *Workshop on Network on Chip Architectures*, pages 57–62, 2009.

[17] J. Hu and R. Marculescu. Energy- and performance-aware mapping for regular noc architectures. *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, 24(4):551–562, 2005.

[18] K. Kuchcinski. Constraints-driven scheduling and resource assignment. *ACM Trans. Des. Autom. Electron. Syst.*, 8:355–383, July 2003.

[19] W. Lipski, Jr. An o(n log n) manhattan path algorithm. *Inf. Process. Lett.*, 19:99–102, September 1984.

[20] S. Murali and G. D. Micheli. Bandwidth-constrained mapping of cores onto NoC architectures, 2004.

[21] H. Ramaprasad and F. Mueller. Bounding worst-case data cache behavior by analytically deriving cache reference patterns. In *IEEE Real-Time Embedded Technology and Applications Symposium*, pages 148–157, Mar. 2005.

[22] S. Stuijk, T. Basten, M. Geilen, and H. Corporaal. Multiprocessor resource allocation for throughput-constrained synchronous dataflow graphs. In *Design Automation Conference*, pages 777 –782, june 2007.

[23] G. Venkatesh, J. Sampson, N. Goulding, S. Garcia, V. Bryksin, J. Lugo-Martinez, S. Swanson, and M. B. Taylor. Conservation cores: reducing the energy of mature computations. In *Architectural Support for Programming Languages and Operating Systems*, pages 205–218, 2010.

[24] Villalpando, C.Y., Johnson, A.E., Some, R., J. Oberlin, Goldberg, and S. Investigation of the tilera processor for real time hazard detection and avoidance on the altair lunar lander. In *Aerospace Conference, 2010 IEEE*, pages 1 –9, march 2010.

[25] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. F. Brown III, and A. Agarwal. On-chip interconnection architecture of the tile processor. *IEEE Micro*, 27:15–31, September 2007.

[26] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta. The splash-2 programs: characterization and methodological considerations. In *International Symposium on Computer Architecture*, pages 24 –36, june 1995.

[27] J. Zhu, I. Sander, and A. Jantsch. Constrained global scheduling of streaming applications on mpsocs. In *Asia and South Pacific Design Automation Conference*, pages 223–228, Jan. 2010.