Efficient Exploration of the SoC Communication Architecture Design Space [*]

Kanishka Lahiri
Department of ECE
UC San Diego
klahiri@ece.ucsd.edu

Anand Raghunathan
NEC USA C&C Research Labs
Princeton, NJ
anand@ccrl.nj.nec.com

Sujit Dey
Department of ECE
UC San Diego
dey@ece.ucsd.edu

Abstract

In this paper, we present a methodology and efficient algorithms for the design of high-performance system-on-chip communication architectures. Our methodology automatically and optimally maps the various communications between system components onto a target communication architecture template that can consist of an arbitrary interconnection of shared or dedicated channels. In addition, our techniques simultaneously configure the communication protocols of each channel in the architecture in order to optimize system performance.

We motivate the need for systematic exploration of the communication architecture design space, and highlight the issues involved through illustrative examples. We present a methodology and algorithms that address these issues, including the size and complexity of the design space. We present experimental results on example systems, including a cell forwarding unit of an ATM switch, that demonstrate the benefits of using the proposed techniques. Experimental results indicate that our techniques are successful in achieving significant improvements in system performance over conventional communication architectures (observed speedups over typical architectures such as single shared buses averaged 53%). Moreover, we demonstrate that our design space exploration methodology and optimization algorithms are efficient (low CPU times), underlining their usefulness as part of any system design flow.

I. Introduction

Electronic system design is being revolutionized by widespread adoption of the System-on-Chip (SoC) paradigm. The benefits of using such an approach are numerous, including improvements in system performance, cost, size, power dissipation, and design turn-around-time. In order to exploit these potential advantages to the fullest, a complete design methodology must adequately address two dimensions of system design. Firstly, it is essential to efficiently and optimally map an application's computation requirements to a set of high-performance system components, like CPUs, DSPs, application specific cores, memories *etc*. Secondly, it is equally important to empower a designer with techniques and tools to map the system's communication requirements onto a well optimized communication architecture that is well suited to the specific application at hand. The focus of this paper lies on the second of these two aspects of system design.

Increasing levels of integration are leading to a growing volume and diversity of data and control traffic exchanged among SoC components. As a result, a poorly designed on-chip communication architecture could become a severe impediment to optimal system performance and power consumption. In order to support high-performance components, the on-chip communication architecture must efficiently transport the large volume of heterogeneous communication traffic they generate. Hence techniques to efficiently and optimally map the system's communication requirements to a target communication architecture need to be included as an integral part of any system design flow.

[*]This work was supported by NEC USA Inc., and by the California Micro Program

A. Paper Overview and Contributions

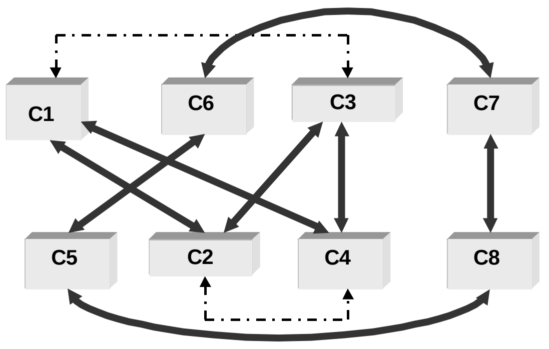
In this paper, we present a design space exploration and optimization technique that takes as inputs a system description that has been partitioned into HW/SW, and mapped onto appropriate components. The exploration technique optimally maps the communication requirements of the system onto a target template communication architecture. The technique can be used to determine the best way to assign system components to the template architecture so as to meet desired design goals. Our technique not only generates an optimal mapping, but also provides a set of statically configured communication protocols for each channel in the architecture that are customized for the derived mapping.

Our technique is based on identifying, through efficient performance analysis, the characteristics of inter-component communication in a given system, including an accurate analysis of potential contentions for shared channels in the architecture. The technique consists of (i) a constructive algorithm to determine an initial architecture which maps various SoC communications to specific paths in a template communication topology, and to select an appropriate communication protocol for each channel, and (ii) an iterative improvement strategy that improves on the quality of the initial solution to generate a well-optimized mapping of communications along with carefully configured communication protocols.

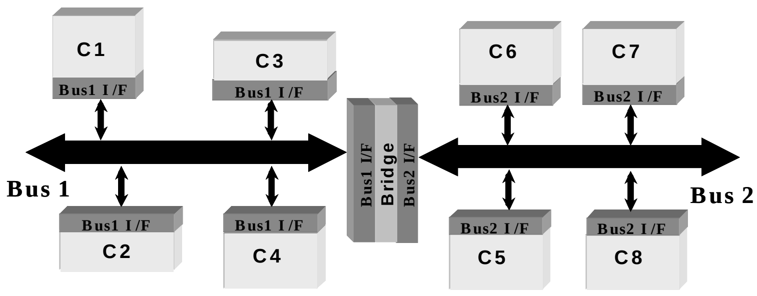
In the following sections we first motivate the need for such exploration techniques by studying the nature and size of the communication architecture design space, and the potential advantages of thorough exploration. We then illustrate the complexity of the problem by showing how a simple approach can result in an architecture that provides substantially sub-optimal performance. Finally we show how the problem is further complicated by the interdependence of the choice of mapping and that of different protocols for each channel in the architecture. We demonstrate how our technique addresses these issues, and detail the various steps and algorithms. Experimental results on example systems, including a cell forwarding unit of an ATM switch, confirm the benefits of using our technique. We show that using our technique, system performance can be improved significantly (upto 53%). Moreover we demonstrate that our technique generates optimized communication architectures at appreciably low cost in terms of CPU times, thus underlining its usefulness as part of any system design flow.

B. Related Work

There is a large body of work on system-level synthesis of application-specific architectures through HW/SW partitioning and mapping of the application tasks onto pre-designed cores and application-specific hardware [1, 2, 3, 4, 5, 6, 7, 8]. While most



(a) Example system of communicating components



(b) Candidate architecture — `Arch1`

Figure 1: Example system of communicating components with a template communication architecture consisting of multiple channel interconnected by a bridge

previous research has focussed on optimally mapping system functionality onto components, comparatively little work has addressed mapping a system's data and control communications onto an on-chip communication architecture.

Research on system-level synthesis of communication architectures [9, 10, 11, 12] mostly deals with synthesis of the communication architecture topology. While topology selection can be a critical step in custom communication architecture design, in this work we address a complementary problem. We assume the topology is predetermined by choosing a template communication architecture, several of which are available from interconnect core providers [13, 14]. The problem we address is how to decide on an optimal assignment of system components to the given architectural template, with a suitable selection of on-chip communication protocols.

Fast and accurate system level performance analysis is the key to a practical design space exploration methodology. Research on system level performance analysis that takes into account effects of the communication architecture, include approaches based on simulation of the entire system, modeling communication at varying levels of abstraction [15, 16]. However the high computational cost of simulation based techniques make them infeasible when exploring a large design space. More efficient static performance estimation techniques include [8, 9, 10, 11, 17], but they do not model dynamic effects like bus contention accurately enough to drive an exploration/optimization methodology. We adopt a trace based performance analysis framework in our work [18, 19], which provides for accurate modeling of dynamic effects, while at the same time being far more efficient than simulation based techniques.

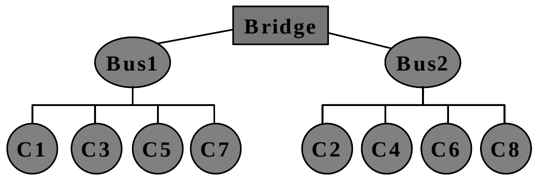
II. Exploring the Communication Architecture Design Space

We formulate the problem of designing a communication architecture for a partitioned HW/SW system as consisting of three steps: (i) the task of defining a communication topology consisting of a network of channels (each serving as a dedicated point-to-point link or as a shared bus) interconnected by bridges, (ii) the task of mapping the system's communications onto paths in the topology (by mapping components onto channels), and (iii) the task of selecting or customizing the protocol for each channel. In this work we focus on the latter two steps, assuming that the designer has selected a template topology. The template could consist of an arbitrary network of shared and dedicated communication links. The reasons we chose this approach are twofold: (i) Several such templates are commercially available to the designer today, including multiple communication channels organized into a hierarchy [14], approaches based on time-division multiplexing [13], *etc.* (ii) Being faced with such choices, we believe the designer should be empowered by automatic tools in order to evaluate alternative templates for a given system. Our techniques aim at providing these tools, using which the designer can optimally map the system to each template in turn using our exploration techniques, and then evaluate each resulting solution using fast system level performance analysis.

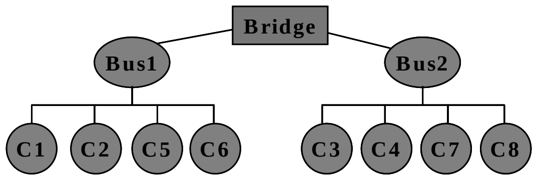
In this section, using examples, we illustrate a few issues that arise in the process of mapping a system's communications to a template communication architecture. We first show that even when a template topology is provided, the design space comprising alternative mappings and communication protocols can be quite large. Moreover, performance variation across the design space is significant enough to motivate thorough exploration. Second, we show that simple techniques to identify which components should be grouped to share a channel (*e.g.*, based on clustering components that communicate frequently) are not sufficient to generate an optimal solution. This is due to the additional complexity introduced by conflicts that arise when multiple components contend for shared communication resources (like a shared bus). Finally, we show that selecting a particular assignment of components to communication channels independent of the on-chip communication protocols can result in significantly sub-optimal designs.

Example 1: Figure 1(a) shows a system consisting of a set of components each of which execute a set of computation tasks and also execute data and control communications. Bold lines indicate transfer of data, while dotted lines indicate exchange of control or synchronization signals. Note that these lines indicate the logical view of inter-component communication, and not physical paths in the communication architecture. Figure 1(b) shows an implementation where all the communications generated by the system components are mapped to either (or both) of two shared buses connected by a bridge. However, given this architectural template, there could be other mappings as well, such as the ones shown symbolically in Figures 2(a) and (b).

In order to illustrate the potential impact of alternative architectures on performance, Table 1 reports the performance of the system under different mappings as measured by a performance analysis tool described in [19]. Each row represents a distinct mapping of components to buses in the communication architecture. For example, in `Arch1`, components C1-C4 are grouped onto one bus, and components C5-C8 are on another bus (Figure 1(b)). In this case, the system takes 11723 cycles to process a fixed sequence of input stimuli. `Arch1` results in 30.6% lesser execution time than `Arch2` (Figure 2(a)), which takes 15314 cycles to process



(a) Alternative solution — `Arch2`



(b) Alternative solution — `Arch3`

Figure 2: Sample points in the design space for the example system of communicating components

the same sequence of input stimuli. However, `Arch1` is 26.8% slower when compared to `Arch3` (9242 cycles), which shows the best performance among the chosen design points. ■

The above example illustrates the following issues:

- The design space of possible mappings can be very large for a system with multiple components and channels. For an arbitrary set of n components and k distinct channels the number of possible mappings is bounded by $O(k^n k!)$. In practice, however, the design space can be pruned somewhat by constraining the number of components attached to each channel. Even so, for small examples such as the one in Figure 1, (a system with 8 components and 2 channels) if we assume that the two buses are identical, and that the number of components on each bus is equal, the number of choices is $\binom{8}{4}$ or 70! Moreover, as shown later, for each mapping, there exist performance critical choices to be made while choosing an implementation of the per channel communication protocols. Combined, these factors result in a challenging optimization problem.
- Performance variation across this design space can be significant, (in this example upto 65%), thereby motivating the need for organized exploration of the design space.
- Knowledge of the application specific characteristics of the communication traffic, like volume, concurrency, and performance criticality of inter-component traffic should be used by the automated exploration tool to prune the search space, and quickly generate an optimal architecture. To illustrate this, note that in the above example, `Arch1` outperforms `Arch2`. The reason behind this is that the mapping of components to buses in `Arch1` is based on clustering components that exchange large volumes of performance critical data.

In the next example we demonstrate that using simple metrics such as the volume of communication traffic to drive the design of the communication architecture need not produce the best mapping.

Example 2: Consider again the system discussed in Example 1, and the choices of alternative mappings in a little more detail. Table 1 reported that system performance under the `Arch1` configuration is superior to that under `Arch2`. The mapping in `Arch1` results in the following: (i) the number of communication transactions that span multiple buses are minimized, and (ii) components that exchange large amounts of performance critical data are attached to the same bus. Both these factors result in low transmission latencies.

However, as we see next, the above approach does not necessarily yield an architecture that produces the best performance. If we change the mapping of components to channels in our example to that of `Arch3` (Figure 2(b)), then we discover that the system completes its task in 9242 cycles, an improvement of 27% over `Arch1`. The reason for the improvement is that the new mapping separates components that have largely overlapping communication lifetimes (C1 from C3 and C5 from C7), resulting in an implementation that causes fewer conflicts and hence enhanced concurrency in the system's execution. However, the price paid is that the number of communication transactions going across the bridge is no longer at a minimum, since component C1 communicates with component C4 and C3 with C2 (Figure 1(a)). For this system, since the benefit of reduced conflicts outweighs the penalty of bridge transactions, `Arch3` is the implementation that best recognizes and takes advantage of the patterns of communication traffic. The example shows a case where a simple clustering heuristic like the one described earlier fails to generate the best solution. ■

Table 1: Performance variation over different points in the design space

Cases	Bus 1	Bus 2	Performance (clock cycles)
Arch1	C1 C2 C3 C4	C5 C6 C7 C8	11723
Arch2	C1 C3 C5 C7	C2 C4 C6 C8	15314
Arch3	C1 C2 C5 C6	C3 C4 C7 C8	9242

Table 2: Effect of communication protocol in the design space

Case	Bus 1 Protocol	Bus 2 Protocol	Performance (cycles)
Arch3-subopt	C1>C2>C5>C6 DMA = 5	C3>C4>C7>C8 DMA = 10	12504
Arch3 -opt	C1>C6>C2>C5 DMA = 10	C3>C7>C4>C8 DMA = 10	9242

Example 3: In this example we illustrate that communication architecture design tools should incorporate the influence of the on-chip communication protocols in order to judge the quality of a candidate solution. Using the same example from Figure 1, recall that the performance analysis results indicated that `Arch3` was the best configuration due to reduced conflict level on each bus. However, the performance estimate for `Arch3` in Table 1 was derived assuming that an optimally tuned bus protocol was assigned to each of the two buses.

Suppose while examining alternative solutions, we ignore the effect of the bus protocol, and assume a fixed protocol while evaluating different solutions. To demonstrate the potentially sub-optimal outcome of such an approach, we ran an experiment where each bus was assigned a static priority based arbitration protocol, with specific protocol parameters. Here we consider the parameter `DMA`, which defines the maximum block transfer size, and the bus access priorities of each component. In the first experiment, we evaluated the `Arch3` configuration (Figure 2(b)), leaving the protocol parameters unchanged from `Arch 2`, (as shown in row 1 of Table 2). The system took 12504 cycles to complete the task, a degeneration of 6.6% over `Arch1`! However, after we regenerated a set of bus protocols that were optimal (row 2 of Table 2), the best performance result of 9242 cycles was obtained. ■

This example demonstrates the following:

- In order to examine a candidate solution, the effects of the on-chip protocols cannot be ignored since they can significantly impact the performance metric. When the set of components mapped to a channel changes, the traffic characteristics on that channel change, and therefore invalidate the optimality of the previously chosen protocols.
- The problems of selecting an optimal mapping of components to channels in the communication architecture and that of choosing the best set of protocols are inter-related. Solving each problem separately, independent of the other, could easily lead to sub-optimal solutions. In our example, such an approach would have made us overlook `Arch3`. In order to evaluate one solution over another we must derive a set of optimal protocols for each architecture, and analyze the performance of the combination of the architecture and its associated protocols.

III. Overall Communication Architecture Design Methodology

In this section, we present an overview of our communication architecture design methodology highlighting the important steps. In the next section, we describe how some of the crucial steps are conducted in more detail.

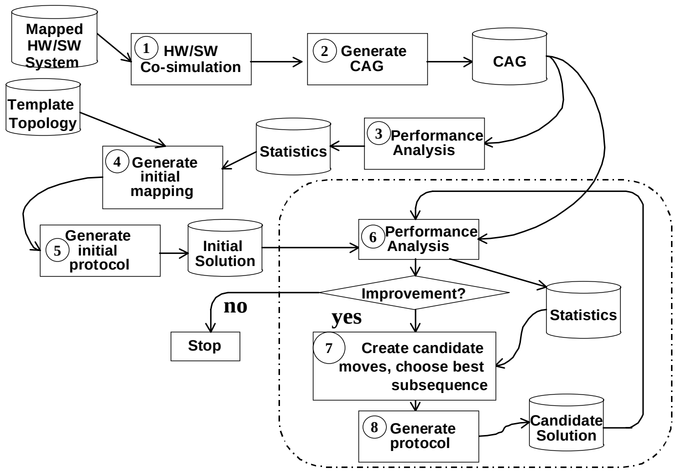


Figure 3: Overall algorithm for design space exploration

The overall methodology is shown in Figure 3. The inputs consist of (i) a system specification that has been partitioned into HW and SW, and mapped to appropriate cores or custom HW, and (ii) a template communication architecture topology consisting of multiple (shared/dedicated) channels interconnected by bridges. The algorithms automatically generate an optimal mapping of system components to specific channels in the target architecture, as well as a set of optimized communication protocols for each channel.

In the first step, HW/SW co-simulation of the partitioned/mapped system description is performed, with communication among components modeled assuming completely parallel exchange of communication events at a fixed rate of data transfer. Execution traces are collected and stored in a compact representation called a *Communication Analysis Graph* (CAG), which captures the abstracted system behavior (including computation, communication, and inter-component synchronization) over the entire simulation trace [19]. Using the analysis algorithms detailed in [18] and [19], Step 3 generates various statistics about the system performance and inter-component communication traffic. Based on these statistics, and a specification of the topology of the template architecture, a constructive heuristic procedure (Step 4) generates an initial mapping of components to the target architecture. Step 5 determines a set of optimal protocol parameters for each channel. The results of Steps 4 and 5 together constitute an initial solution.

As demonstrated in Section II, an approach that stopped here could lead to significantly sub-optimal system performance. Hence the need for the second, iterative part of our technique. In Step 6, the performance analysis tool is re-invoked, to consider effects of the selected communication architecture. By analyzing the CAG, the tool incorporates the effects of the communication architecture, and re-evaluates performance and communication statistics.

Based on these statistics, Step 7 explores alternative solutions by calculating potential performance gains of performing various *moves*, whereby an already assigned component is re-mapped from one channel to another, and chooses the best set of candidate moves to construct a new solution. The output of Step 7 is a new mapping of components to the target architecture. Step 8 chooses an optimal set of protocol parameters, for reasons illustrated in Section II. The new solution is re-evaluated, and the iterative procedure (Steps 6 through 8) is repeated till no further improvement in performance is obtained.

IV. Algorithms for Design Space Exploration

In this section we first describe how Step 4 of Figure 3 makes use of the statistics and topology information to generate an initial solution. We then consider how the iterative part of our technique improves on that solution.

A. Inter-Component Communication Statistics

Statistics generated by the performance analysis of Step 3 are represented in an inter-component *Communication Graph*. The Communication Graph is a directed graph consisting of one vertex for each component, and an edge (v_i, v_j) when there exists communications between component i and j. The direction on the edge is dependent on which of the two components drives the transactions between them. Information on an edge (v_i, v_j) includes several properties of the communication transactions seen between components i and j, including the number of transactions, distribution of their sizes (mean variance *etc.*), critical path information, (expressed as the distribution of their slacks), number of transactions with zero slack (critical transactions), *etc*. While the various parameters on each edge may be used in several different ways, in our implementation, we chose to derive a single weight for each edge by taking the the product of the average size and the number of transactions between v_i and v_j and scaling it by the by the average slack (Figure 4). This takes into account frequency, volume and criticality of transactions that occur place between components i and j.

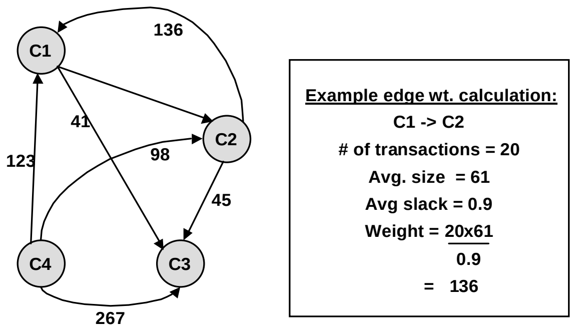
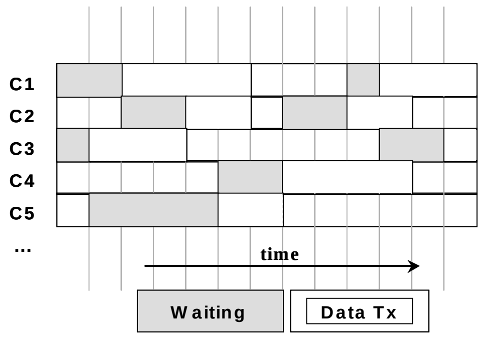
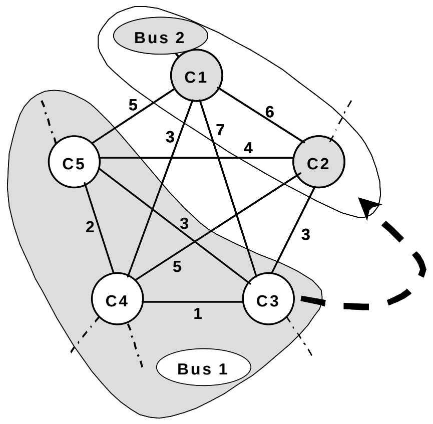


Figure 4: Communication Graph: Example

By examining the Communication Graph, Step 4 calculates for each component, a measure of the *demand* it places on the



(a) Execution trace showing communication lifetimes



(b) Communication Conflict Graph

Congestion Levels		
Bus No.	Before	After
1.	6	2
2.	6	16

(c) Congestion levels when considering $C3 : Bus1 \rightarrow Bus2$

Figure 5: Taking into account conflicts during estimation of gain of performing a move

communication architecture. For component i this is the sum of the weights of the outgoing edges from v_i in the Communication Graph. It then arranges the components in descending order of demand.

B. Modeling the Template Topology

After ranking the components as described above, Step 4 also ranks the channels in the communication architecture by building and analyzing a model of the template communication topology. The topology is modeled using a graph representation. The *Template Graph* consists of one vertex for each channel, and bi-directional edges between vertices whenever two channels are connected by a bridge. The information on these edges contain parameters that describe the properties of the bridge — the overhead of transmitting a single word over it, its frequency of operation *etc*. Each vertex also has a self looping edge, which describes properties of the channel it represents. These include the number of cycles required to transmit unit data (1 word), width of the channel (in bits), the frequency of operation (in Mhz), and the number of handshake cycles that precede each independent communication transaction. From these parameters, a *connectivity* metric for each channel is calculated as follows.

Given a set of communication transactions, with source and destination channels P and Q, we calculate the amount of time it takes to complete the transactions by measuring the following quantities:

- The *initiation delay* is is measured by summing up the handshaking time on each channel and bridge on the path between P and Q. This time is incurred for every independent transaction.
- The *transmission delay* is measured by calculating, for each channel on the path between P and Q the expression $\frac{n}{wf}$ where $n =$ number of cycles required to transmit 1 word, $w =$width of the channel in bits $f =$frequency of operation in Mhz. The channel on the path with the smallest bandwidth determines the rate at which data is transfered from P to Q.

The time taken for a set of communication transactions is given by the sum of the initiation delays and the transmission delays. The total delay of a set of communications involving bus P is used to derive the connectivity metric used to rank bus P. Note that this calculation ignores the possibility of conflicts. The intuition is that channels which have high performance and are "well connected" to the rest of the topology are given a high rank.

C. Construction of the Initial Solution

The initial solution construction procedure picks the highest ranked component that is not yet assigned to a channel and then tries to decide which channel to assign it to. For example, suppose channel P has components i and j assigned to it and component k is being considered for assignment. Its interaction level with channel P is measured by summing the weights of the edges $(v_k, v_j), (v_k, v_i), (v_i, v_k), (v_j, v_k)$ in the Communication Graph. The channel for which this interaction level is maximum is the target channel for component k. If this level exceeds a threshold, then the assignment is made. If not, it implies that too few components have been assigned to make an informed decision using the above technique. In this case, the list of components is scanned for a component which has not yet been assigned, and has maximum interaction with the current component k. Based on the ranks of the two components and the number of components assigned to each channel so far, they are both together assigned to an appropriately ranked channel. If there still exist choices among a few alternatives, a channel is randomly chosen.

Assigning protocol parameters: We assume a static priority based arbitration scheme for each channel since this protocol is used by many bus architectures, *e.g.*, [20]. Hence, the parameters that need to be determined are the priorities of each component, and the maximum permissible block transfer size or DMA size. Priorities are assigned to components sharing a channel by examining the ranks of each component in the sorted list of components generated earlier. The maximum block transfer size for a channel k is calculated from the sub-graph of the Communication Graph that consist of vertices representing components that are assigned to channel k. It is given by a weighted average of the size of transactions among components of the given channel, where the weight incorporates the criticality, derived from the average values of the slack. This favors a large block transfer size when large transactions often lie on the system critical path.

D. Iterative improvement

Here we describe how we construct a sequence of moves (or transformations) to yield a solution that improves the system's performance. Given an assignment of components to channels, with a set of protocol parameters for each channel, Step 7 first executes `Evaluate_Gain(i,P,Q)`, which computes the potential gain of moving component i from its currently mapped channel P to another channel Q ($P \neq Q$), for every combination of component and potential destination channels. The rest of the algorithm uses a well-known variable depth-first-search approach to determine the

```
Evaluate_gain
inputs: COMPONENT $i$, BUS $P$, BUS $Q$
outputs: FLOAT $gain$
begin
    for each $v \varepsilon P$, $u \varepsilon Q$
        $old\_delay = old\_delay +$
            $calculate\_comm\_delays(v, old\_speeds)$;
        $old\_delay = old\_delay +$
            $calculate\_comm\_delays(u, old\_speeds)$;
    end for
    $l_{Q'} = congestion\_level(P)$;
    $l_{Q'} = congestion\_level(Q)$;
    for each component $j$ such that
        $j \varepsilon Q$ && $(i, j) \rightarrow overlap\_cycles \neq 0$:
        $l_Q = l_Q + (i, j) \rightarrow overlap\_cycles$;
    end for
    for each component $k$ such that
        $k \varepsilon P$ && $(i, k) \rightarrow overlap\_cycles \neq 0$:
        $l_P = l_P - (i, j) \rightarrow overlap\_cycles$;
    end for
    $Q \rightarrow new\_speed = \frac{l_Q - l_{Q'}}{l_{Q'}} \times (Q \rightarrow old\_speed)$;
    $P \rightarrow new\_speed = \frac{l_{P'} - l_P}{l_{P'}} \times (P \rightarrow old\_speed)$;
    $send(i, P, Q)$;
    for each $v \varepsilon P$, $u \varepsilon Q$:
        $new\_delay = new\_delay +$
            $calculate\_comm\_delays(v, new\_speeds)$;
        $new\_delay = new\_delay +$
            $calculate\_comm\_delays(u, new\_speeds)$;
    end for
    $gain = old\_delay - new\_delay$;
    return $gain$;
end
```

Figure 6: `Evaluate_gain` procedure

best sub-sequence of moves (whose cumulative gain is maximum) to construct an improved solution [21].

Estimating the potential gain of a move: The potential gain of a move is estimated using additional information that is generated by the performance analysis tool for the architecture under current consideration. Under a given architecture, the lifetime of a communication transaction is made up of three parts: (i) waiting time arising out of handshaking overhead, (ii) waiting time arising out of simultaneous access attempts to the shared channel, and (iii) time taken to transfer the data. The performance analysis engine generates, for each pair of components (i, j), the number of cycles for which the lifetimes of communication transactions driven by i overlaps with transaction lifetimes driven by j. Figure 5(a) shows a set of overlapping communication transaction lifetimes, and Figure 5(b) the resulting *Communication Conflict Graph*.

From the Communication Conflict Graph, for a given mapping of components to channels, a *congestion level* for each channel P is obtained. This is given by summing up the weights for every edge (v_i, v_j) in the Communication Conflict Graph where v_i and v_j represent components mapped to channel P. When considering a move of component i from channel P to channel Q, the algorithm calculates the potential decrease in the congestion level on channel P and the potential increase in the congestion level on channel Q. To illustrate this, consider the example shown in Figure 5(b) where the components are shown grouped into two buses, and component C_3 is being re-mapped from $Bus1$ to $Bus2$. For example, from Figure 5(a) it is clear that on $Bus1$, the number of overlapping time units is 6 (1 between C_3 and C_4, and 2 between C_4 and C_5, and 3 between C_3 and C_5). After removal of C_3, the number of time units of overlap on $Bus1$ reduces to 2 (those between C_3 and C_4) while the number of units of overlap on $Bus2$ increase from 6 to 16. Figure 5(c) shows the congestion levels on each bus before and after moving vertex C_3 from $Bus1$ to $Bus2$.

To calculate the potential gain of moving component i from channel P to Q, the pseudocode of Figure 6 is executed. The first loop accumulates the communication delays associated with communications generated by each component v on channel P followed by each component u on channel Q. This is stored in old_delay. Then the old congestion levels on the each channel are calculated as described earlier and saved in $l_{P'}$ and $l_{Q'}$. Next, the new congestion levels on each channel are calculated. The second loop calculates the increased congestion level l_Q on channel Q (the destination channel), and the third loop calculates the decreased congestion level l_P on channel P (the source channel). Then the speed of each channel is symbolically scaled by the congestion level on each channel. The time taken for all transactions involving channels P and Q are recalculated (*new_delay*), and compared with the previous value. The difference is the potential gain of performing the move. At each step in the iterative procedure, the move that is chosen is the one producing the maximum gain. Note that, at any given step the move producing the maximum gain may result in a deterioration (gain may be negative). However, a sequence of moves with a net positive gain may be enabled by considering individual moves with negative gain. Thus, the exploration technique provides for hill-climbing in order to avoid local maxima.

V. Experimental Results

In this section, we illustrate the benefits of using the exploration techniques presented in this paper on some example systems. The experiments are aimed at measuring performance improvement obtained by using an optimized mapping of components to channels versus commonly used conventional bus architectures, and more naive solutions. We also report CPU time consumed by the exploration tool while generating these solutions.

To demonstrate the effectiveness of our exploration technique, we conducted experiments on two example systems. The first is a cell forwarding unit of an output queued ATM switch. The `ATM` system consists of 8 output ports, each with a local queue of cell headers. The system also has three shared memory banks, to store the arriving cell payloads. Each port periodically polls its queue to detect presence of a cell. When non-empty, it issues a *dequeue* signal to its local queue, extracts the relevant cell from the appropriate shared memory and sends it onto its output link. The second system (`SYS`) is the one discussed in Section II.

Each system was specified as a set of concurrent communicating tasks, with communication modeled as the exchange of abstract communication events. HW/SW partitioning and mapping was performed using the POLIS [22] framework, and system level simulation was carried out in PTOLEMY [23]. The resulting simulation traces were used in the subsequent communication architecture analysis and exploration algorithms. For the `SYS` example, the template architecture consisted of two buses connected by a bridge with specified parameters (width, speed, *etc.*) as shown in Figure 1(b). For the `ATM` example, the topology consisted of three buses interconnected by 2 bridges.

Table 3 reports the performance of each of the two systems under various communication architecture choices. The rows in Table 3 correspond to the following architectures: in row 1, all components are mapped to a single shared system bus; in row 2, the mapping of components to channels and protocol parameters

Table 3: Experimental results

	ATM			SYS		
Case	Performance (cycles)	Speedup	CPU time (seconds)	Performance (cycles)	Speedup	CPU time (seconds)
shared	24654	1.00	10.3	32328	1.00	6.8
random	15314	1.61	11.3	25593	1.26	7.0
initial	11723	2.10	12.1	19998	1.62	6.7
opt	9242	2.67	23.5	18139	1.78	11.8
Abstract comm .	4992	4.94	138	9988	3.24	134

are chosen at random; in row 3, the mapping and protocols are determined by the initial solution (note, that this is the result of a selection algorithm that ignores access conflicts); in row 4, the mapping and protocols are optimized by the proposed exploration procedure, including the iterative refinement; in row 5, the communication architecture topology is an abstract one, with infinite concurrency and bandwidth (this is a loose lower bound on the system execution time.

For each system, in columns 2 and 5, Table 3 reports the actual performance measurement in terms of the number of clock cycles taken to accomplish a given task. In the case of SYS this was the time taken to process 2000 input stimuli, and for the ATM example, it was the time taken to process 1000 cells. In columns 3 and 6, Table 3 reports performance of each configuration as the speed up relative to the case when all communication goes through a shared bus (row 1). Columns 4 and 7 report CPU times with the following interpretations. In rows 1 and 2, (shared bus and random), the time spent in design space exploration is zero (the solution is pre-determined). Hence the reported CPU times indicate the time spent on a single evaluation of the system performance using the tool presented in [18, 19]. In row 3 (initial), CPU time includes the time spent in performance analysis as well as constructing the initial solution. In row 4 (optimized solution) it is the sum of the times spent in analysis, construction of the initial solution and the iterative procedure. Finally, for the last row, CPU time indicates the time required to generate the initial system execution trace via HW/SW co-simulation (performed only once for each system), since no subsequent performance analysis was necessary.

From this table we observe the following:

- Performance of each system under the optimized mapping is superior to any of the other solutions. In particular, for the ATM example, it is *2.67 times faster* than the commonly used single shared bus architecture, and 27% faster than one designed using a naive approach that ignores conflicts.
- The CPU times consumed in the exploration algorithms are dominated by the time spent in performance analysis. This is clear because the times reported in rows 1, 2 and 3 are roughly the same. Also, CPU time reported in row 4 is roughly double the CPU time reported in the previous rows, owing to an extra invocation of the performance analysis tool during the iterative improvement step.
- Using co-simulation as a performance analysis tool in a design space exploration methodology would not be feasible, since the large cost of a simulation would be encountered at each and every design space point that needs to be examined. In comparison, our performance analysis technique is over an order of magnitude faster than complete system simulation.

VI. Conclusions

In this paper we presented a new methodology and a set of algorithms to help system designers automatically create an optimal mapping of their system's communication needs to a target template communication topology. We presented examples to illustrate the need for such automated design space exploration for application specific system-chips, described the issues that need to be addressed while designing such a framework, and the difficulty of the problem in general. Experimental results indicate that our methodology performs well, generating solutions that provide significant performance improvement over more ad hoc techniques. Also our methodology is efficient, due to use of efficient performance analysis to drive the exploration algorithms.

References

[1] D. D. Gajski, F. Vahid, S. Narayan and J. Gong, *Specification and Design of Embedded Systems*. Prentice Hall, 1994.

[2] R. Ernst, J. Henkel, and T. Benner, "Hardware-software cosynthesis for microcontrollers," *IEEE Design & Test Magazine*, pp. 64–75, Dec. 1993.

[3] T. B. Ismail, M. Abid, and M. Jerraya, "COSMOS:A codesign approach for a communicating system ," in *Proc. IEEE International Workshop on Hardware/Software Codesign*, pp. 17–24, 1994.

[4] A. Kalavade and E. Lee, "A globally critical/locally phase driven algorithm for the constrained hardware sowftware partitioning problem ," in *Proc. IEEE International Workshop on Hardware/Software Codesign*, pp. 42–48, 1994.

[5] P. H. Chou, R. B. Ortega, and G. B. Borriello, "The CHINOOK hardware/software cosynthesis system ," in *Proc. Int. Symp. System Level Synthesis*, pp. 22–27, 1995.

[6] B. Lin, "A system design methodolgy for software/hardware codevelopment of telecommunication network applications ," in *Proc. Design Automation Conf.*, pp. 672–677, 1996.

[7] B. P. Dave, G. Lakshminarayana, and N. K. Jha, "COSYN: hardware-software cosynthesis of embedded systems ," in *Proc. Design Automation Conf.*, pp. 703–708, 1997.

[8] P. Knudsen and J. Madsen, "Integrating communication protocol selection with partitioning in hardware/software codesign ," in *Proc. Int. Symp. System Level Synthesis*, pp. 111–116, Dec. 1998.

[9] T. Yen and W. Wolf, "Communication synthesis for distributed embedded systems ," in *Proc. Int. Conf. Computer-Aided Design*, pp. 288–294, Nov. 1995.

[10] J. Daveau, T. B. Ismail, and A. A. Jerraya, "Synthesis of system-level communication by an allocation based approach ," in *Proc. Int. Symp. System Level Synthesis*, pp. 150–155, Sept. 1995.

[11] M. Gasteier and M. Glesner, "Bus-based communication synthesis on system level ," in *ACM Trans. Design Automation Electronic Systems*, pp. 1–11, Jan. 1999.

[12] R. B. Ortega and G. Borriello, "Communication synthesis for distributed embedded systems ," in *Proc. Int. Conf. Computer-Aided Design*, pp. 437–444, Nov. 1998.

[13] "Sonics Integration Architecture, Sonics Inc." http://www.sonicsinc.com.

[14] "IBM On-chip CoreConnect Bus Architecture." http://www.chips.ibm.com/products/coreconnect/index.html.

[15] J. A. Rowson and A. Sangiovanni-Vincentelli, "Interface Based Design ," in *Proc. Design Automation Conf.*, pp. 178–183, June 1997.

[16] K. Hines and G. Borriello, "Optimizing Communication in embedded system cosimulation ," in *Proc. International Workshop on Hardware/Software Codesign (codes/CASHE)*, pp. 121–125, Mar. 1997.

[17] S. Dey and S. Bommu, "Performance analysis of a system of communication processes," in *Proc. Int. Conf. Computer-Aided Design*, pp. 590–597, Nov. 1997.

[18] K. Lahiri, A. Raghunathan, and S. Dey, "Fast performance analysis of bus-based system-on-chip communication architectures," in *Proc. Int. Conf. Computer-Aided Design*, pp. 566–572, Nov. 1999.

[19] K. Lahiri, A. Raghunathan, and S. Dey, "Performance analysis of systems with multi-channel communication architectures," in *Proc. Int. Conf. VLSI Design*, pp. 530–537, Jan. 2000.

[20] "Peripheral Interconnect Bus Architecture." http://www.omimo.be.

[21] B. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs," *Bell Systems Technical Journal*, vol. 49, pp. 291–307, 1970.

[22] F. Balarin, M. Chiodo, H. Hsieh, A. Jureska, L. Lavagno, C.Passerone, A. Sangiovanni-Vincentelli, E. Sentovich, K. Suzuki and B. Tabbara. , *Hardware-software Co-Design of Embedded Systems: The POLIS Approach*. Kluwer Academic Publishers, Norwell, MA, 1997.

[23] J. Buck and S. Ha and E. A. Lee and D. D. Masserchmitt, "Ptolemy: A framework for simulating and prototyping heterogeneous systems," *International Journal on Computer Simulation, Special Issue on Simulation Software Management*, vol. 4, pp. 155–182, Apr. 1994.