Data Path Management in Mesh-Based Programmable Routers

Qiang Wu and Tilman Wolf
Department of Electrical and Computer Engineering
University of Massachusetts, Amherst, MA, USA
Email: {qwu,wolf}@ecs.umass.edu

***Abstract*—With dozens to hundreds of processing cores deployed in next generation packet processor, regular topologies such as mesh are widely adopted in Network-on-Chip design to provide scalable interconnection to cores. Although such packet processors are rich in raw system processing power, utilization of hardware resource plays a critical role in overall system performance. In this paper, we focus on processing task mapping and on-chip packet routing, which are the key issues for data path performance on next-generation packet processors. We present a genetic algorithm to explore the assignment of tasks, and utilize on-chip interconnections by splitting the traffic between cores across multiple paths. The split flow traffic assigned to each routing path is solved with linear programming. Our experimental results on a packet processor architecture prototype show that the proposed algorithm is efficient and scalable.**

***Index Terms*—network processor, runtime management, task allocation.**

I. Introduction

The functionality of a data communication network is characterized by the operations that are performed on its routers. In particular, data path processing encompasses operations that are performed on all network traffic. (Control path processing, which includes routing protocols, is less critical from a performance perspective and thus beyond the focus of our work.) Conventional routers in the Internet implement very simple packet processing functions in the data path that meet the requirements for Internet Protocol (IP) version 4. However, practical considerations demand that numerous additional functions be supported. Examples of such functions are: multicast, load balancing, security, classification, measurement, etc. In practice, the packet processing functions change over time (e.g., improved load balancing algorithm, new flow classification). Therefore, it is important that router use packet processing systems that can be reprogrammed after deployment to support these new functions.

Programmable packet processing systems are typically implemented with specialized embedded multi-processor system on a chip (MPSoC). These network processors often contain dozens to hundreds of simple processing cores [1], [2]. This high level of parallelism is particularly suitable for processing in the networking domain, which is characterized by high levels of inherent parallelism and simplicity.

One of the key challenges in making programmable packet processing systems a reality is their management during runtime. To achieve the necessary levels of parallelism and performance, network processors use very simple processor cores that cannot run conventional operating systems. Instead, lightweight runtime systems are used for managing processing tasks [3], [4], [5]. These runtime systems need to adapt to changing traffic workloads and reallocate processing tasks as workloads change.

The problem that we address in this paper is: *How can we efficiently allocate processing tasks to processing resources to achieve high throughput performance on a packet processing system?* Packet processing operations are presented as task graphs that consist of basic processing steps (e.g., Click elements [6]) and connections that indicate functional dependencies. The processor system is assumed to be a mesh interconnect as proposed in [7]. This problem is challenging since processing and communication is tightly coupled in a network processor. Bottlenecks in either system component lead to a significant degradation in throughput performance. In addition, task allocation systems need to support dynamic adaptation as workload requirements change.

Our work explores algorithm design for solving this task allocation problem. Specifically, our contributions are:

- A genetic algorithm for task scheduling on network processing systems with a mesh interconnect,
- An algorithm for multi-path routing of traffic flows on the interconnect, and
- Evaluation results that explores system throughput performance for these algorithms.

The remainder of the paper is organized as follows. Section II discusses related work. Section III introduces system architecture of packet processor and programming abstraction. The processing tasks mapping problem and on-chip packet routing are discussed in Section IV and Section V, respectively. Section VI presents mapping algorithm for data path management. Evaluation results are introduced in Section VII. Section VIII concludes this paper.

II. Related Work

Scalable on-chip interconnection has been studied as an aspect of system-on-a-chip design in many different ways, ranging from shared bus with dedicated inter-processor communication paths (e.g. Intel IXP [1]) to routers with more sophisticated interconnection (e.g. Tilera Tile64 [8]). With even thousands of cores embedded on a single chip, on-chip interconnections that scale close to linearly with system size present a firm demand for architecture design [9]. To achieve large scalability, the idea of routing packets instead of wires is borrowed from packet switched networks [10].

Mesh structure and its variants have been extensively researched and implemented on on-chip interconnection due to its elegant mathematical properties and scalability [11], [8], [12], [13]. The performance and power consumption of packetized on-chip interconnection has been analyzed by Ye et al. [14]. Hu et al. [15] attempted flexible routing of communication among on-chip IPs with minimized energy consumption. The mapping of processing cores onto mesh-based NoC under bandwidth constrains is addressed in [16]. Both of these approaches represent static mapping of hardware resource to application processing and communication. More flexible and adaptive mapping methodology however, is required in network domain as processing and communication requirement for applications varies with network traffic that excises the system.

On systems with general programmability, it is important to consider programming abstraction that can be used to simplify software development [17], or serves as basic scheduling unit [5], [18]. These work focus more on processing aspect of tasks than communication. Therefore, they are not directly applicable to recent approaches in parallel network processing platforms.

The general problem of task mapping to processing cores has been modeled with “bin packing” problem and identified as NP-Hard. Therefore heuristic algorithms have been proposed and compared for specific application domains [19]. Genetic algorithm, as a general approach for solving complex problems, has been introduced in this area [20], [21], [22]. Our work differs from these works as it focuses on not only processing task mapping, but providing deadlock-free routing under interconnection bandwidth constrains.

III. Packet Processing System

The design of packet processing systems is centered on packet processors, which provide hardware resources for both processing and storage that are requested by each packet in network traffic. To effectively manage data path hardware resource allocation and allow system users to develop new functionalities, programming abstraction is also required for such systems. In this section, we first discuss the architecture of our proposed packet processor design[1]. Then we introduce the programming abstraction for the system.

A. System architecture

The key idea behind packet processing system design is to utilize the inherent inter-packet parallelism present in network traffic. Figure1 shows the overall architecture. Packets that enter the system through input interface are classified by flow and dispatched to processing grid. Classification and dispatch units in data path can be parallelized to better utilize inter-packet parallelism. The control processor governs flow classification, processing tasks allocation to packet processors, and packet routing inside processing grid.

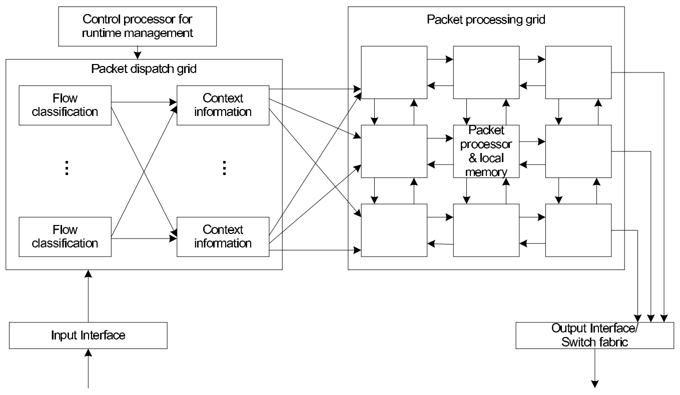

Fig. 1. Packet processing system architecture

Processing grid is where flow-specific functionalities are performed on each corresponding packet. Inside the grid, packet processors are interconnected by mesh network. Details of packet processor design are discussed in [7]. In this work, we focus on utilization of overall processing grid, which is characterized by *Architecture Graph*:

Definition 1. The architecture graph is a directed graph, $A(P, C)$ with each node $p_i \in P$ representing a processor and each interconnection $c_{i,j} \in C$ representing communication from processor p_i to processor p_j. The weight of processor p_i denoted by $w(p_i)$ represents the processing capacity of p_i in million instructions per second. The number of hardware threads that processor p_i supports is denoted by $H(p_i)$. The weight of interconnection $c_{i,j}$, denoted by $b(c_{i,j})$, represents communication bandwidth of $c_{i,j}$ in packets per second.

B. Programming Abstraction

In general, network processing function defines a sequence of operations that can be performed on input packets. It is nature to represent such functions with task graph in which tasks denotes operation sets and directed edges denotes communication among tasks. The main difference between programming abstractions in this domain is granularity (e.g. basic functional blocks [6], individual instructions [4] etc.). The partition of functions to tasks has been exploited in detail in [23]. In this paper, we choose the granularity similar to Click, which is suitable for per-thread instruction store in packet processor.

Figure 2 shows an example of task graph. Each flow requires services from a set of network functions in certain sequence. When the first packet of a new flow enters system, flow management running in control processor determines required tasks and builds corresponding task graph. Tasks can be shared by different flows (e.g. t_3 in Figure 2). We formulate task graph definition as following:

Definition 2. The task graph is a directed graph, $G(T, E)$ with each node $t_x \in T$ representing an application and each edge

[1]Our work on runtime resource allocation can be applied to any packet processor with parallel processing cores grid interconnected by regular NoC topologies.

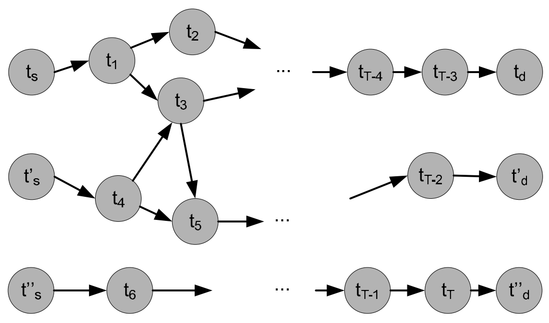

Fig. 2. Task Graph

$e_{x,y} \in E$ representing communication from task t_x to task t_y. The weight of task t_x denoted by $w(t_x)$ represents the processing requirement of t_x in million instructions per second. The weight of edge $e_{x,y}$, denoted by $r(e_{x,y})$, represents communication bandwidth requirement of $e_{x,y}$ in packets per second.

IV. Processing Task Mapping

The processing requirements represented by task graph need to be assigned to processing cores. Such assignment presents the key challenge in translating raw system processing power into desired packet throughput. In order to obtain high quality mapping, data path resource management needs to determine processing requirement of each task quantitatively to let the mapping be guided by system processing capacity.

In this section, we first present how to estimate task requirement and system processing capacity. In order to loosen the capacity limitation without hardware re-engineering, we then introduce a method called task duplication proposed in our prior work. And finally we formulate the task mapping problem.

A. Task Processing Requirement Estimation

The processing requirement of a task is determined by its processing time and the network traffic that needs to be processed by the task. Processing time is determined by task's functionality, and is often related to packets under processing (e.g. packets' length, content and etc.). To capture accurate processing time of a task, runtime profiling technique becomes necessary. However, in practical systems, we can assume the processing time distribution of a task is not time-dependent and matches its empirical observation. Therefore, we can use average processing time to estimate task workload.

Let $S(t_y)$ represent the average processing time of task t_y in "million instructions per packets." The requirement of t_y can be obtained by:

$$w(t_y) = S(t_y) \times \sum_{e_{x,y} \in E} r(e_{x,y}) \tag{1}$$

B. System Processing Capacity

The system processing capacity follows two dimensions:

- Single processor capacity: Any task t_x should not overload the processor p_i it is assigned to (i.e. $w(t_x) \leq w(p_i)$).
- Overall processing capacity: Accumulated task processing requirement should not exceed overall system processing capacity (i.e. $\sum_{t_x \in T} w(t_x) \leq \sum_{p_i \in P} w(p_i)$)

However, as tasks' processing requirements may vary significantly on practical systems [5], it is possible that a single task can overload an entire processor. We therefore use task duplication to address this issue.

C. Task Duplication

Task duplication creates additional instances of tasks that are fully connected to the same predecessor and successor tasks of the original task. By splitting input packets to duplicated instances, the processing requirement of original task obtained by Equation 1 can be shared by all instances. Detailed discussion and algorithm for task duplication is explained in [23].

Applying task duplication on a task graph $G(T, E)$ results in an extended task graph $G'(T', E')$, which has more tasks than $G(T, E)$. As task mapping is a one-to-one assignment between tasks and hardware threads, we assume that when task duplication is applied, the number of tasks is no more than the number of hardware threads present in the system (i.e. $|T'| \leq \sum_{p_i \in P} H(p_i)$). For simplicity, in the rest of this paper, we use $G(T, E)$ to represent both task graph and extended task graph.

D. Mapping Problem Statement

Given an architecture graph $A(P, C)$ and a task graph $G(T, E)$, for each task $t_x \in T$ in $G(T, E)$, find a mapping:

$$m : T \rightarrow P, \text{ s.t. } m(t_x) = p_i, \forall t_x \in T, \forall p_i \in P \tag{2}$$

under the constraints:

$$\sum_{m(t_x)=p_i} \leq H(p_i), \forall t_x \in T, \forall p_i \in P \tag{3}$$

$$\sum_{m(t_x)=p_i} w(t_x) \leq w(p_i), \forall t_x \in T, \forall p_i \in P \tag{4}$$

Equation 2, 3 and 4 form the processing task mapping problem, which focuses on processing demand of network functions. However, the assignment of tasks to parallel processors also needs to be guided by inter-processor communication. In the next section, we discuss adaptive on-chip packet routing for parallel programmable routers. The detailed genetic algorithm for solving task mapping problem is presented in Section VI.

V. On-Chip Packet Routing

The mesh topology naturally utilizes two dimensional on-chip area. On-chip mesh networks also benefit from its elegant mathematical properties and inherent flexible communication patterns. Inside a mesh, each processor is only directly connected to a few others. Communication between non-neighbor processors is therefore implemented by routing messages inside the network. Key characters of high throughput communication in such a network include the ability to avoid deadlock and traffic splitting.

A. Deadlock-Free Routing

Deadlock avoidance in mesh topology has been studied in the area of wormhole routing. Glass et al. [24] proposed a turn model, which prohibits certain turns in order to break all possible cycles in any routing path. By prohibiting two designated types of turns of all eight types, three deadlock-free routing algorithms were proposed: West-First, North-Last and Negative-First. Instead of restricting specific types of turns on all mesh intersections, Chiu [25] proposed Odd-Even turn model, which restricts certain types of turns at some locations to avoid deadlock. In our work, we follow West-First turn model to construct multiple routing paths for each source-destination pair of processors.

In our work, we use the turn model to find all possible routing paths for each edge that connects two tasks mapped to processors. If no legal routing paths exist between two processors, the particular task mapping is considered to be invalid.

B. Multi-Path Routing

An edge $e_{x,y}$ in task graph represents communication demand $r(e_{x,y})$ from task t_x to t_y. After tasks are assigned to processors, there are multiple possible routing paths consisting of different interconnections exist between each pair or processors. Let $L_{m(t_x),m(t_y)}$ be the set of candidate paths $l^k_{m(t_x),m(t_y)}, 1 \le k \le |L_{m(t_x),m(t_y)}|$ between processor $m(t_x)$ and $m(t_y)$, and $f^k_{m(t_x),m(t_y)}, 1 \le k \le |L_{m(t_x),m(t_y)}|$ be the bandwidth of flow in each path. We define the utilization of connection $c_{i,j}$ as:

$$u(c_{i,j}) = \sum_{e_{x,y} \in E} \sum_{k=1}^{|L_{m(t_x),m(t_y)}|} f^k_{m(t_x),m(t_y)} \times M^k_{x,y} \quad (5)$$

The values of $M^k_{x,y}$ are obtained as follows:

$$M^k_{x,y} = \begin{cases} 1 & \text{if } c_{i,j} \in l^k_{m(t_x),m(t_y)} \\ 0 & \text{otherwise} \end{cases}$$

Equation 5 obtains interconnection utilization by enumerating all edges that have a routing path contains the particular interconnection. It associates the bandwidth requirement from task graph with bandwidth capacity from architecture graph.

C. Routing Problem Statement

Given an architecture graph $A(P,C)$ and a task graph $G(T,E)$ with tasks mapped on processors, for each edge $e_{x,y}$ in the task graph, there exist deadlock-free routing path set $L_{m(t_x),m(t_y)}$ consisting of physical interconnections. Find the flow assignment $f^k_{m(t_x),m(t_y)}, 1 \le k \le |L_{m(t_x),m(t_y)}|$ to each possible path $l^k_{m(t_x),m(t_y)}, 1 \le k \le |L_{m(t_x),m(t_y)}|$, under constrains of:

$$\forall c_{i,j} \in C, u(c_{i,j}) \le b(c_{i,j}) \quad (6)$$

$$\forall e_{x,y} \in E, \quad \sum_{k=1}^{|L_{m(t_x),m(t_y)}|} f^k_{m(t_x),m(t_y)} = r(e_{x,y}) \quad (7)$$

Condition 6 means that utilization of any interconnection should *not* exceed its bandwidth. Condition 7 guarantees that bandwidth requirement of any edge in task graph is completely split.

Task mapping determines which processor a task is assigned to, thus has significant impact on routing. In the next section, we discuss the implementation of a genetic algorithm for task mapping, which integrates evaluation of on-chip multi-path routing.

VI. Data Path Management

The management of such parallel packet processing system focuses on resource allocation for processing and communication requirements.

A. System Operations

During system operation, the task graph is constructed dynamically at runtime. Each time a new connection request presents, tasks that provide services for the new connection and edges that interconnect tasks are added to task graph. Bandwidth requirements of edges can be directly derived from new connection request. Tasks' processing requirements can be obtained by Equation 1 in Section IV-A. For new connections, runtime management needs to upgrade mapping since old mapping may not provide enough hardware resource. However, mapping does not necessarily change when existing connections terminate.

B. Mapping Algorithm

Algorithm 1 Mapping Algorithm.

```
1: Read input architecture graph $A(P,C)$ and task graph $G(T,E)$
2: Generate initial mapping population
3: Evaluate fitness of each individual mapping
4: repeat
5:     Selection of individuals with best fitness
6:     Generate intermediate generation with crossover
7:     Generate next generation with mutation
8:     Evaluate fitness of each individual mapping
9: until Suffient fitness achieved or maximum number of iteration reached
10: return  Mapping with best fitness
```

The genetic algorithm for task mapping is demonstrated in Algorithm 1.

1) Initial Mapping Generation: A predefined number of mappings are generated in this phase. A new mapping chromosome is formed by randomly assigning a processor to each task. Identical mapping chromosomes are prohibited in order to achieve large diversity in the initial generation.

2) Fitness Evaluation: The fitness of a mapping chromosome follows four dimensions:

- Hardware thread capacity of processors: The thread capacity of each processor determines how many tasks the processor can support, as illustrated by Condition 3 in Section IV-D.
- Processing capacity of processors: The sum of processing requirements of tasks mapped on a processor should not overload the processor. This evaluation is illustrated by Condition 4 in Section IV-D.
- Deadlock avoidance in routing: Although we adopt deadlock-free turn model to enumerate multiple routing paths, there is a possibility that deadlock-free routing does not exist in randomly generated mappings.
- Bandwidth capacity of interconnections: Traffic assigned to each interconnection should not exceed the interconnection's bandwidth capacity. This evaluation is shown by Condition 6 in Section V-C.

Each mapping m is checked against the four criteria. For the first two processor related criteria, the total number of processors that violate each one is counted as $F_{1,m}$ and $F_{2,m}$ respectively. The deadlock detection is implemented with turn model, in which all deadlock-free routing paths for each edge in task graph are enumerated. The number of edges that do not have a valid routing path is counted as $F_{3,m}$. And finally, the interconnection bandwidth capacity check is carried by solving multi-path routing problem described in Section V-C with linear programming. We define a variable $F_{4,m}$ for this criterion as

$$F_{4,m} = \begin{cases} 0 & \text{if routing problem is solvable} \\ |E| & \text{otherwise} \end{cases}$$

The fitness $F(m)$ of a mapping m is therefore defined as $F(m) = \sum_{k=1}^{4} F_{k,m}$. Then, a rank-based roulette wheel selection scheme can be used for chromosome selection.

3) Crossover and Mutation: The crossover for two mapping chromosomes is performed by randomly choosing a cut-off position to divide both mapping arrays into two parts, then exchange the second parts of both mapping. The mutation operator works on single chromosome. For each task in a mapping, it alters the processor assignment randomly with a small probability.

VII. Evaluation

We evaluate the effectiveness of our algorithm for task mapping and multi-path routing through simulation and analysis.

A. Experimental Setup

Our experiment setup is based on PacketBench [26], on which we develop six typical network functions for RISC processor. Figure 3 shows the task graph of functions used

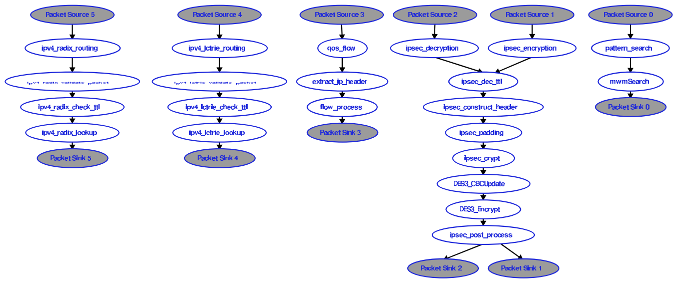

Fig. 3. Benchmark Network Applications

TABLE I
Network Function Processing Requirement.

Flow	Function	Task Workload (in instructions per packet)
1	IPsec Decryption	1859.66
2	IPsec Encryption	1859.66
3	String Match	3963.37
4	IPv4 Forwarding (Trie)	225.57
5	QoS	295.72
6	IPv4 Forwarding (Radix)	4973.67

in our evaluation. We apply network trace collected from the Internet uplink of our institutional network on PacketBench, and obtain average processing time for each task, as shown in Table I.

B. System Throughput

We develop a simulator, which generates particular or random new connection requests for network functions. Bandwidth of each connection is assumed to be 1 Kpps (or 1 Mbps for average packet length of 1000 bits). Figure 4 shows the maximum system throughput for two different processing grid configuration (i.e. 3×3 and 4×4) with various processor capacities.

It can be observed that the task mapping algorithm is scalable to processing capacity increase. The throughput increases linearly with processing capacity. Number of processors also has significant impact. On average, 4×4 configuration has 1.24x higher throughput than 3×3 configuration.

C. Interconnection Bandwidth Utilization

Maximum interconnection utilization relative to system throughput is shown in Figure 5. It can be observed that interconnection utilization is always far less than overall

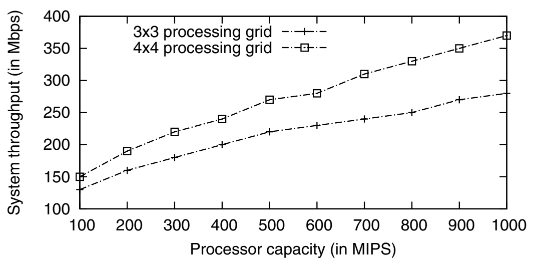

Fig. 4. System throughput

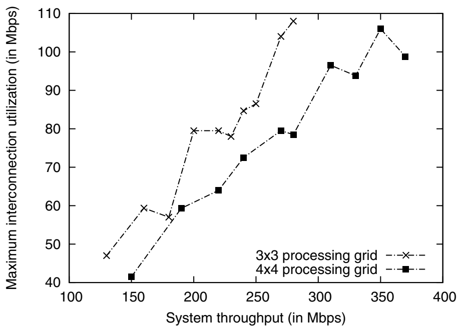

Fig. 5. Interconnection bandwidth

system throughput. This effect is due to multi-path routing effectively splits traffic across multiply paths.

For same throughput, configuration with more processors (thus more processing capacity) can host more duplicated tasks. Therefore the system workload and communication requirement is spread more evenly across the system, which leads to lower interconnection utilization. This result indicates scalability of our algorithm.

VIII. Conclusion

We have presented a genetic algorithm for task mapping and on-chip multi-path routing problem in mesh-based programmable router. For multi-core packet processing systems, such data path resource allocation is critical to system performance. Our mapping algorithm shows significant scalability and effectiveness in evaluation. As network processors scale towards higher bandwidths and more processor cores, it is necessary for task mapping algorithms to consider inter-processor communication. We believe that our work for a mesh-based packet processing system provide an important step in this direction.

References

[1] *Intel Second Generation Network Processor*, Intel Corporation, 2005, http://www.intel.com/design/network/products/npfamily/.

[2] *The Cisco QuantumFlow Processor: Cisco's Next Generation Network Processor*, Cisco Systems, Inc., San Jose, CA, Feb. 2008.

[3] R. Kokku, T. Riché, A. Kunze, J. Mudigonda, J. Jason, and H. Vin, "A case for run-time adaptation in packet processing systems," in *Proc. of the 2nd Workshop on Hot Topics in Networks (HOTNETS-II)*, Cambridge, MA, Nov. 2003.

[4] T. Wolf, N. Weng, and C.-H. Tai, "Run-time support for multi-core packet processing systems," *IEEE Network*, vol. 21, no. 4, pp. 29–37, Jul. 2007.

[5] Q. Wu and T. Wolf, "On runtime management in multi-core packet processing systems," in *Proc. of ACM/IEEE Symposium on Architectures for Networking and Communication Systems (ANCS)*, San Jose, CA, Nov. 2008.

[6] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," *ACM Trans. Comput. Syst.*, vol. 18, no. 3, pp. 263–297, 2000.

[7] Q. Wu and T. Wolf, "Design of a network service processing platform for data path customization," in *Proc. of The Second ACM SIGCOMM Workshop on Programmable Routers for Extensible Service of TOmorrow (PRESTO)*, Barcelona, Spain, Aug. 2009.

[8] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. Brown, and A. Agarwal, "On-chip interconnection architecture of the tile processor," *Micro, IEEE*, vol. 27, no. 5, pp. 15–31, Sept.-Oct. 2007.

[9] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick, "The landscape of parallel computing research: A view from berkeley," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2006-183, Dec 2006. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html

[10] W. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," in *Design Automation Conference, 2001. Proceedings*, 2001, pp. 684–689.

[11] S. Kumar, A. Jantsch, J.-P. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja, and A. Hemani, "A network on chip architecture and design methodology," in *VLSI, 2002. Proceedings. IEEE Computer Society Annual Symposium on*, 2002, pp. 105–112.

[12] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar, "A 5-ghz mesh interconnect for a teraflops processor," *IEEE Micro*, vol. 27, no. 5, pp. 51–61, 2007.

[13] A. O. Balkan, G. Qu, and U. Vishkin, "A mesh-of-trees interconnection network for single-chip parallel processing," in *ASAP '06: Proceedings of the IEEE 17th International Conference on Application-specific Systems, Architectures and Processors*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 73–80.

[14] T. Ye, L. Benini, and G. De Micheli, "Packetized on-chip interconnect communication analysis for MPSoC," in *Design, Automation and Test in Europe Conference and Exhibition, 2003*, 2003, pp. 344–349.

[15] J. Hu and R. Marculescu, "Exploiting the routing flexibility for energy/performance aware mapping of regular noc architectures," in *DATE '03: Proceedings of the conference on Design, Automation and Test in Europe*. Washington, DC, USA: IEEE Computer Society, 2003, p. 10688.

[16] S. Murali and G. De Micheli, "Bandwidth-constrained mapping of cores onto noc architectures," in *DATE '04: Proceedings of the conference on Design, automation and test in Europe*. Washington, DC, USA: IEEE Computer Society, 2004, p. 20896.

[17] N. Shah, W. Plishker, K. Ravindran, and K. Keutzer, "Np-click: A productive software development approach for network processors," *IEEE Micro*, vol. 24, no. 5, pp. 45–54, 2004.

[18] B. Chen and R. Morris, "Flexible control of parallelism in a multiprocessor pc router," in *Proceedings of the General Track: 2002 USENIX Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, 2001, pp. 333–346.

[19] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *J. Parallel Distrib. Comput.*, vol. 61, no. 6, pp. 810–837, 2001.

[20] S. M. Alaoui, O. Frieder, and T. A. El-Ghazawi, "A parallel genetic algorithm for task mapping on parallel machines," in *Proceedings of the 11 IPPS/SPDP'99 Workshops Held in Conjunction with the 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing*. London, UK: Springer-Verlag, 1999, pp. 201–209.

[21] C. Ravikumar and A. Gupta, "Genetic algorithm for mapping tasks onto a reconfigurable parallel processor," *Computers and Digital Techniques, IEE Proceedings -*, vol. 142, no. 2, pp. 81–86, Mar 1995.

[22] L. Wang, H. J. Siegel, V. R. Roychowdhury, and A. A. Maciejewski, "Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach," *J. Parallel Distrib. Comput.*, vol. 47, no. 1, pp. 8–22, 1997.

[23] Q. Wu and T. Wolf, "Dynamic workload profiling and task allocation in packet processing systems," in *Proc. of IEEE Workshop on High Performance Switching and Routing (HPSR)*, Shanghai, China, May 2008.

[24] C. J. Glass and L. M. Ni, "The turn model for adaptive routing," *J. ACM*, vol. 41, no. 5, pp. 874–902, 1994.

[25] G.-M. Chiu, "The odd-even turn model for adaptive routing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 11, no. 7, pp. 729–738, 2000.

[26] R. Ramaswamy and T. Wolf, "PacketBench: A tool for workload characterization of network processing," in *Proc. of IEEE 6th Annual Workshop on Workload Characterization (WWC-6)*, Austin, TX, Oct. 2003, pp. 42–50.