In System on Chip for Real-time Systems. Kluwer Academic Publishers, To appear.

# SYSTEM SYNTHESIS FOR OPTICALLY-CONNECTED, MULTIPROCESSORS ON-CHIP

Neal K. Bambha and Shuvra S. Bhattacharyya
*Department of Electrical and Computer Engineering, and Institute for Advanced Computer Studies University of Maryland, College Park*

**Abstract**: Optical interconnects are being considered as a possible solution to the well-known problems of scaling in VLSI interconnects. Along with enabling higher speed interconnects, optics allows the construction of highly connected and irregular networks that are streamlined for particular applications. Using these networks, it is possible to implement application mappings that allow flexible, single-hop communication patterns between processors. This has advantages for reduced system latency and power. Such optically connected multiprocessors are particularly promising for embedded digital signal processing (DSP) applications, which are highly parallel, and typically have tight constraints on latency and power consumption. This paper addresses novel trade-offs involving communication routing flexibility, power consumption, and performance that arise in the context of system synthesis of optically-interconnected multiprocessors. We report on experimental results that expose these trade-offs, and propose systematic techniques to address them efficiently. We demonstrate the performance of these techniques on several benchmark examples.

**Key words**: optical interconnect, scheduling, interconnect synthesis, multi-hop communication, low power

## 1. INTRODUCTION

In recent years, optics has played an increasing role in multiprocessor systems. Various studies have predicted that the energy consumed by data communication will ultimately limit the processing speed in electronic processors [8], [10]. Light signals do not suffer from effects such as

electromagnetic interference and capacitive effects, which limit electrical interconnects. Many research groups have demonstrated optically-connected multiprocessor systems (e.g., see [4], [5], [7]). However, relatively little work has been done to develop synthesis techniques and automated mapping tools to take advantage of such systems. A key advantage of optics for a multiprocessor system is that it allows highly parallel data links and a large degree of connectivity between processors. This work addresses some fundamental issues related to mapping of applications onto optically-connected systems, particularly for generating low-latency, low-power schedules.

The problem of generation of communication topologies at the system level has been examined for bus-based implementations [3], [9]. These techniques try to optimize communication between a set of processes, and try to minimize the total communication costs. Many techniques have also been presented for scheduling application graphs onto multiprocessor networks with various fixed topologies (e.g., see [2]). Our work is different because we incorporate communication topology synthesis, limited hop communication, and optical interconnects which are based on unidirectional point-to-point links and give rise to new issues of schedule deadlock and communication flexibility.

## 2. OPTICALLY CONNECTED SYSTEMS

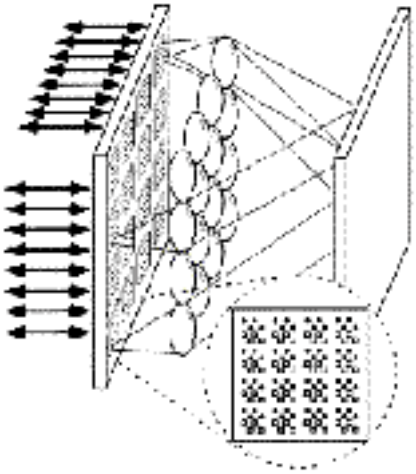

*Figure 1.* [FAST-Net prototype system]

One example of an optically connected system is the *FAST-Net* prototype [5], a high-throughput data-switching concept that uses a reflective optical

system to globally interconnect a multi-chip array of processors. The three-dimensional optical system links each chip directly to every other with a dedicated bidirectional parallel data path. This is illustrated in *Figure 1* [5]. Initiatives such as the DARPA VLSI Photonics Program [13] are pushing this technology from multi-chip arrays of processors towards single-chip arrays of processors.

## 3. TASK GRAPH SCHEDULING

The computational model used in this work is that of conventional acyclic *task graphs*, in which graph vertices (*tasks* or *nodes*) represent computations and each edge represents the communication of a packet of data from the source task to the sink task. Henceforth in this paper, we refer to a task graph representation of an application as an *application graph*. A vast range of scheduling techniques for task graphs has been developed (e.g., see [12] for a review of several representative approaches); however, these techniques typically assume a fixed communication network, and do not systematically take connectivity constraints into account. By connectivity constraints, we mean the inability of certain pairs of processors to communicate with each other. These constraints arise because it is advantageous to configure the schedules in such a way that multi-hop communication is minimized, and the relative abundance of communication links is used instead to achieve the required communication flexibility.

One contribution of this paper is to develop a general framework for extending arbitrary list scheduling approaches to avoid deadlock, and operate efficiently in the presence of connectivity constraints. We also apply this framework to jointly streamline the communication network and task graph mapping for a given application. This can be used both for minimum-cost dedicated implementations, and for reconfigurable networks, where the goal is to save power consumption by activating a minimal subset of available laser-detector pairs.

## 4. COMMUNICATION FLEXIBILITY

One major consequence of single-hop communication schedules is that each processor $p$ is restricted to send data to a subset of the set of all processors $\Phi$, and to receive data from a (possibly different) subset of $\Phi$. If these constraints are not considered, deadlock can easily occur during the scheduling process. We define a *feasible set* of processors $F[\nu]$ for a task $\nu$ as the largest subset of $\Phi$ on which $\nu$ can be scheduled without

deadlock. We have developed an efficient algorithm to determine the feasible set of processors $F[v]$ for all $v \in G$ at any point during the scheduling process.

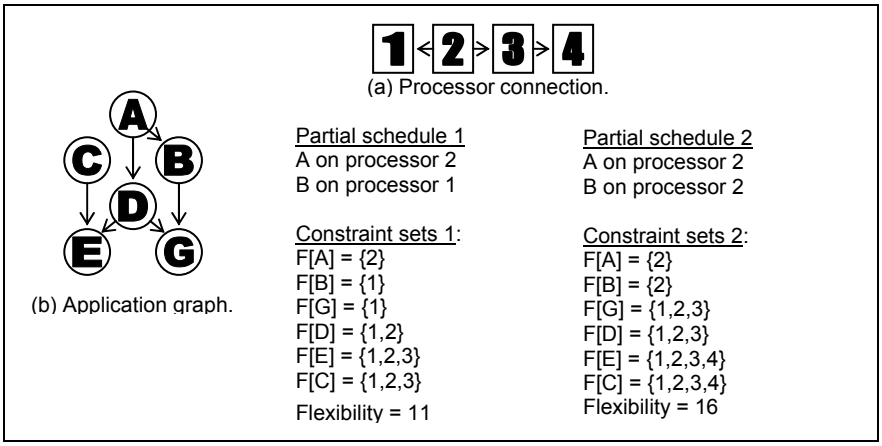


*Figure 2.* [Example demonstrating flexibility metric.]

We define the *communication flexibility* (or simply *flexibility* for short) of the system at any point during the scheduling process as the sum of the sizes of $F[v]$ for all $v \in G$. The flexibility gives some measure of the degree of constraint imposed on all tasks by a given scheduling move. Figure 2 depicts a simple example of an application graph with six tasks scheduled on four processors.

## 5. SINGLE-HOP SCHEDULING FOR LOW POWER

The general class of list scheduling algorithms can easily be adapted to produce single-hop (or n-hop) schedules by incorporating our constraint algorithm. This is advantageous because it allows us to leverage a large library of useful scheduling techniques.

In list scheduling, a priority list L of tasks is constructed. The priority list is a linear ordering $(v_1, v_2, \ldots, v_{|V|})$ of the tasks in the application graph $G = (V, E)$ such that for any pair of distinct tasks $v_i$ and $v_j$, $v_i$ is to be given higher scheduling priority than $v_j$ if and only if $i < j$. Each task is mapped to an available processor as soon as it becomes the highest-priority task according to L among all tasks that are ready. This process is repeated until all tasks are scheduled.

The concepts of feasibility and flexibility can be incorporated into the general framework of list scheduling by restricting the set of candidate pro-

cessors to include only those that are feasible at the given scheduling step, and by taking flexibility into account in designing the priority metric through which tasks are ordered.

Single-hop communication is desirable for low power, since every communication hop incurs an additional energy cost. However, restricting the communication to single-hop reduces the flexibility of the scheduler to fewer possible moves in each scheduling step. In general, this would not be optimal for producing the lowest makespan schedules. We show in Section 7 that restricting the schedule to single-hop communication can produce significant power savings when compared to multi-hop schedules, while not penalizing the makespan significantly.

If the communication is not constrained to be single hop, some routing algorithm must be employed to schedule the interprocessor communication. In our experiments, we used a greedy load-based minimal routing algorithm. Minimal routing algorithms consider only shortest paths between processors. Since each communication hop in an optical interconnect incurs an energy cost, shortest path routes are desirable. A weight is assigned to each edge on all shortest paths as a function of the change in the relative load that would occur if it were to be used by the route. A shortest path from source processor to destination processor is then computed with respect to those weights. Such algorithms are often employed in multiprocessor systems [1]. Although this routing algorithm runs in polynomial time [1], in the context of a fixed optimization budget, the time spent computing the routes in a multi-hop schedule reduces the number of possible scheduling moves that the scheduling algorithm can explore when compared to a single-hop schedule.

## 6. TPLA ALGORITHM

Realistic optical networks may incorporate relatively high, but not necessarily complete (fully connected), levels of connectivity. Even in fully-connected systems, such as *FAST-Net* [5], it is still desirable from the viewpoint of power and heat dissipation to have a minimal interconnect mapping, since for a given application, non-essential transmitters can be turned off. In this section, we illustrate how our single-hop scheduling strategies, and the underlying concept of communication flexibility, can be used to guide the synthesis of application-specific interconnect structures. The main idea here is that for embedded multiprocessors, the interconnect topologies should be driven by the specific application mappings that will execute across them, and jointly designing the two is advantageous.

Specifically, we have developed a greedy, heuristic algorithm, called the *two-phase link adjustment* (TPLA) algorithm, to synthesize an interconnect and an associated multiprocessor schedule for a given application. The TPLA algorithm starts with a fully connected network, and operates in *down* and *up* phases. Input to the algorithm is either a makespan constraint for the application, or a constraint on the total number of links.

Each step of the down phase in TPLA removes one link, while each step of the up phase adds one link. One step of the down phase consists of assigning each existing link a score based on the schedule makespan resulting from its removal, and removing the link with the lowest score. A history of scores is kept for each link. For the first pass through the down phase, ties between links are broken randomly. On subsequent passes, the link history is used to break ties. The down phase continues until all the links are removed.

Conversely, one step of the up phase in TPLA consists of assigning a score to each missing link based on the makespan resulting from its addition. The up phase continues until the network is fully connected. Repeated, alternating invocations of down and up phases are executed for some time limit (determined by the user), and the best result found is taken. Given a makespan constraint, this best result minimizes the number of links. Alternatively, given a constraint on the number of links, the best result minimizes the makespan.

## 7. EXPERIMENTS

Our techniques for scheduling and interconnection pattern synthesis operate in conjunction with a given list scheduling strategy. In these experiments, we employed the DLS algorithm [11] as the underlying list scheduling strategy, although, as described in Section 5, any list scheduling algorithm could have been used.

We examined a set of DSP application benchmarks and scheduled them using two different scheduling modes, one that incorporates only feasibility information (to avoid deadlock), and another that takes both feasibility and flexibility into account. We refer to these as the *feasibility-only* and *feasibility-flexibility* modes, respectively. To evaluate the performance across a range of connectivity levels, we scheduled the applications onto networks with varying degrees of connectivity. We also compared the makespan and power of single-hop schedules to that of multi-hop schedules. For the multi-hop scheduling, we used the greedy load-based minimal routing algorithm discussed earlier.

In the feasibility-only mode, the processor P considered for a given task $v$ at each scheduling step was restricted to be in the feasible set $F[v]$ for $v$, as described earlier, and no modification was made to the task prioritization metric of the underlying list scheduling strategy (DLS).

In the feasibility-flexibility mode, the processor P considered at each scheduling step was again restricted to be in the feasible set for $v$; however, whenever two processor assignments for $v$ resulted in equal priority levels $L(v)$, where $L$ represents the priority metric of the original DLS algorithm, priority was given to the assignment that resulted in a higher value of flexibility. In other words, priority was given to assignments that offered greater flexibility for future scheduling decisions.

For each application, we chose a number $N$ of processors, and then generated a fully connected network with $N(N-1)$ links. We scheduled the application using both feasibility-only and feasibility-flexibility modes onto this network. Next we removed one link from the network at random, and again scheduled the application using both scheduling modes. We continued this process of removing links until no links remained, resulting with all the tasks scheduled on a single processor. We define the relative improvement of the feasibility-flexibility mode over the feasibility-only mode by comparing the average makespan over all points.

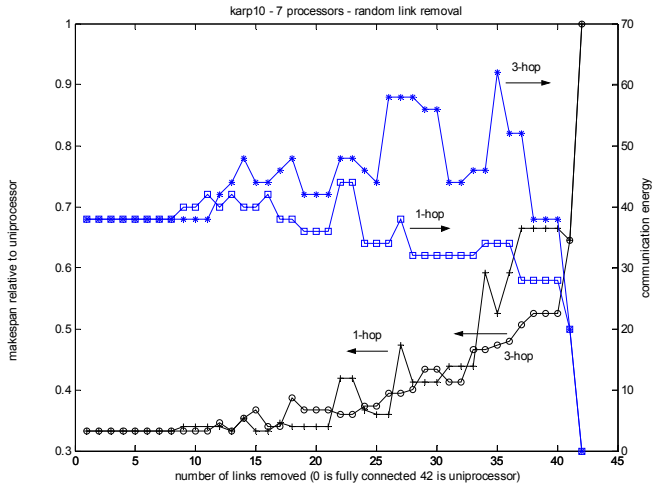


*Figure 2.* [Makespan and communication energy with single-hop and three-hop maximum communication]

Figure 2 compares the makespan and communication energy for single-hop and three-hop communication for the karp10 application. For this application, the single-hop schedule required 24% less communication

energy while producing on average a 4% longer makespan than the three-hop schedule. For this application, the makespan obtained without using the flexibility metric was 26% longer than the 1-hop makespan. Table 1 summarizes this comparison for several other DSP application benchmarks.

We performed experiments with the following application graphs: fft1, irr, fft3, karp10, qmf4, Laplace, sum1, and RBFNN. The fft graphs are different implementations of the fast Fourier transform [6]. Karp10 refers to the Karplus-Strong music synthesis algorithm with 10 voices (21 nodes); qmf4 is a quadrature mirror filter bank (14 nodes). Laplace is a laplace transform. Irr is an adaptation of a physics algorithm, and sum1 is an upside down binary tree representing the sum of products computation. RBFNN is a neural network classifier algorithm.

*Table 1.* [Relative makespan improvement obtained by using flexibility information in the scheduling process.]

| Application | N | 1-hop makespan improvement using flexibility | Reduction in communication energy | Makespan penalty for 1-hop vs. 3-hop |
|---|---|---|---|---|
| Fft1 | 7 | 30 | 16 | 8 |
| Karp10 | 6 | 26 | 24 | 4 |
| Irr | 8 | 17 | 16 | (2) |
| Qmf4 | 7 | 19 | 32 | 3 |
| Nn16-3-4 | 8 | 21 | 58 | 2 |
| Sum1 | 6 | 8 | 1 | 4 |
| Laplace | 7 | 23 | 4 | (3) |
| Fft2 | 7 | 15 | 12 | 2 |

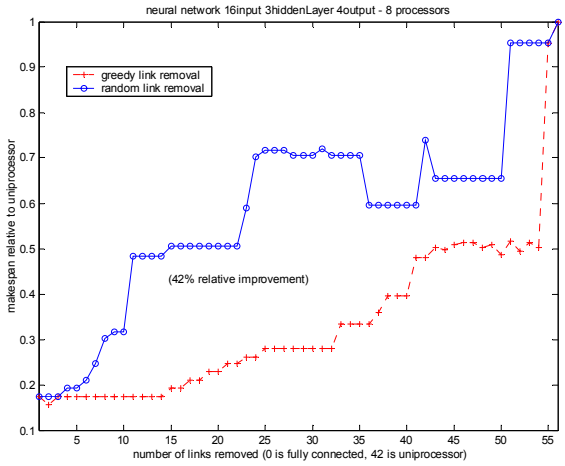


*Figure 3.* [Link synthesis using the TPLA algorithm.]

We evaluated the TPLA algorithm on the RFBNN benchmark described above. The bottom curve of Figure 3 shows the best makespan achieved for each level of connectivity between 0 and fully connected, after one "down" phase and one "up" phase. This gives a Pareto curve of the trade-off between number of links and makespan for the application. For purposes of comparison, the upper curve of Figure 3 shows the makespan achieved by starting with fully connected and randomly removing one link at a time. The TPLA algorithm shows a significant improvement (42% relative improvement) over random removal, and is thus a promising starting point for developing more sophisticated link synthesis methods. More broadly, it demonstrates the effectiveness of feasibility- and flexibility-driven scheduling in iterative co-design of optical interconnection structures.

## 8. CONCLUSIONS

Optical interconnect technology is promising for global communication in embedded multiprocessors, since the interconnection patterns can flexibly be streamlined and reconfigured to match the target applications. However, due to the power consumption characteristics of optical links, it is useful to restrict communication across them to single-hop transfers. We have demonstrated an effective algorithm for determining the set of feasible processors that will avoid schedule deadlock in a single-hop schedule, and a useful metric, called communication flexibility, for the degree to which a given scheduling decision constrains future decisions (in the context of the given communication topology).

We used this algorithm and the flexibility metric in conjunction with the DLS algorithm to map several DSP applications across a wide range of interconnect topologies. These experiments demonstrated that incorporating the flexibility metric into existing scheduling algorithms improves the schedule makespan, and that single-hop schedules can significantly reduce power consumption of the system. We also demonstrated that these scheduling techniques could be used to effectively guide an algorithm for jointly synthesizing the interconnection network together with the mapping of an application onto the network.

## ACKNOWLEDGEMENTS

This work was supported by the DARPA funded Optoelectronics Center for Innovative Photonic Chipscale Technologies (Contract number

MDA972-00-1-0023). We would like to thank Vida Kianzad for her compilation of the benchmark examples.

## REFERENCES

[1] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, O. Waarts, "On-Line Routing of Virtual Circuits with Applications to Load Balancing and Machine Scheduling," *Journal of the ACM*, vol. 44, no. 3, pages 486-504, May 1997.

[2] S. Banerjee, D. Picker, R. Fellman, and P. Chau, "Improved Scheduling of Signal Flow Graphs onto Multiprocessor Systems Through an Accurate Network Modeling Technique," *Proceedings of the International Workshop on VLSI Signal Processing*, pages 157-167, 1994

[3] M. Gasteier and M. Glesner, "Bus-Based Communication Synthesis on System-Level," *Proceedings of the 9th International Symposium on System Synthesis*, pages 65-70.

[4] P. S. Guilfoyle, "Digital Optical Computing Architectures for Compute Intensive Applications," *Proceedings 1994 International Conference on Optical Computing*, 1994.

[5] M. W. Haney, M. P. Christensen, P. Milojkovic, "Description and Evaluation of the FAST-NET Smart Pixel-Based Optical Interconnection Prototype," *Proceedings of the IEEE*, vol. 88, no. 6, June 2000.

[6] A. Kahn, C. McCreary, J. Thompson and M. McArdle, "A Comparison of Multiprocessor Scheduling Heuristics," *Proceedings of 1994 International Conference on Parallel Processing*, vol. II, pages 243-250, 1994.

[7] N. McArdle, M. Naruse, A. Okuto, and M. Ishikawa, "Implementation of a Pipelined Optoelectronic Processor: OCULAR II," *Technical Digest of Optics in Computing '99.*

[8] J. D. Meindl, "Low Power Microelectronics: Retrospect and Prospect," *Proceedings IEEE*, vol. 83, no. 4, 1995.

[9] S. Narayan and D. Gajski, "Synthesis of System Level Bus Interfaces," *Proceedings of European Design and Test Conference '94*, pages 395-399, Feb. 1994.

[10] P.W. Smith, "On the Physical Limits of Digital Optical Switching and Logic Elements," *Bell System Technical Journal*, vol. 61, no. 8, 1982.

[11] G. C. Sih, *Multiprocessor Scheduling to Account for Interprocessor Communication, Ph.D. Thesis*, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, April 1991.

[12] S. Sriram and S. S. Bhattacharyya. *Embedded Multiprocessors: Scheduling and Synchronization*. Marcel Dekker, Inc., 2000

[13] http://www.darpa.mil/MTO/VLSI.